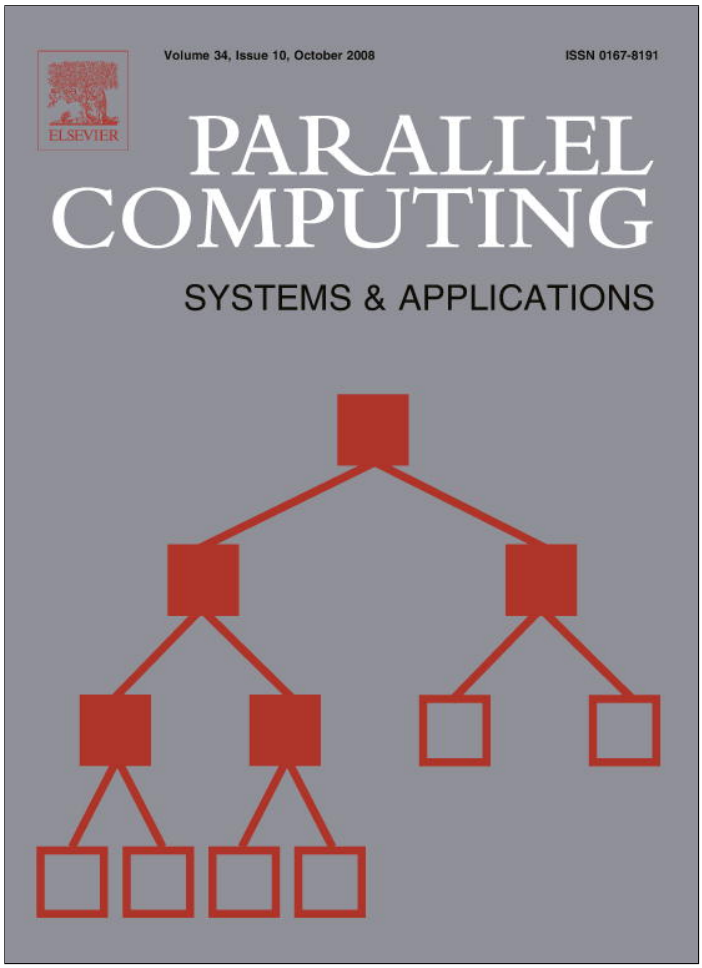

Parallel Computing 34 (2008) 551–569

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Parallel preconditioners for the conjugate gradient algorithm using Gram–Schmidt and least squares methods

Julien Straubhaar

Université de Neuchâtel, Institut de Mathématiques, Rue Emile-Argand 11, CH-2009 Neuchâtel, Switzerland

ARTICLE INFO

Article history:
Received 8 January 2007
Received in revised form 22 October 2007
Accepted 12 June 2008
Available online 19 June 2008

Keywords:
Preconditioned conjugate gradient method
Gram–Schmidt orthogonalization
Least squares
Parallelization
Performance
Speed-up

ABSTRACT

This paper is devoted to the study of some preconditioned conjugate gradient algorithms on parallel computers. The considered preconditioners (presented in [J. Straubhaar, Preconditioners for the conjugate gradient algorithm using Gram–Schmidt and least squares methods, Int. J. Comput. Math. 84 (1) (2007) 89–108]) are based on incomplete Gram–Schmidt orthogonalization and least squares methods. The construction of the preconditioner and the resolution are treated separately. Numerical tests are performed and speed-up curves are presented in order to evaluate the performance of the algorithms.

1. Introduction

This paper is a continuation of the article [8]. Let us first briefly recall the object of this previous paper. Consider the linear system

$$Ax = b, \tag{1}$$

where $A = (a_{ij})$ is a real symmetric positive definite (SPD), $n \times n$ matrix and b a vector in $\mathbb{R}^n$. This equation is replaced by the equivalent system

$$TAT^{t}\tilde{x} = Tb, \quad x = T^{t}\tilde{x}, \tag{2}$$

where T is an $n \times n$ regular matrix, called preconditioner of A. The system (2) is solved using the preconditioned conjugate gradient (PCG) algorithm:

E-mail address: julien.straubhaar@unine.ch

0167-8191/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2008.06.001

PCG Algorithm

Let $x_0 \in \mathbb{R}^n$
Compute $r_0 = b - Ax_0$, $s_0 = T^t \cdot Tr_0$ and set $d_0 = s_0$
If $\|r_0\|/\|b\| < tol$: quit
For $k = 0, 1, \ldots$, do:
  $\tilde{\alpha}_k = (r_k|s_k)/(d_k|Ad_k)$
  $x_{k+1} = x_k + \tilde{\alpha}_k d_k$
  $r_{k+1} = r_k - \tilde{\alpha}_k Ad_k$
  If $\|r_{k+1}\|/\|b\| < tol$: quit the loop
  $s_{k+1} = T^t \cdot Tr_{k+1}$
  $\tilde{\beta}_k = (r_{k+1}|s_{k+1})/(r_k|s_k)$
  $d_{k+1} = s_{k+1} + \tilde{\beta}_k d_k$
End for k

The object of this paper is the study of some preconditioners with parallel computing. The *speed-up* and the *efficiency* are used to evaluate the performances of parallelized algorithms. The speed-up is the ratio $S_p = T_1/T_p$, where T_p denotes the computation time with p processors. The ideal situation is $S_p = p$ (when the numbers of processors is multiplied by a factor p, the computation time is divided by p). The efficiency is the rate $E_p = S_p/p$, i.e. it is the efficiency in comparison with the ideal case. Notice that if the first test is performed with m processors (i.e. no test has done with less processors), the speed-up is calculated by $S_p(m) = mT_m/T_p$ and the efficiency by $E_p(m) = S_p(m)/p$.

The construction of the preconditioner and the resolution with the PCG Algorithm constitute two distinct parts. The performance evaluation for each one is presented in Sections 4 and 5 separately.

The preconditioner DIAG + LS CGS (OPT) (see Section 3 and [8]) and its block version are considered for these evaluations. Note that for the construction part, the preconditioners whose columns (or rows) are computed independently can be treated in a similar way, for example the preconditioner FSPAI (*factorized sparse approximate inverse*) presented in [8] and developed in [5]. For the resolution part, the performance of the PCG Algorithm is evaluated and many preconditioners can be considered, for example FSPAI or INC CGS (see Section 3 and [1]).

2. Computational resources

The numerical tests presented in this paper are performed on the CRAY XT3 parallel machines in CSCS (Swiss National Supercomputing Center). More informations can be found in http://www.cscs.ch.

The code of the programs are written in *fortran* and the MPI (Message Passing Interface, see [2,3]) and SHMEM libraries are used to manage the parallel environment. Some documentation can be found in http://www.cray.com.

3. Preconditioners using Gram–Schmidt and least squares method

Let us recall the construction of the preconditioners of kind LS CGS (for **L**east **S**quares and **C**onjugate **G**ram–**S**chmidt) presented in [8]. Using the Gram–Schmidt orthogonalization process relatively to the A-inner product $((x|y)_A = (x|Ay) = x^t \cdot A \cdot y)$, an upper triangular and unitary diagonal $n \times n$ matrix $\widetilde{Z}$ such that $\widetilde{D} = \widetilde{Z}^t \cdot A \cdot \widetilde{Z}$ is diagonal can be constructed. An approximation Z of $\widetilde{Z}$ and the corresponding diagonal matrix

$$D = \mathrm{diag}((z_1|Az_1), \ldots, (z_n|Az_n))$$

give a preconditioner $T = D^{-1/2} \cdot Z^t$ (verifying $TAT^t \approx I$). Some coefficients in Z are fixed to zero in order to get a sparse preconditioner. Several methodologies can be used to obtain an approximation Z. One possibility developed in [1] consists to apply the A-orthogonalization process, ignoring the coefficients in Z fixed to zero (this leads to the preconditioner INC CGS (*incomplete conjugate Gram–Schmidt*)).

The point of view developed in [8] is to construct the columns of Z independently with approximations in least squares sense. Let z_k the kth column of Z, $z_k(k) = Z_{kk} = 1$ and

$$\mathscr{J}_k = \{j_1 < \cdots < j_p\} \subset \{1, \ldots, k-1\},$$

the indices set of the components of z_k to determine. The relations

$$(z_k|Az_i) = 0, \quad i = 1, \ldots, k-1$$

can be written by the linear system

$$By = c, \tag{3}$$

where

$$y = (y_1, \ldots, y_p)^t = z_k(\mathscr{J}_k) = (Z_{j_1 k}, \ldots, Z_{j_p k})^t,$$

$B = (b_{il})$ is the $(k-1) \times p$ matrix defined by $b_{il} = (a_{j_l}|z_i)$ (with a_{j_l} the j_lth column of A) and $c = (c_1, \ldots, c_{k-1})^t$ is the vector in $\mathbb{R}^{k-1}$ defined by $c_i = -(a_k|z_i)$.

If $Z_{k-1} = (Z_{ij})_{1 \leqslant i,j \leqslant k-1}$ denotes the principal submatrix of Z of order $k-1$, $\widetilde{A}$ the $(k-1) \times p$ matrix obtained by keeping in A the $k-1$ first rows and the columns $j_1, \ldots, j_p$ (i.e. $\widetilde{A}_{il} = a_{ij_l}$) and $\tilde{a}_k$ the vector in $\mathbb{R}^{k-1}$ made up of the $k-1$ first components of a_k (the kth column of A), we have

$$B = Z_{k-1}^t \cdot \widetilde{A}$$

and

$$c = -Z_{k-1}^t \cdot \tilde{a}_k.$$

Since the matrix Z_{k-1} is regular (upper triangular and unitary diagonal), the linear system (3) is equivalent to

$$\widetilde{A}y = -\tilde{a}_k. \tag{4}$$

The system (4) is solved in the least squares sense, i.e. the considered solution is the vector $y \in \mathbb{R}^p$ which minimize

$$\|\widetilde{A}y + \tilde{a}_k\|. \tag{5}$$

This vector is given by the system (see for example [7, pp. 106–107])

$$\widetilde{A}^t \cdot \widetilde{A}y = -\widetilde{A}^t \cdot \tilde{a}_k. \tag{6}$$

Consider the QR decomposition $\widetilde{A} = QR$, where $Q = (q_1, \ldots, q_p)$ is an $(k-1) \times p$ matrix satisfying $Q^t \cdot Q = I_p$ and $R = (r_{ij})$ a regular square and upper triangular matrix of order p (see [6, p. 11]). The solution of (6) is obtained by solving the triangular system

$$Ry = -Q^t \tilde{a}_k. \tag{7}$$

According to (5), the vector $\tilde{z}_k = (Z_{1k}, \ldots, Z_{k-1,k})^t$, with $Z_{j_l k} = y_l$, $l = 1, \ldots, p$ and $Z_{ik} = 0$ if $i \notin \mathscr{J}_k$, realizes the minimum

$$m = \min_{\substack{u \in \mathbb{R}^{k-1} \\ u_i = 0, i \notin \mathscr{J}_k}} \|A_{k-1}u + \tilde{a}_k\|. \tag{8}$$

Note that the computation of the kth column z_k of the matrix Z is independent of the other columns.

The set of indices $\mathscr{J}_k$ can be chosen in several ways (see [8]). Recall here the "optimal filling" of the matrix Z (using some ideas in the article [4]). The set $\mathscr{J} = \mathscr{J}_k$ and the vector z_k are constructed so that the minimum m of (8) is lower or equal to a given number ε. For this, the maximal number $p_{\max}$ of indices in $\mathscr{J}$ and the "additional filling" $s (1 \leqslant s \leqslant p_{\max})$ are fixed; we then proceed in the following way:

(i) start with $\mathscr{J} = \phi$,
(ii) compute the vector $\tilde{z}_k$ realizing the minimum m of (8),
(iii) if $m \leqslant \varepsilon$ or $|\mathscr{J}| \geqslant p_{\max}$: quit the loop,
(iv) add s indices to $\mathscr{J}$ and go to (ii).

Once out of the loop, we set $z_k = (\tilde{z}_k^t, 1, 0, \ldots, 0)^t \in \mathbb{R}^n$. Notice that the condition $m \leqslant \varepsilon$ in (ii) avoids a strong filling in Z. It remains to choose the additional indices for a given set $\mathscr{J}$. Let

$$r = A_{k-1}\tilde{z}_k + \tilde{a}_k,$$

where $\tilde{z}_k$ realizes the minimum of (8), i.e. $\|r\| = \min\{\|A_{k-1}u + \tilde{a}_k\| | u \in \mathbb{R}^{k-1}, u_i = 0, i \notin \mathscr{J}\}$. Let $\mathscr{L} = \{1 \leqslant l \leqslant k-1 | r_l \neq 0\}$ and, for each $l \in \mathscr{L}$, set $\mathscr{M}_l = \{1 \leqslant j \leqslant k-1 | a_{lj} \neq 0, j \notin \mathscr{J}\}$. Then

$$\widetilde{\mathscr{J}} = \bigcup_{l \in \mathscr{L}} \mathscr{M}_l$$

is a nonempty set (see [8]) in which the s new indices are selected in order to reduce $\|r\|$. For each $j \in \widetilde{\mathscr{J}}$, we have

$$\min_{\mu \in \mathbb{R}} \|r + \mu A_{k-1}e_j\|^2 = \|r\|^2 - \frac{(r|A_{k-1}e_j)^2}{\|A_{k-1}e_j\|^2},$$

we consider the weight

$$\omega_j = \frac{(r|A_{k-1}e_j)^2}{\|A_{k-1}e_j\|^2}$$

and we choose s indices in $\widetilde{\mathscr{J}}$ among those of largest weight and we add them to $\mathscr{J}$ (if $|\widetilde{\mathscr{J}}| \leqslant s$, we select $\widetilde{\mathscr{J}}$ entirely).

To compute the vector $\tilde{z}_k$ realizing the minimum

$$\min_{\substack{u\in\mathbb{R}^{k-1}\\ u_i=0, i\notin \mathscr{J}\sqcup\widetilde{\mathscr{J}}}} \|A_{k-1}u+\tilde{a}_k\|,$$

the QR decomposition $\widehat{A}=QR$ of the matrix

$$\widehat{A}=(A_{k-1}e_{j_1},\ldots,A_{k-1}e_{j_p},A_{k-1}e_{\tilde{j}_1},\ldots,A_{k-1}e_{\tilde{j}_s})$$

is computed (obtained by extending the QR decomposition)

$$\widetilde{A}=(A_{k-1}e_{j_1},\ldots,A_{k-1}e_{j_p})=\widetilde{Q}\widetilde{R}$$

known from the previous step and the vector $y=(Z_{j_1k},\ldots,Z_{j_pk},Z_{\tilde{j}_1k},\ldots,Z_{\tilde{j}_sk})^{\mathrm{t}}$ is obtained using (7). Note that $j_1,\ldots,j_p,\tilde{j}_1,\ldots,\tilde{j}_s$ is not necessary in an increasing order.

Moreover, the system (1) can first be preconditioned with the diagonal matrix $T_1=\mathrm{diag}(a_{11}^{-1/2},\ldots,a_{nn}^{-1/2})$. Then, the preconditioner T_2 presented above is computed for the SPD matrix $T_1\cdot A\cdot T_1^{\mathrm{t}}$. This leads to the coupled preconditioner $T_2\cdot T_1$, called DIAG + LS CGS (OPT).

The block version of this preconditioner consists to decompose the matrix A into diagonal blocks (block Jacobi decomposition) and compute the previous preconditioner for each one (see [8]).

Theoretical results and numerical tests are given for this preconditioner in [8]. A parallel computing approach is developed in the following sections.

4. Construction of the preconditioner

Assume that the program use p processors. They are numbered from 0 to $p-1$. The $n\times n$ matrix Z in the preconditioner LS CGS (OPT) is distributed in the following way. The jth column of Z is computed by the processor $j-1$ modulo p. Since Z is upper triangular, this equilibrates the filling and the computation charges on each processor. A continuous splitting of Z (i.e. the first $\lfloor n/p\rfloor(+1)$ on the processor 0, the $\lfloor n/p\rfloor(+1)$ following ones on the processor 1, and so on) would be a bad choice, because the processors with a small number would finish to compute their part much before the processors with a high number. For the construction of the kth column of Z, the principal submatrix A_{k-1} of order $k-1$ and the first $k-1$ components of the kth column of A are used (for details, see previous section or [8]). Hence, the whole matrix A is stored on every processors to avoid too many communications between them. The columns of the preconditioner are computed independently; the copying of the matrix A into all processors and an error test at the end of computation constitute all the necessary communications (realized with the MPI library).

Remark 4.1. The CSR (or CSC) storage format (see Section 5.1) is used for the matrix A and the local part of Z is stored in CSC format.

For the block version of this preconditioner, the same method is applied to each block successively: the corresponding block in A is copying on every processors and an error test completes the computation of the considered block. If M blocks are considered and $n=q\cdot M+r$ is the Euclidean division of n by M (q, r are integers with $0\leqslant r<M$), the r first blocks are of order $q+1$ and the other ones of order q. The block treatment of this preconditioner allows to consider large matrices.

Remark 4.2. The algorithm of the construction of the preconditioner is parallelizable intrinsically. So, only one matrix is considered for the performance evaluation in the next section.

4.1. Performance evaluation

The SPD test matrix *gyro_k* is considered. It's a matrix of order $n=17,361$ with $nnz=519,620$ nonzero coefficients in its upper (or lower) part. This matrix is obtained from http://www.cise.ufl.edu/research/sparse/matrices.

This "small" problem gives a good idea of the computation performance according to the parameter $p_{\max}$ and the number of blocks for the preconditioner DIAG + LS CGS (OPT). The parameters $p_{\max}=10,20,50,100$, $\varepsilon=9.09\mathrm{E}-13$ and $s=1$ (see Section 3) are considered. The filling obtained for Z is given in Table 1 for standard and block versions. The quantity nnz is the number of nonzero coefficients in Z.

The computation time (T_p), the speed-up (S_p) and the efficiency (E_p) for the construction of these preconditioners are presented in Tables 2–5. The number p is the number of used processors. The speed-up and efficiency curves for the computation of the preconditioner Z for different values of the number of blocks are presented in Figs. 1–6. The number p of used processors is represented on X-coordinate and the speed-up S_p or the efficiency E_p on the Y-coordinate.

Note that in Fig. 2 the curve for $p_{\max}=100$ represents the values of $E_p(2)$.

In the standard version (Figs. 1 and 2), one block of order 17,361 is considered. In the 16 blocks version (Figs. 3 and 4), the first block is of order 1086 and the 15 following ones are of order 1085. In the 32 blocks version (Figs. 5 and 6), the first 17 blocks are of order 543 and the 15 others ones are of order 542.

Table 1
Filling of preconditioner matrix

p_{max}	*nnz*		
	1 block	16 blocks	32 blocks
10	189,702	179,641	170,015
20	360,174	327,797	297,542
50	863,688	729,669	607,914
100	1,683,529	1,309,240	993,436

Table 2
Performance evaluation for construction of preconditioner with $p_{max} = 10$

p	1 block			16 blocks			32 blocks		
	T_p	S_p	E_p	T_p	S_p	E_p	T_p	S_p	E_p
1	1.31E+03	1.00E+00	1.00E+00	2.44E+02	1.00E+00	1.00E+00	8.92E+01	1.00E+00	1.00E+00
2	6.54E+02	2.00E+00	1.00E+00	1.23E+02	1.99E+00	9.94E−01	4.52E+01	1.97E+00	9.86E−01
4	3.28E+02	3.99E+00	9.96E−01	6.24E+01	3.92E+00	9.79E−01	2.31E+01	3.86E+00	9.65E−01
8	1.64E+02	7.97E+00	9.96E−01	3.23E+01	7.55E+00	9.44E−01	1.23E+01	7.24E+00	9.04E−01
16	8.32E+01	1.57E+01	9.83E−01	1.73E+01	1.41E+01	8.84E−01	6.83E+00	1.31E+01	8.17E−01
32	4.26E+01	3.07E+01	9.59E−01	9.62E+00	2.54E+01	7.93E−01	4.14E+00	2.16E+01	6.74E−01
64	2.16E+01	6.05E+01	9.45E−01	5.53E+00	4.42E+01	6.90E−01	2.70E+00	3.31E+01	5.17E−01
128	1.15E+01	1.13E+02	8.85E−01	3.71E+00	6.59E+01	5.15E−01	1.99E+00	4.48E+01	3.50E−01

Table 3
Performance evaluation for construction of preconditioner with $p_{max} = 20$

p	1 block			16 blocks			32 blocks		
	T_p	S_p	E_p	T_p	S_p	E_p	T_p	S_p	E_p
1	3.46E+03	1.00E+00	1.00E+00	6.72E+02	1.00E+00	1.00E+00	2.26E+02	1.00E+00	1.00E+00
2	1.74E+03	1.99E+00	9.97E−01	3.38E+02	1.99E+00	9.94E−01	1.15E+02	1.98E+00	9.88E−01
4	8.69E+02	3.98E+00	9.96E−01	1.71E+02	3.93E+00	9.83E−01	5.86E+01	3.87E+00	9.66E−01
8	4.36E+02	7.94E+00	9.93E−01	8.95E+01	7.51E+00	9.39E−01	3.11E+01	7.28E+00	9.10E−01
16	2.21E+02	1.57E+01	9.80E−01	4.72E+01	1.43E+01	8.91E−01	1.70E+01	1.33E+01	8.32E−01
32	1.14E+02	3.03E+01	9.47E−01	2.62E+01	2.57E+01	8.02E−01	9.97E+00	2.27E+01	7.09E−01
64	5.92E+01	5.84E+01	9.13E−01	1.49E+01	4.52E+01	7.06E−01	6.27E+00	3.61E+01	5.64E−01
128	3.04E+01	1.14E+02	8.90E−01	9.53E+00	7.05E+01	5.51E−01	4.35E+00	5.20E+01	4.06E−01

Table 4
Performance evaluation for construction of preconditioner with $p_{max} = 50$

p	1 block			16 blocks			32 blocks		
	T_p	S_p	E_p	T_p	S_p	E_p	T_p	S_p	E_p
1	1.41E+04	1.00E+00	1.00E+00	2.67E+03	1.00E+00	1.00E+00	8.08E+02	1.00E+00	1.00E+00
2	7.07E+03	1.99E+00	9.96E−01	1.34E+03	1.99E+00	9.95E−01	4.08E+02	1.98E+00	9.89E−01
4	3.55E+03	3.97E+00	9.93E−01	6.77E+02	3.95E+00	9.87E−01	2.08E+02	3.88E+00	9.71E−01
8	1.78E+03	7.92E+00	9.90E−01	3.55E+02	7.52E+00	9.41E−01	1.10E+02	7.33E+00	9.17E−01
16	9.00E+02	1.56E+01	9.78E−01	1.85E+02	1.45E+01	9.05E−01	5.96E+01	1.36E+01	8.47E−01
32	4.60E+02	3.06E+01	9.56E−01	1.02E+02	2.62E+01	8.20E−01	3.43E+01	2.36E+01	7.36E−01
64	2.36E+02	5.96E+01	9.32E−01	5.73E+01	4.67E+01	7.29E−01	2.10E+01	3.84E+01	6.00E−01
128	1.20E+02	1.17E+02	9.13E−01	3.56E+01	7.51E+01	5.86E−01	1.37E+01	5.90E+01	4.61E−01

Table 5
Performance evaluation for construction of preconditioner with $p_{max} = 100$

p	1 block			16 blocks			32 blocks		
	T_p	S_p	E_p	T_p	S_p	E_p	T_p	S_p	E_p
1				7.78E+03	1.00E+00	1.00E+00	2.13E+03	1.00E+00	1.00E+00
2	2.22E+04	2.00E+00	1.00E+00	3.91E+03	1.99E+00	9.94E−01	1.07E+03	1.98E+00	9.92E−01
4	1.11E+04	4.00E+00	1.00E+00	1.97E+03	3.95E+00	9.88E−01	5.48E+02	3.89E+00	9.73E−01
8	5.56E+03	7.98E+00	9.97E−01	1.02E+03	7.59E+00	9.49E−01	2.90E+02	7.34E+00	9.18E−01
16	2.81E+03	1.58E+01	9.87E−01	5.32E+02	1.46E+01	9.14E−01	1.55E+02	1.37E+01	8.57E−01
32	1.43E+03	3.10E+01	9.68E−01	2.89E+02	2.69E+01	8.40E−01	8.85E+01	2.41E+01	7.53E−01
64	7.40E+02	6.00E+01	9.37E−01	1.63E+02	4.77E+01	7.46E−01	5.34E+01	3.99E+01	6.24E−01
128	3.78E+02	1.17E+02	9.16E−01	1.00E+02	7.77E+01	6.07E−01	3.48E+01	6.13E+01	4.79E−01

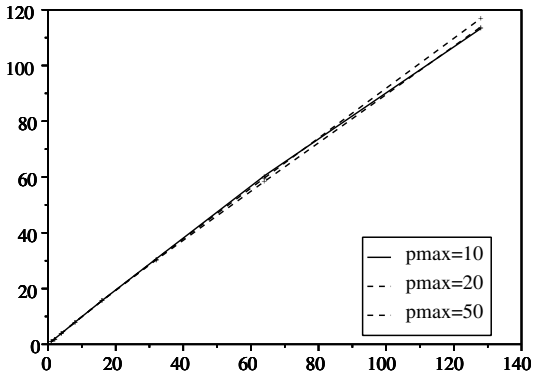

Fig. 1. Speed-up in function of #proc. Speed-up for the construction, 1 block.

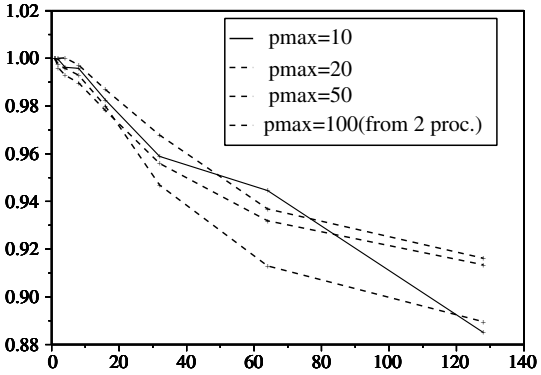

Fig. 2. Efficiency in function of #proc. Efficiency for the construction, 1 block.

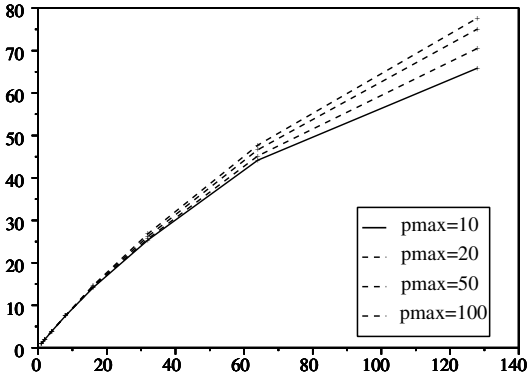

Fig. 3. Speed-up in function of #proc. Speed-up for the construction, 16 blocks.

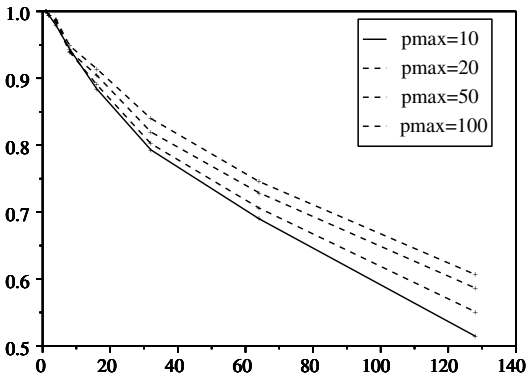

Fig. 4. Efficiency in function of #proc. Efficiency for the construction, 16 blocks.

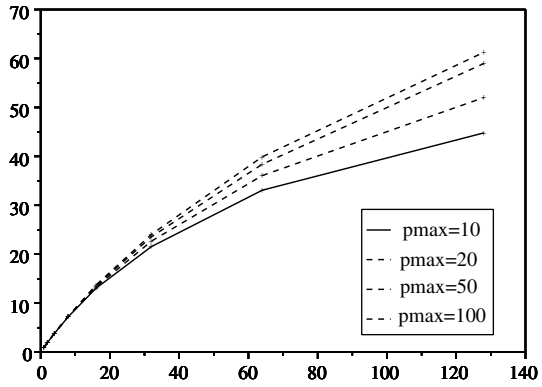

Fig. 5. Speed-up in function of #proc. Speed-up for the construction, 32 blocks.

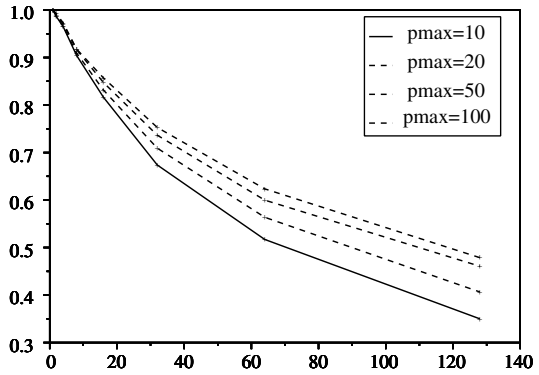

Fig. 6. Efficiency in function of #proc. Efficiency for the construction, 32 blocks.

For each processor the number of computed columns in one block varies from one at the maximum. Consider for example the 32 blocks version with 128 processors. Since $542 = 128 \cdot 4 + 30$, for the blocks of order 542, each processor from 0 to 29 computes 5 columns, and each other processor (from 30 to 127) computes 4 columns. For the blocks of order 543, each processor from 0 to 30 computes 5 columns, and each other processor (from 31 to 127) computes 4 columns. Then, for the whole preconditioner, each processor from 0 to 29 computes $32 \cdot 5 = 160$ columns, the processor 30 computes $17 \cdot 5 + 15 \cdot 4 = 145$ columns and each processor from 31 to 127 compute $32 \cdot 4 = 128$ columns. This can explain why the block versions are less efficient that the one with only one block.

4.2. Observations and conclusion

In general, when the amount of computations increases, the quality of performance is also increasing: the performance is of good quality for large matrices. Compare Figs. 1–6 and the filling Table 1.

For the computation of each block, two communications (copying of part of A and error test) require synchronization of the processors. Hence, the quality of the performance decreases when the number of blocks increases (see Figs. 1–6). However, for the construction of the preconditioner, the block decomposition is a good methodology to save computer time (see Tables 2–5).

5. Resolution with the PCG Algorithm

In this section, the coupled preconditioner DIAG + LS CGS (OPT), written in the form $T = D^{-1/2}Z^{t}$, is supposed to be constructed and the resolution with the parallelized PCG Algorithm is studied. For this let us first introduce the storage methodology, some communication routines and some simple computations.

5.1. Storage

Let B be a sparse $m \times n$ matrix and nnz the number of nonzero coefficients in B. This matrix can be described by the three vectors

(1) *ab*: real vector of length *nnz* containing the nonzero coefficients of B, row by row;
(2) *jab*: integer vector of length *nnz* containing the column indices of the corresponding coefficients in *ab*;

(3) *iab*: integer vector of length $m+1$ containing the pointers of the beginning of each row in *ab* and *jab*, i.e. the values in *ab* and *jab* from position $iab(k)$ to $iab(k+1)-1$ concern the kth row. We have $iab(1)=1$ and $iab(m+1)=nnz+1$.

This storage format is called CSR (compressed sparse row) storage format.

Exchanging the roles of rows and columns and replacing m by n in the CSR storage format, we obtained the CSC (compressed sparse column) storage format.

For more details about storage formats, see [6] and the *sparskit* software developed by Saad from http://www-users.cs.umn.edu/~saad/.

Assume now that p processors numbered from 0 to $p-1$ are used and let M be a sparse $n \times n$ matrix. In this case, a continuous splitting by rows of the matrix M can be considered: the q_0 first rows are stored on the processor 0, the q_1 following ones on the processor 1, ..., and the q_{p-1} last ones on the processor $p-1$, where $q_0 = \cdots = q_{r-1} = q+1$ and $q_r = \cdots = q_{p-1} = q$ and $n = q \cdot p + r$ is the Euclidean division of n by p. The local part of the matrix M stored on the processor i is denoted by $M_{\text{loc}}^{(i)}$, its dimension is $q_i \times n$ and the CSR storage format is used.

In a similar way, a continuous splitting by columns of the matrix M can be considered. Then the local part of the matrix M on the processor i is of dimension $n \times q_i$ and the CSC storage format is used.

For a vector u in $\mathbb{R}^n$, a continuous splitting by components can be considered and the local vector $u_{\text{loc}}^{(i)} \in \mathbb{R}^{q_i}$ will be stored on the processor i. This is also used for a diagonal matrix D since it is characterized by a vector; then $D_{\text{loc}}^{(i)}$ denotes the local diagonal $q_i \times q_i$ matrix on the processor i.

Remark 5.1. Using the continuous splitting (by rows, columns or components) above, the notation $M_{\text{loc}}^{(i)}$ can be replaced by M_{loc}, $u_{\text{loc}}^{(i)}$ by u_{loc}, $D_{\text{loc}}^{(i)}$ by D_{loc} and q_i by n_{loc}; then each processor has its own local matrices M_{loc}, D_{loc}, its own local vector u_{loc} and its own dimension n_{loc}.

5.2. Some communication routines

Let us here briefly present the concept of three communication routines provided by the MPI library which will be used in the parallelized PCG Algorithm. For details, see [2,3].

The two first routines involve global communications, that is, **every** processors call the routine and receive its result on return.

- MPI_ALLREDUCE routine. If, for each i from 0 to $p-1$, v_i is a vector in $\mathbb{R}^l$ stored on processor i, this routine allows to get the sum $v = v_0 + \cdots + v_{p-1}$ on **every** processors.
- MPI_ALLGATHERV routine. If, for each i from 0 to $p-1$, v_i is a vector in $\mathbb{R}^{l_i}$ stored on processor i, this routine allows to get the vector v of length $l = l_0 + \cdots + l_{p-1}$ obtained by gathering $v_0, \ldots, v_{p-1}$ end to end (i.e. $v = (v_0, \ldots, v_{p-1})$) on **every** processors.

The following routine is a one-sided communication routine, **only one** processor call it.

- MPI_GET routine. This routine allows the processor that call it to get a copy of a variable (or a vector) stored on another processor.

Remark 5.2. The SHMEM library can also be used for communications. For example, the SHMEM_GET routine corresponds to the MPI_GET routine for this library.

5.3. Simple computations

In this section, the computation of the product $u = M \cdot v$ where M is an $n \times n$ matrix and v a vector in $\mathbb{R}^n$ is explained in some situations.

Using a single processor, assume that M is stored in CSR or CSC format with three vectors *am*, *jam* and *iam* (see above).

Algorithm 1.

Comp. of $u = Mv$, M in CSR format

```
For i from 1 to n, do:
    w = 0
    For j from iam(i) to iam(i+1) − 1, do:
        w = w + am(j) · v(jam(j))
    End for j
    u(i) = w
End for i
```

Algorithm 2.

Comp. of $u = Mv$, M in CSC format

$u(i) = 0$, $i = 1, \ldots, n$
For i from 1 to n, do:
  $w = v(i)$
  For j from $iam(i)$ to $iam(i+1) - 1$, do:
    $u(jam(j)) = u(jam(j)) + am(j) \cdot w$
  End for j
End for i

Assume now that p processors are used and that the matrix M and the vector v are stored respecting the splitting of Section 5.1. Let us present the product $u = M \cdot v$ in three cases (the notations of Section 5.1 are used).

Algorithm 3. Splitting by rows, M_{loc} in CSR format, gathering

(1) Use MPI_ALLGATHERV to get the vector v on every processors from v_{loc}.
(2) Compute $u_{\text{loc}} = M_{\text{loc}} \cdot v$ on each processor using Algorithm 1.

Algorithm 4. Splitting by rows, M_{loc} in CSR format, one-sided communications

(1) Using MPI_GET, each processor get only the necessary components in the vectors v_{loc} stored on the others processors in order to perform the local product below.
(2) Compute $u_{\text{loc}} = M_{\text{loc}} \cdot v$ on each processor using Algorithm 1.

Algorithm 5. Splitting by columns, M_{loc} in CSC format

(1) Compute $w = M_{\text{loc}} \cdot v_{\text{loc}} \in \mathbb{R}^n$ on each processor using Algorithm 2.
(2) Use MPI_ALLREDUCE to get the vector u on every processors adding the p local vectors w in each processor.

Note that at the end of Algorithms 3 and 4, each processor has only the local part u_{loc} of the vector u in its memory, whereas at the end of Algorithm 5, the whole vector u is stored on every processors.

Remark 5.3. If Algorithms 1, 3 or 4 (resp. 2 or 5) is used for a matrix M stored in CSC (resp. CSR) format, the vector $u = M^t \cdot v$ is obtained.

5.4. PCG Algorithm step by step

Since the system matrix A is symmetric, the CSR and CSC formats are exactly the same for this matrix. For the matrix Z in the preconditioner $T = D^{-1/2}Z^{\text{t}}$, the CSC storage format is used, according to its construction by columns. Assume that the continuous splitting of Section 5.1 is used to store the matrix A (splitting by rows, CSR format), the matrix Z (splitting by columns, CSC format) and the diagonal matrix D^{-1} (splitting by components on the diagonal).

Then recall in details the steps in one iteration of the PCG Algorithm from the if-test in the loop:

$$\tilde{s} = Z^{\text{t}} r, \tag{9a}$$
$$\hat{s} = D^{-1}\tilde{s}, \tag{9b}$$
$$s = Z\hat{s}, \tag{9c}$$
$$t = (r|s), \tag{9d}$$
$$\tilde{\beta} = t/t_{\text{old}}, \tag{9e}$$
$$t_{\text{old}} = t, \tag{9f}$$
$$d = s + \tilde{\beta}d, \tag{9g}$$
$$q = Ad, \tag{9h}$$
$$\tilde{\alpha} = t/(d|q), \tag{9i}$$
$$x = x + \tilde{\alpha}d, \tag{9j}$$

$$r = r - \tilde{\alpha} q. \tag{9k}$$

$$\text{If } \|r\|^2/\|b\|^2 < tol^2 : \text{ go out of the loop.} \tag{9l}$$

Assume that the vector r is splitted over all processors at the start. The step (9a) is performed using Algorithm 3 or Algorithm 4. Then, each processor computes

$$\hat{s}_{\text{loc}} = D_{\text{loc}}^{-1} \tilde{s}_{\text{loc}}$$

and the step (9c) is completed using Algorithm 5. The whole vector s is then stored on every processors.

For the step (9d), the local inner product $(r_{\text{loc}}|s_{\text{loc}})$ is computed locally on each processor and the MPI_ALLREDUCE routine is used to get the sum. The steps (9e)–(9g) are executed by each processor globally. The step (9h) is local, i.e. each processor computes

$$q_{\text{loc}} = A_{\text{loc}} d.$$

The inner product in the step (9i) is obtained like in the step (9d). Then the steps (9j) and (9k) are local, the following computations are performed on each processor:

$$x_{\text{loc}} = x_{\text{loc}} + \tilde{\alpha} d_{\text{loc}},$$
$$r_{\text{loc}} = r_{\text{loc}} - \tilde{\alpha} q_{\text{loc}}.$$

For the test (9l), the inner product $(r|r)$ is computed like in the steps (9d) and (9i).

Remark 5.4. Consider the block version of the preconditioner DIAG + LS CGS (OPT) with M blocks (see Section 3). Assume that $p = M$ processors are used for the PCG Algorithm. In this case, there is one block on each processor and the three first steps ((9a)–(9c)) can be locally completed without communication. If here Z_{loc} denotes the $nloc \times nloc$ diagonal block (own to each processor), using Algorithms 1 and 2 for the first and third steps, respectively we get

$$\tilde{s}_{\text{loc}} = Z_{\text{loc}}^{\text{t}} r_{\text{loc}},$$
$$\hat{s} = D_{\text{loc}}^{-1} \tilde{s},$$
$$s_{\text{loc}} = Z_{\text{loc}} \hat{s}_{\text{loc}}.$$

Then the vector s is not entirely stored on the processors. Hence, the step (9g) is computed locally

$$d_{\text{loc}} = s_{\text{loc}} + \tilde{\beta} d_{\text{loc}}$$

and the step (9h) is completed using Algorithm 3 (or 4).

5.5. Performance evaluation

Two SPD test matrices are considered in order to study the performance of the resolution with the PCG Algorithm.

The convergence tolerance tol is set to 10^{-8}. For each test, the number of necessary iterations is given. Since this number varies slightly, the speed-up is calculated by $S_p = (T_1/It_1)/(T_p/It_p)$, where It_p denotes the number of iterations with p processors. Thus, the comparisons of time are made per iteration. The efficiency will not be given here in order to reduce the tables. The resolution algorithm is called *PCG Algorithm with gathering* if Algorithm 3 is used for the first step (9a) (see previous section) and *PCG Algorithm with one-sided communications* if this step is completed using Algorithm 4.

Remark 5.5. The one-sided communications are performed using the SHMEM_GET routine (see Remark 5.2). Indeed, in experiments, the SHMEM library is more efficient than the MPI library for this kind of communications.

5.5.1. First matrix

The SPD test matrix and the preconditioners of Section 4.1 are considered. The numerical results are presented in Tables 6–9.

Like in Section 4.1, the number p of used processors is represented on X-coordinate and the speed-up S_p on the Y-coordinate in Figs. 7–12.

Comparisons between PCG Algorithm with gathering and PCG Algorithm with one-sided communications in the case of the preconditioner by blocks and $p_{\max} = 100$ are presented in Figs. 13 and 14.

The use of one-sided communications is clearly better than the gathering methodology when the numbers of blocks and the number of processors are the same. In this situation, no communication is necessary for the step (9a) with Algorithm 4. Considering Remark 5.4, the call to the MPI_ALLREDUCE routine in the step (9c) can be replaced by a call to the MPI_ALLGATHERV routine in the step (9h) and so computer time can be save as shown by the Tables 10 and 11. (The number of iterations does not vary when the methodology changes.)

Table 6
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 10$

p	1 block			16 blocks			32 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	4328	3.06E+01	1.00E+00	5561	3.88E+01	1.00E+00	6315	4.37E+01	1.00E+00
2	4468	1.85E+01	1.70E+00	5596	2.28E+01	1.71E+00	6316	2.54E+01	1.72E+00
4	4400	1.23E+01	2.54E+00	5528	1.44E+01	2.67E+00	6328	1.75E+01	2.51E+00
8	4294	9.24E+00	3.29E+00	5678	1.27E+01	3.13E+00	6331	1.42E+01	3.08E+00
16	4470	8.91E+00	3.55E+00	5691	1.13E+01	3.50E+00	6328	1.28E+01	3.43E+00
32	4453	8.38E+00	3.76E+00	5737	1.07E+01	3.72E+00	6415	1.20E+01	3.71E+00
64	4288	7.55E+00	4.02E+00	5680	1.07E+01	3.71E+00	6276	1.17E+01	3.72E+00
PCG Algorithm with one-sided communications									
1	4328	3.06E+01	1.00E+00	5561	3.90E+01	1.00E+00	6315	4.36E+01	1.00E+00
2	4468	2.08E+01	1.52E+00	5596	2.08E+01	1.88E+00	6316	2.61E+01	1.67E+00
4	4400	2.65E+01	1.18E+00	5528	1.39E+01	2.79E+00	6328	1.78E+01	2.46E+00
8	4294	5.00E+01	6.08E−01	5678	9.88E+00	4.03E+00	6331	1.38E+01	3.17E+00
16	4470	4.88E+01	6.48E−01	5691	9.00E+00	4.44E+00	6328	1.20E+01	3.64E+00
32	4453	5.69E+01	5.54E−01	5737	3.10E+01	1.30E+00	6415	9.21E+00	4.81E+00
64	4288	4.76E+01	6.37E−01	5680	3.61E+01	1.10E+00	6276	2.43E+01	1.78E+00

Table 7
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 20$

p	1 block			16 blocks			32 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	3546	3.02E+01	1.00E+00	5208	4.29E+01	1.00E+00	6008	4.78E+01	1.00E+00
2	3665	1.81E+01	1.72E+00	5217	2.41E+01	1.78E+00	5974	2.74E+01	1.73E+00
4	3660	1.16E+01	2.70E+00	5152	1.55E+01	2.74E+00	6201	1.82E+01	2.71E+00
8	3671	8.46E+00	3.70E+00	5328	1.25E+01	3.52E+00	6237	1.39E+01	3.58E+00
16	3689	7.80E+00	4.03E+00	5162	1.05E+01	4.05E+00	5911	1.18E+01	3.98E+00
32	3688	7.08E+00	4.44E+00	5017	9.59E+00	4.31E+00	5909	1.18E+01	4.00E+00
64	3649	6.66E+00	4.67E+00	5326	1.02E+01	4.32E+00	6163	1.10E+01	4.48E+00
PCG Algorithm with one-sided communications									
1	3546	3.04E+01	1.00E+00	5208	4.31E+01	1.00E+00	6008	4.82E+01	1.00E+00
2	3665	2.17E+01	1.45E+00	5217	2.30E+01	1.88E+00	5974	2.93E+01	1.64E+00
4	3660	3.02E+01	1.04E+00	5152	1.42E+01	3.01E+00	6201	1.98E+01	2.51E+00
8	3671	5.51E+01	5.72E−01	5328	1.06E+01	4.16E+00	6237	1.58E+01	3.18E+00
16	3689	5.23E+01	6.05E−01	5162	8.57E+00	4.99E+00	5911	1.27E+01	3.72E+00
32	3688	6.50E+01	4.87E−01	5017	3.03E+01	1.37E+00	5909	8.66E+00	5.47E+00
64	3649	5.78E+01	5.42E−01	5326	3.94E+01	1.12E+00	6163	2.35E+01	2.10E+00

Table 8
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 50$

p	1 block			16 blocks			32 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	2535	3.28E+01	1.00E+00	4815	5.66E+01	1.00E+00	5838	6.27E+01	1.00E+00
2	2532	1.83E+01	1.80E+00	4816	3.17E+01	1.79E+00	5881	3.54E+01	1.78E+00
4	2532	1.09E+01	3.01E+00	4836	1.97E+01	2.88E+00	5800	2.27E+01	2.74E+00
8	2529	7.91E+00	4.14E+00	4823	1.34E+01	4.24E+00	5835	1.57E+01	4.00E+00
16	2448	5.65E+00	5.61E+00	4738	1.07E+01	5.21E+00	5669	1.28E+01	4.74E+00
32	2549	5.26E+00	6.28E+00	4754	9.62E+00	5.81E+00	5799	1.17E+01	5.31E+00
64	2522	4.74E+00	6.89E+00	4739	8.93E+00	6.24E+00	5895	1.09E+01	5.83E+00
PCG Algorithm with one-sided communications									
1	2535	3.31E+01	1.00E+00	4815	5.69E+01	1.00E+00	5838	6.28E+01	1.00E+00
2	2532	2.38E+01	1.39E+00	4816	3.07E+01	1.85E+00	5881	3.90E+01	1.62E+00
4	2532	3.41E+01	9.69E−01	4836	1.85E+01	3.09E+00	5800	2.62E+01	2.38E+00
8	2529	5.14E+01	6.42E−01	4823	1.26E+01	4.54E+00	5835	1.82E+01	3.45E+00
16	2448	5.31E+01	6.02E−01	4738	8.70E+00	6.43E+00	5669	1.50E+01	4.06E+00
32	2549	6.21E+01	5.35E−01	4754	3.26E+01	1.72E+00	5799	9.21E+00	6.78E+00
64	2522	6.08E+01	5.42E−01	4739	4.21E+01	1.33E+00	5895	2.43E+01	2.61E+00

Table 9
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 100$

p	1 block			16 blocks			32 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	1770	3.64E+01	1.00E+00	4464	7.57E+01	1.00E+00	5487	7.77E+01	1.00E+00
2	1771	1.95E+01	1.87E+00	4457	4.12E+01	1.83E+00	5700	4.44E+01	1.82E+00
4	1818	1.17E+01	3.20E+00	4487	2.48E+01	3.06E+00	5511	2.74E+01	2.85E+00
8	1769	7.02E+00	5.19E+00	4523	1.63E+01	4.70E+00	5702	1.94E+01	4.17E+00
16	1818	5.37E+00	6.97E+00	4503	1.24E+01	6.15E+00	5690	1.42E+01	5.67E+00
32	1818	4.54E+00	8.24E+00	4447	9.97E+00	7.56E+00	5699	1.26E+01	6.41E+00
64	1827	3.91E+00	9.62E+00	4355	9.11E+00	8.11E+00	5737	1.14E+01	7.12E+00
PCG Algorithm with one-sided communications									
1	1770	3.65E+01	1.00E+00	4464	7.62E+01	1.00E+00	5487	7.81E+01	1.00E+00
2	1771	2.60E+01	1.41E+00	4457	4.04E+01	1.88E+00	5700	5.03E+01	1.62E+00
4	1818	3.63E+01	1.03E+00	4487	2.39E+01	3.21E+00	5511	3.19E+01	2.46E+00
8	1769	4.57E+01	7.98E−01	4523	1.51E+01	5.12E+00	5702	2.33E+01	3.49E+00
16	1818	5.48E+01	6.84E−01	4503	1.03E+01	7.46E+00	5690	1.89E+01	4.28E+00
32	1818	6.01E+01	6.24E−01	4447	3.02E+01	2.51E+00	5699	9.94E+00	8.17E+00
64	1827	5.72E+01	6.59E−01	4355	4.37E+01	1.70E+00	5737	2.46E+01	3.32E+00

Table 10
Computation time (T_{16}) for 16 blocks and processors

Methodology	$p_{max} = 10$	$p_{max} = 20$	$p_{max} = 50$	$p_{max} = 100$
Gathering	1.13E+01	1.05E+01	1.07E+01	1.24E+01
One-sided communications	9.00E+00	8.57E+00	8.70E+00	1.03E+01
According Remark 5.4	5.82E+00	5.59E+00	5.95E+00	7.85E+00

Table 11
Computation time (T_{32}) for 32 blocks and processors

Methodology	$p_{max} = 10$	$p_{max} = 20$	$p_{max} = 50$	$p_{max} = 100$
Gathering	1.20E+01	1.18E+01	1.17E+01	1.26E+01
One-sided communications	9.21E+00	8.66E+00	9.21E+00	9.94E+00
According Remark 5.4	4.75E+00	4.51E+00	5.03E+00	6.11E+00

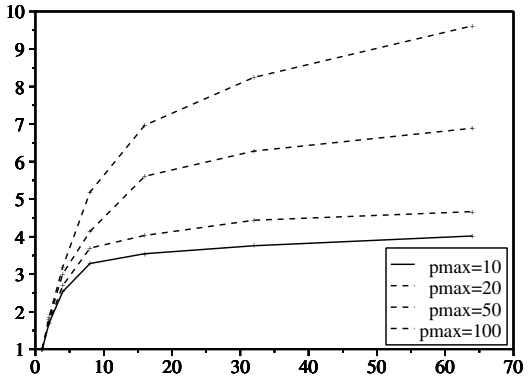

Fig. 7. Speed-up in function of #proc. Speed-up for PCG, 1 block, gathering.

5.5.2. Second matrix

Consider now the SPD test matrix *af_0_k101* of order $n = 503,625$ with $nnz = 9,027,150$ nonzero coefficients in its upper (or lower) part; this matrix is also obtained from http://www.cise.ufl.edu/research/sparse/matrices.

Consider the preconditioner DIAG + LS CGS (OPT) with the parameters $p_{max} = 10, 20, 50, 100$, $\varepsilon = 1.00\text{E} - 10$ and $s = 1$ (see Section 3). The filling obtained for Z is given in Table 12 where the quantity nnz denotes the number of nonzero coefficients in Z.

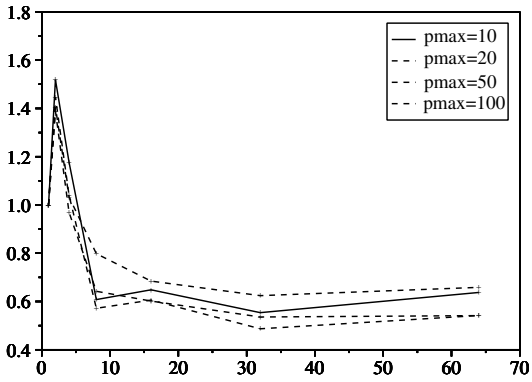

Fig. 8. Speed-up in function of #proc. Speed-up for PCG, 1 block, one-sided.

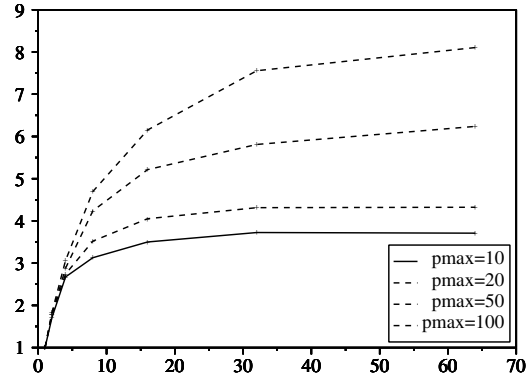

Fig. 9. Speed-up in function of #proc. Speed-up for PCG, 16 blocks, gathering.

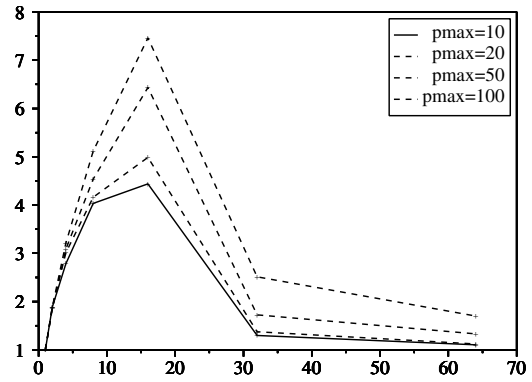

Fig. 10. Speed-up in function of #proc. Speed-up for PCG, 16 blocks, one-sided.

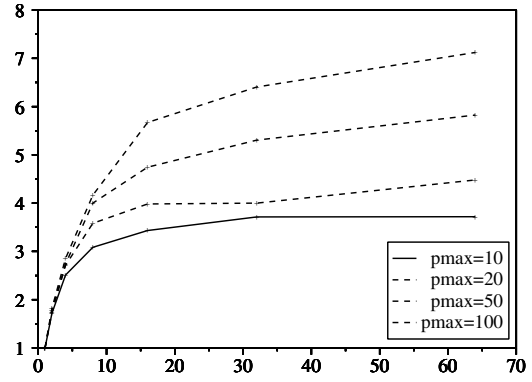

Fig. 11. Speed-up in function of #proc. Speed-up for PCG, 32 blocks, gathering.

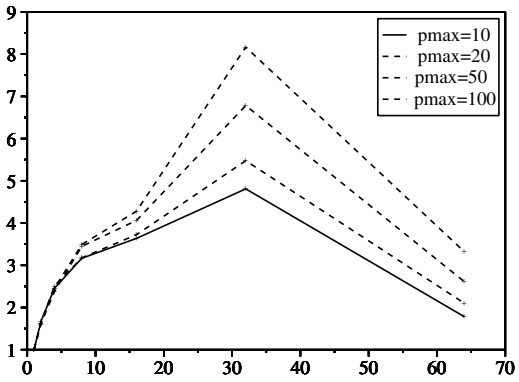

Fig. 12. Speed-up in function of #proc. Speed-up for PCG, 32 blocks, one-sided.

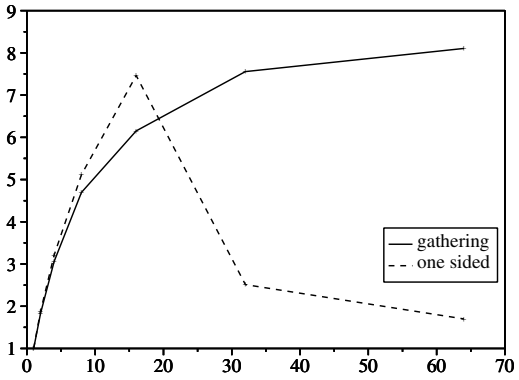

Fig. 13. Speed-up in function of #proc. Speed-up for PCG, 16 blocks, $p_{max} = 100$.

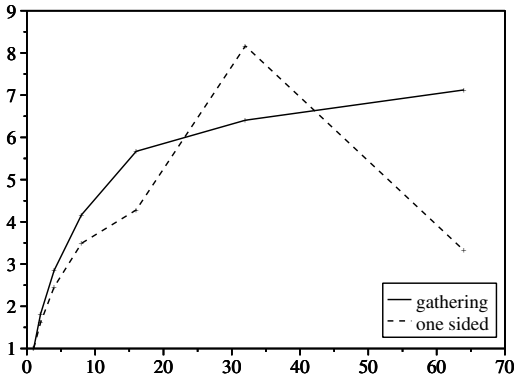

Fig. 14. Speed-up in function of #proc. Speed-up for PCG, 32 blocks, $p_{max} = 100$.

Table 12
Filling of preconditioner matrix

p_{max}	*nnz*		
	50 blocks	100 blocks	200 blocks
10	5,534,416	5,528,927	5,517,830
20	10,555,459	10,534,450	10,492,389
50	25,489,939	25,294,005	24,902,198
100	48,584,507	46,301,395	41,734,029

The numerical results are presented in Tables 13–16.

The corresponding speed-up curves are drawn in Figs. 15–20 (the number p of used processors is represented on X-coordinate and the speed-up S_p on the Y-coordinate).

Table 13
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 10$

p	50 blocks			100 blocks			200 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	25,715	4.14E+03	1.00E+00	26,823	4.31E+03	1.00E+00	28,534	4.57E+03	1.00E+00
10	25,713	1.42E+03	2.91E+00	26,821	1.42E+03	3.03E+00	28,533	1.52E+03	3.00E+00
25	25,713	1.22E+03	3.40E+00	26821	1.26E+03	3.41E+00	28,531	1.34E+03	3.40E+00
50	25,714	1.25E+03	3.31E+00	26,820	1.28E+03	3.37E+00	28,532	1.39E+03	3.30E+00
100	25,714	1.24E+03	3.34E+00	26,820	1.28E+03	3.36E+00	28,532	1.36E+03	3.35E+00
200	25,712	1.25E+03	3.30E+00	26,820	1.29E+03	3.34E+00	28,532	1.41E+03	3.24E+00
PCG Algorithm with one-sided communications									
1	25,715	4.14E+03	1.00E+00	26,823	4.30E+03	1.00E+00	28,534	4.56E+03	1.00E+00
10	25,713	1.31E+03	3.16E+00	26,821	1.36E+03	3.15E+00	28,533	1.36E+03	3.34E+00
25	25,713	1.10E+03	3.76E+00	26,821	1.14E+03	3.75E+00	28,531	1.22E+03	3.74E+00
50	25,714	1.04E+03	3.98E+00	26,820	1.08E+03	3.97E+00	28,532	1.16E+03	3.94E+00
100	25,714	1.35E+03	3.06E+00	26,820	1.06E+03	4.05E+00	28,532	1.14E+03	4.02E+00
200	25,712	1.41E+03	2.93E+00	26,820	1.36E+03	3.15E+00	28,532	1.10E+03	4.15E+00

Table 14
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 20$

p	50 blocks			100 blocks			200 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	21,581	4.39E+03	1.00E+00	23,568	4.61E+03	1.00E+00	27,320	5.32E+03	1.00E+00
10	21,580	1.27E+03	3.44E+00	23,567	1.35E+03	3.43E+00	27,318	1.56E+03	3.42E+00
25	21,581	1.06E+03	4.13E+00	23,567	1.11E+03	4.17E+00	27,318	1.34E+03	3.98E+00
50	21,580	1.06E+03	4.13E+00	23,566	1.16E+03	3.97E+00	27,317	1.35E+03	3.94E+00
100	21,580	1.04E+03	4.21E+00	23,566	1.14E+03	4.04E+00	27,317	1.32E+03	4.04E+00
200	21,580	1.06E+03	4.13E+00	23,566	1.15E+03	4.03E+00	27,318	1.34E+03	3.96E+00
PCG Algorithm with one-sided communications									
1	21,581	4.20E+03	1.00E+00	23,568	4.56E+03	1.00E+00	27,320	5.52E+03	1.00E+00
10	21,580	1.15E+03	3.65E+00	23,567	1.25E+03	3.65E+00	27,318	1.44E+03	3.83E+00
25	21,581	9.64E+02	4.36E+00	23,567	1.05E+03	4.33E+00	27,318	1.23E+03	4.48E+00
50	21,580	8.33E+02	5.04E+00	23,566	9.89E+02	4.61E+00	27,317	1.11E+03	4.97E+00
100	21,580	1.25E+03	3.34E+00	23,566	9.93E+02	4.59E+00	27,317	1.12E+03	4.93E+00
200	21,580	1.35E+03	3.12E+00	23,566	1.39E+03	3.29E+00	27,318	1.07E+03	5.17E+00

Table 15
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 50$

p	50 blocks			100 blocks			200 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	15,560	4.93E+03	1.00E+00	18,703	5.89E+03	1.00E+00	24,069	7.48E+03	1.00E+00
10	15,559	1.10E+03	4.49E+00	18,702	1.51E+03	3.90E+00	24,068	1.68E+03	4.46E+00
25	15,559	8.38E+02	5.89E+00	18,702	9.49E+02	6.20E+00	24,067	1.27E+03	5.90E+00
50	15,559	8.03E+02	6.14E+00	18,702	9.65E+02	6.10E+00	24,067	1.24E+03	6.05E+00
100	15,560	7.68E+02	6.42E+00	18,702	9.24E+02	6.37E+00	24,069	1.19E+03	6.28E+00
200	15,559	7.58E+02	6.50E+00	18,702	9.16E+02	6.42E+00	24,068	1.17E+03	6.39E+00
PCG Algorithm with one-sided communications									
1	15,560	4.95E+03	1.00E+00	18,703	5.90E+03	1.00E+00	24,069	7.52E+03	1.00E+00
10	15,559	1.06E+03	4.67E+00	18,702	1.28E+03	4.62E+00	24,068	1.62E+03	4.64E+00
25	15,559	7.78E+02	6.36E+00	18,702	9.48E+02	6.22E+00	24,067	1.20E+03	6.27E+00
50	15,559	6.73E+02	7.35E+00	18,702	8.46E+02	6.97E+00	24,067	1.07E+03	7.03E+00
100	15,560	1.14E+03	4.35E+00	18,702	8.23E+02	7.16E+00	24,069	1.02E+03	7.37E+00
200	15,559	1.19E+03	4.14E+00	18,702	1.20E+03	4.90E+00	24,068	9.40E+02	7.99E+00

Like for the first matrix, PCG Algorithm with gathering and PCG Algorithm with one-sided communications are compared with $p_{max} = 100$ in Figs. 21–23.

The use of one-sided communications gives better results than the gathering methodology when the numbers of processors is smaller or equal to the number of blocks. In the case of the number of blocks and the number of processors are the

Table 16
Performance evaluation for resolution with PCG Algorithm, preconditioner with $p_{max} = 100$

p	50 blocks			100 blocks			200 blocks		
	It_p	T_p	S_p	It_p	T_p	S_p	It_p	T_p	S_p
PCG Algorithm with gathering									
1	13,289	6.58E+03	1.00E+00	17,529	8.34E+03	1.00E+00	23,781	1.05E+04	1.00E+00
10	13,287	1.17E+03	5.63E+00	17,527	1.52E+03	5.48E+00	23,781	1.97E+03	5.31E+00
25	13,289	8.07E+02	8.16E+00	17,528	1.06E+03	7.86E+00	23,781	1.40E+03	7.49E+00
50	13,287	7.46E+02	8.82E+00	17,528	9.53E+02	8.75E+00	23,781	1.30E+03	8.05E+00
100	13,287	6.86E+02	9.59E+00	17,528	9.73E+02	8.57E+00	23,781	1.21E+03	8.65E+00
200	13,289	6.60E+02	9.97E+00	17,527	8.80E+02	9.48E+00	23,781	1.18E+03	8.89E+00
PCG Algorithm with one-sided communications									
1	13,289	6.61E+03	1.00E+00	17,529	8.37E+03	1.00E+00	23,781	1.05E+04	1.00E+00
10	13,287	1.11E+03	5.93E+00	17,527	1.43E+03	5.87E+00	23,781	1.87E+03	5.62E+00
25	13,289	7.54E+02	8.76E+00	17,528	1.03E+03	8.16E+00	23,781	1.35E+03	7.76E+00
50	13,287	6.79E+02	9.73E+00	17,528	8.66E+02	9.66E+00	23,781	1.15E+03	9.10E+00
100	13,287	1.15E+03	5.75E+00	17,528	7.49E+02	1.12E+01	23,781	1.09E+03	9.63E+00
200	13,289	1.24E+03	5.32E+00	17,527	1.17E+03	7.14E+00	23,781	9.56E+02	1.10E+01

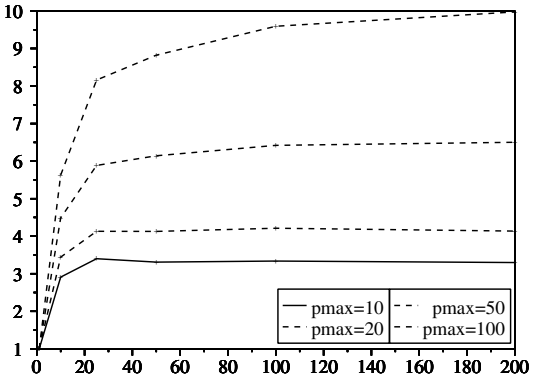

Fig. 15. Speed-up in function of #proc. Speed-up for PCG, 50 blocks, gathering.

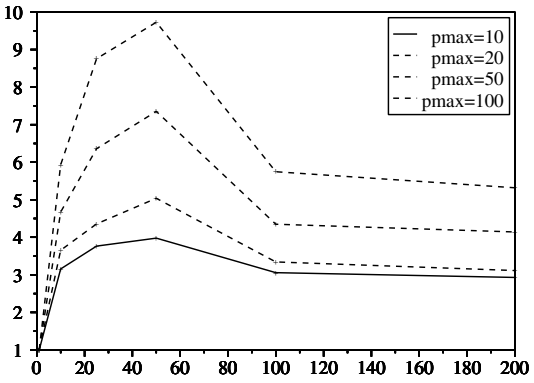

Fig. 16. Speed-up in function of #proc. Speed-up for PCG, 50 blocks, one-sided.

same, the approach of Remark 5.4 allows to save computer time (like for the first matrix), see Tables 17–19 (the number of iterations does not vary when the methodology changes).

5.5.3. Observations and conclusion

Like for the construction (Section 4), when the amount of computations increases, the quality of performance is also increasing. The quality of performance is rather not good here. However, knowing the number of used processors, the block methodology for the class of preconditioner (DIAG+) LS CGS (OPT) allows to get an efficient PCG Algorithm according to Remark 5.4 (see Tables 10, 11, 17, 18 and 19).

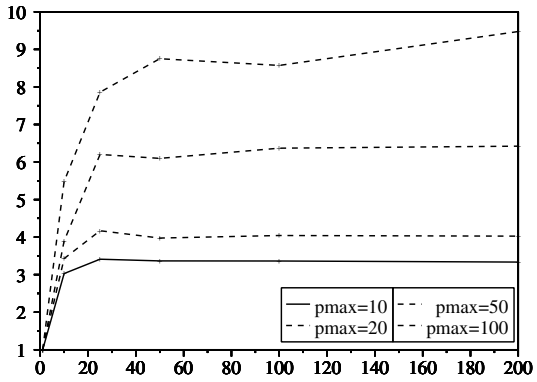

Fig. 17. Speed-up in function of #proc. Speed-up for PCG, 100 blocks, gathering.

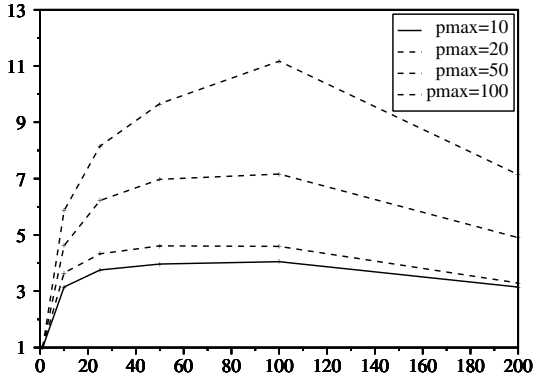

Fig. 18. Speed-up in function of #proc. Speed-up for PCG, 100 blocks, one-sided.

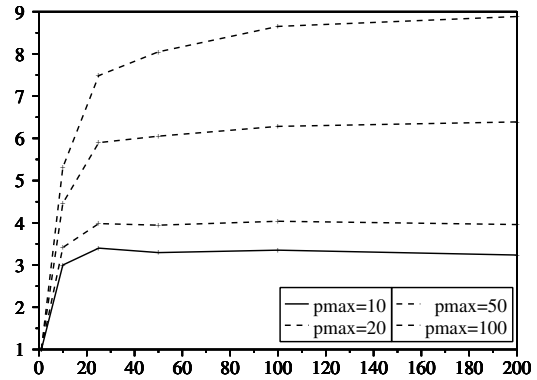

Fig. 19. Speed-up in function of #proc. Speed-up for PCG, 200 blocks, gathering.

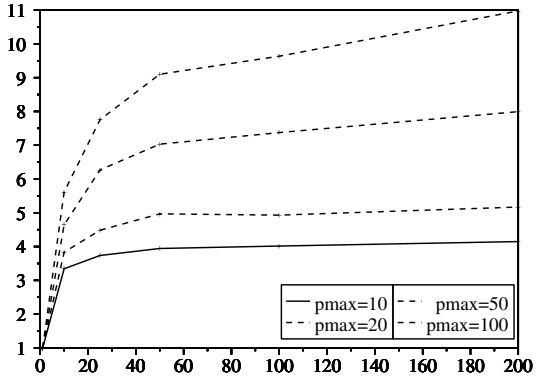

Fig. 20. Speed-up in function of #proc. Speed-up for PCG, 200 blocks, one-sided.

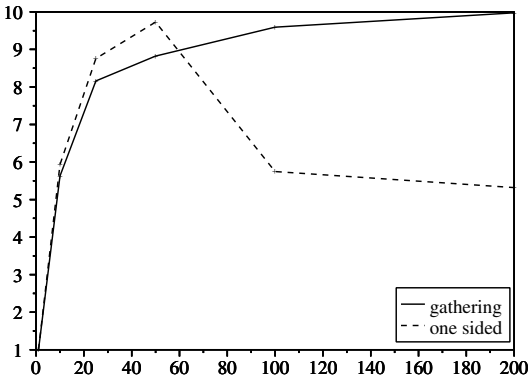

Fig. 21. Speed-up in function of #proc. Speed-up for PCG, 50 blocks, $p_{max} = 100$.

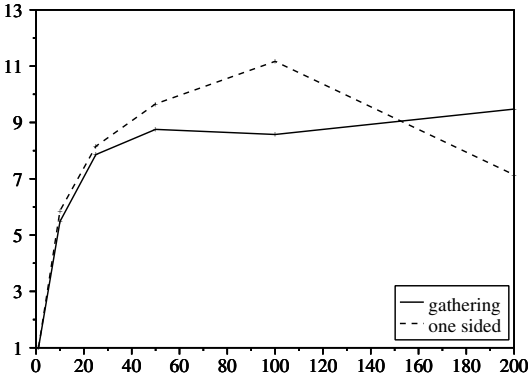

Fig. 22. Speed-up in function of #proc. Speed-up for PCG, 100 blocks, $p_{max} = 100$.

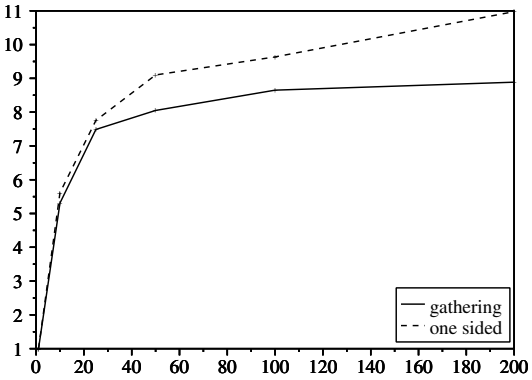

Fig. 23. Speed-up in function of #proc. Speed-up for PCG, 200 blocks, $p_{max} = 100$.

Table 17
Computation time (T_{50}) for 50 blocks and processors

Methodology	$p_{max} = 10$	$p_{max} = 20$	$p_{max} = 50$	$p_{max} = 100$
Gathering	1.25E+03	1.06E+03	8.03E+02	7.46E+02
One-sided communications	1.04E+03	8.33E+02	6.73E+02	6.79E+02
According Remark 5.4	2.98E+02	2.72E+02	2.31E+02	2.46E+02

Table 18
Computation time (T_{100}) for 100 blocks and processors

Methodology	$p_{max} = 10$	$p_{max} = 20$	$p_{max} = 50$	$p_{max} = 100$
Gathering	1.28E+03	1.14E+03	9.24E+02	9.73E+02
One-sided communications	1.06E+03	9.93E+02	8.23E+02	7.49E+02
According Remark 5.4	2.80E+02	2.59E+02	2.65E+02	2.45E+02

Table 19
Computation time (T_{200}) for 200 blocks and processors

Methodology	$p_{max} = 10$	$p_{max} = 20$	$p_{max} = 50$	$p_{max} = 100$
Gathering	1.41E+03	1.34E+03	1.17E+03	1.18E+03
One-sided communications	1.10E+03	1.07E+03	9.40E+02	9.56E+02
According Remark 5.4	3.01E+02	3.12E+02	2.84E+02	2.89E+02

Acknowledgement

I sincerely thank O. Besson for his helpful advices and discussions.

References

[1] M. Benzi, C.D. Meyer, M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput. 17 (5) (1996) 1135–1149.
[2] J. Dongarra, S. Huss-Lederman, S. Otto, M. Snir, D. Walker, MPI – The Complete Reference: The MPI Core, second ed., vol. 1, The MIT Press, 1998.
[3] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, M. Snir, MPI – The Complete Reference: The MPI Extensions, vol. 2, The MIT Press, 1998.
[4] M. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comput. 18 (3) (1997) 838–853.
[5] L.Yu. Kolotilina, A.Yu. Yeremin, Factorized sparse approximate inverse preconditionings: I. Theory, SIAM J. Matrix Anal. Appl. 14 (1) (1993) 45–58.
[6] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Company, 1996.
[7] G. Strang, Linear Algebra and its Applications, Academic Press, Inc., 1976.
[8] J. Straubhaar, Preconditioners for the conjugate gradient algorithm using Gram–Schmidt and least squares methods, Int. J. Comput. Math. 84 (1) (2007) 89–108.