Parallel m-step Preconditioners for the Conjugate Gradient Method ⋆

Cristina Corral [a], Isabel Giménez [a], José Marín [a] and José Mas [a]

[a] *Departament de Matemàtica Aplicada. Universitat Politècnica de València. 46071 València, Spain.*
e-mail: {ccorral,igimenez,jmarinma,jmasm}@mat.upv.es

Abstract

To obtain an efficient parallel algorithm to solve sparse linear systems with the preconditioned conjugate gradient (PCG) method, two types of parallel preconditioners are introduced. The first is a polynomial preconditioner type based on a multisplitting of the matrix system, and the second one is obtained considering only the diagonal blocks of the multisplitting, reducing in this case the communication cost of the algorithm. Therefore it is better for machines with slow communication networks. Its validity as preconditioner is justified theoretically. Indeed, the complexity of the PCG is analyzed using the BSP model *(Bulk Synchronous Parallel)*. Experimental results obtained on a IBM SP2 and a CONVEX SPP1000 using the Oxford BSP Library are reported.

Key words: multisplitting, polynomial preconditioning, conjugate gradient, BSP model.

1 Introduction

1.1 Problem description

The main goal of preconditioning a given linear system

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

⋆ This work was supported by DGES projects No. PB96–1054–C02–01 and No. PB97–0334.

is to reduce the amount of time needed to get a good approximation of the solution by applying an iterative method (tipically a Krylov subspace method). Thus, the iterative method is applied to the new preconditioned system

$$M^{-1}Ax = M^{-1}b \,, \quad \text{or} \quad AM^{-1}y = b \,, \quad x = M^{-1}y \tag{2}$$

where M is the preconditioning matrix. If the preconditioned matrix $\hat{A} = M^{-1}A$ (or $\hat{A} = AM^{-1}$) has a better condition number than A or its eigenvalues are better clustered, then a small number of iterations are needed to achieve convergence to the solution.

When A is a symmetric positive definite matrix (*spd*), one of the most competitive and widely used iterative methods is the Preconditioned Conjugate Gradient (PCG) [8,14]. In this case the preconditioning matrix M has to be also symmetric and positive definite, and the preconditioning is done by solving an auxiliary system

$$Ms = r \tag{3}$$

where $r = b - Ax$ is the residual corresponding to the approximate solution obtained in the previous iteration. Since the solution of this additional system represents an overhead, it is crucial that the preconditioner will be easy to apply. On the other hand, if M is close to A in some sense, a faster convergence to the solution of (1) is expected. Thus, since in general both requirements are in disagreement, the choice of a good preconditioner is far from trivial. Several preconditioners have been developed in the literature [15]. Preconditioners based on incomplete factorizations of the original matrix A, as ILU type preconditioners [9], are widely used, but they are notoriously difficult to implement on a parallel architecture. *Polynomial preconditioners* are inherently parallel, and therefore a good choice for parallel environments. Among those, preconditioners based on Chebyshev Polynomials have the disadvantage that determining the optimal iteration parameters is difficult in practice. Polynomial preconditioners based on a *splitting* of A, $A = P - Q$, where P is nonsingular, save this disadvantage. They are based on the Neumann expansion. If $H = P^{-1}Q$ and $\rho(H) < 1$ (where $\rho(H)$ denotes the spectral radius of H) then (see [14, Theorem 3.4.1]) one has that

$$A^{-1} = \left(\sum_{k=0}^{\infty} H^k \right) P^{-1}. \tag{4}$$

The Neumann expansion (4) suggests to take as approximation to A the matrix given by a partial sum:

$$M_m = P \left(I + H + H^2 + \cdots + H^{m-1} \right)^{-1}, \quad m = 1, 2, 3, \ldots \tag{5}$$

This matrix is called the *m-step polynomial preconditioner* [1]. Depending on the splitting, specific preconditioners are obtained. Now, the system (3) can be solved by doing m steps on the iterative scheme

$$Ps^{(i)} = Qs^{(i-1)} + r, \quad i = 1, 2, \ldots, m \tag{6}$$

with $s^{(0)} = 0$.

A generalization of the preconditioner (5) is introduced in [2] using two splittings of A, $A = P_l - Q_l, \quad l = 1, 2$, and taking the solution s of (3) as the average of the solutions of (6) obtained by considering both splittings. The *m-step polynomial additive preconditioner* verifies

$$M_m^{-1} = \frac{1}{2}\left(I + H + H^2 + \cdots + H^{m-1}\right)\left(P_1^{-1} + P_2^{-1}\right), \tag{7}$$

$$H = \frac{1}{2}\left(P_1^{-1}Q_1 + P_2^{-1}Q_2\right).$$

In [2] the splittings $A = P_l - Q_l$ are chosen in such a way that the method used corresponds to the SSOR method.

To obtain efficient algorithms in parallel systems, a new generalization is introduced in [5] and [6] using *multisplittings*: given k splittings $A = P_l - Q_l, \quad l = 1, 2, \ldots, k$, of A, the m-step preconditioner is defined by

$$M^{-1} = \frac{1}{k}\left(I + H + \cdots + H^{m-1}\right)\left(P_1^{-1} + P_2^{-1} + \cdots + P_k^{-1}\right), \tag{8}$$

$$H = \frac{1}{k}\sum_{l=1}^{k} P_l^{-1}Q_l.$$

Here we deal with this type of polynomial preconditioner, using different multisplittings of the matrix A. The multisplittings are chosen in such a way that the solution of the system in the preconditioning step is done completely in parallel by solving several subsystems independently. On the other hand, to reduce the computation time these subsystems are chosen to be triangular or diagonal. The complexity of the resulting algorithms is analyzed by using the BSP model, briefly described in the next subsection.

1.2 BSP model

The complexity of the algorithms presented in this paper is analyzed under the assumptions of the BSP model (Bulk Synchronous Parallel) [10,11].

This model has an increasing popularity and it represents a realistic model for modern parallel computers. It was introduced by Valiant [16] with the purpose to play the same role as the traditional von Neumann model for sequential computation. The model provides an unified framework for the design and programming of general purpose parallel computing systems. A BSP computer consists of a set of processor-memory pairs, a global communication network, and a mechanism for synchronising all the processors. To execute a parallel algorithm, a BSP computer proceeds in phases or supersteps. In a superstep each processor can perform a number of arithmetic operations and communicate a number of messages, followed by a barrier synchronisation. An important feature of the model is the ability to easily asses the cost of parallel algorithms. For this purpose provides a set of parameters which characterize the performance of a BSP computer, abstracting any particular parallel architecture: s the processor speed in flop/s, ℓ the number of flops for barrier synchronisation and g the number of flops required for all processors to simultaneously communicate a single word. In this paper we will consider flops as the cost unit. The BSP cost of a parallel superstep is decomposed into three factors corresponding each one to the different actions that any processor can perform: (ω) the maximum number of arithmetic operations performed by each processor. $(h \cdot g)$ the communication cost, where h is the largest amount of data either entering or leaving a single processor. In the BSP terminology this is the cost of realizing an *h-relation*. An h-relation denotes a routing problem where each processor has at most h words of data to send to other processors and where each processor is also due to receive at most h words of data from other processors. And finally (ℓ) the cost for synchronising all the processors. Therefore, the cost of the algorithm is simply the sum of the costs of each superstep and can be expressed as,

$$\begin{aligned} \mathrm{cost(flops)} &= \mathrm{comp\ cost} + \mathrm{comm\ cost} + \mathrm{synch\ cost} \\ &= \Omega + H \cdot g + \gamma \cdot \ell \end{aligned} \tag{9}$$

where Ω is the sum of local computations in each superstep, H is the sum of the h-relations in each superstep and γ is the number of supersteps needed by the algorithm to conclude. The cost in seconds is obtained dividing the equation (9) by the parameter s.

2 Multisplittings

The implementation of the PCG algorithm with a polynomial preconditioner based on a multisplitting (8), can be done as follows.

Algorithm 1 Sequential Algorithm: m steps, k splittings

(1) *Compute first residual* $(r := r^0)$
(2) *Preconditioner: solve* $Ms = r$
$i := 0,\ s_i := 0,$
REPEAT
Solve systems $P_l t_l = Q_l s_i + r, \quad l = 1, 2, \ldots, k$
$i := i + 1$
$s_i := \frac{1}{k} \sum_{l=1}^{k} t_l$
UNTIL $i =$ *preconditioner steps number* (m)
$s^0 := s_m$
(3) *Compute first direction* $v^0 := s^0$ *and* (s^0, r^0)
(4) *Iterations: FOR* $n = 0, 1, \ldots$ *UNTIL convergence*
(4a) *Compute new residual* (r)
$\alpha := (s^n, r^n)/(v^n, Av^n)$
$r^{n+1} = r^n - \alpha A v^n$
(4b) *Preconditioner: solve* $Ms = r$ *(Equal step (2))*
(4c) *Compute new solution* (x)
$x^{n+1} = x^n + \alpha v^n$
(4d) *Convergence test*
if $(s^{n+1}, r^{n+1}) < \epsilon$ *then*
if $(r^{n+1}, r^{n+1}) < \epsilon$ *then END*
(4e) *Compute new direction* (v)
$\beta := (s^{n+1}, r^{n+1})/(s^n, r^n)$
$v^{n+1} := s^{n+1} + \beta v^n$

where (,) denotes the usual inner product. To implement in parallel the preconditioning steps **(2)** and **(4b)** in the above algorithm, in [5] the authors propose the choice of $2p$ splittings such that $P_{2l} = P_{2l-1}^T$, with the nonzero entries of P_{2l-1} defined by

$$p_{ij} = a_{ij} \quad \text{if} \begin{cases} i = j & \text{or} \\ i < j & i, j \in \{q(l-1)+1, \ldots, ql\} \end{cases}, \tag{10}$$

where p is the number of processors. For simplicity we assume that the matrix size is $n = pq$. Therefore A is partitioned into blocks of contiguous rows such that each block A_i contains the rows with indexes $\{q(i-1)+1, \ldots, qi\}$. Figure 1 illustrates these splittings for a 9×9 block matrix and 3 processors. Here L_{ii}, U_{ii}, D_{ii} are the lower triangular, upper and diagonal triangular part of the square diagonal block matrix A_{ii}, with size $q \times q$.

$$P_1 = \begin{pmatrix} L_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \quad P_3 = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \quad P_5 = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$P_2 = \begin{pmatrix} U_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \quad P_4 = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & U_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \quad P_6 = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & U_{33} \end{pmatrix}$$

Fig. 1. *First type of splittings*

In [6] Corral defines another preconditioner taking two splittings such that the nonzero entries of P_1 are defined by

$$p_{ij} = a_{ij} \quad \text{if} \begin{cases} i = j & \text{or} \\ i < j & i, j \in \{q(t-1)+1, \ldots, qt\} \quad t = 1, 2, \ldots, p \end{cases} \tag{11}$$

and $P_2 = P_1^T$. (Note that for $k = 2$, the definition of M in (8) coincides with (7), however the preconditioner presented in [5] and [6] are not the same as in [2] because the splittings (10) or (11) are different. Only for the sequential case, when $p = 1$, both preconditioners coincide with the SSOR preconditioner presented in[2]). Figure 2 shows the multisplitting defined by (11) for $p = 3$.

$$P_1 = \begin{pmatrix} L_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ 0 & 0 & L_{33} \end{pmatrix} \quad P_2 = \begin{pmatrix} U_{11} & 0 & 0 \\ 0 & U_{22} & 0 \\ 0 & 0 & U_{33} \end{pmatrix}$$

Fig. 2. *Second type of splittings*

2.1 Complexity of the sequential algorithm

In the remainder of the paper it will be assumed that A is a large and sparse matrix of size $n \times n$ with density d, such that there are dn^2 nonzero elements in A evenly distributed. In particular, there are aproximately dn nonzero elements on a given row of the matrix. Because the iterative nature of the method we will compute only the BSP complexity by iteration, without taking into account the initialization steps **(1)**, **(2)** and **(3)**.

Phase 4a. The computation of α takes one matrix-vector product ($(2dn - 1)n$), one inner product ($2n-1$) and one division. To compute r^{k+1} one multiplication and one addition per element are required ($2n$). Hence, this phase requires $2dn^2 + 3n$ flops.

Phase 4b. In each step of the preconditioner must be solved two triangular systems of the form $U_l t_l = Q_l s_i + r$ and $L_l t_l = Q_l s_i + r$. Each system requires one triangular matrix-vector product $(dn^2+(d-1)n)$ one vector-vector addition (n) and the solution of one triangular system $(dn^2+(d-1)n)$. Moreover, the computation of $s_i=(t_1+t_2)/2$ requires one addition and one division per element $(2n)$. Hence, the complexity for this phase is $m(4dn^2+4dn)$, where m is the preconditioner steps number.

Phase 4c. The computation of a new solution requires the same number of operations than those required to compute a new residual, i.e, $2n$.

Phase 4d. To test if the method has reached a good solution, two inner products and two conditions are performed in the worst case. Hence, this phase requires at most $4n$ flops.

Phase 4e. To compute a new direction, $2n+1$ operations are required.

Adding the cost of each phase we have the following result.

Theorem 1 *The sequential algorithm 1 requires at most* $(4m+2)dn^2+(4dm+11)n+1$ *arithmetic operations by iteration.*

When n is large the cost can be approximated by $(4m+2)dn^2$. Hence, the complexity of the sequential algorithm is $O(mdn^2)$. Indeed, in most cases the density of the matrix is $O(n^{-1})$ (for instance, in most banded systems), such that the complexity in this case is $O(mn)$.

3 Parallel Algorithm

3.1 Description

To implement the PCG algorithm on a multiprocessor system (with distributed or shared memory) the matrix A is partitioned such that the processor j stores or has access to one block of q consecutive rows with indexes $\{q(j-1)+1,\ldots,qj\}$. The residual r, the new solution x, and the vectors s and v, are distributed in the same way. Moreover, each processor stores the parameters α and β. Because all the vectors are distributed, the memory requirement for matrices with density $O(n^{-1})$ is $O(n/p)$. In [3] an optimal BSP algorithm for the matrix–vector product with a different distribution of the system matrix is proposed. In this work we focus in the preconditioning step. Thus, we use a row block distribution for simplicity and clarity. The results presented could be easily extended to the distributions presented therein. Moreover, for banded systems (see section 5) this distribution is near to optimal.

The parallel algorithm for the processor j is as follows.

Algorithm 2 *Parallel Algorithm: m steps, k splittings*

(1) *Compute first residual* $(r_j := r_j^0)$
(2) *Preconditioner: solve* $Ms = r$
(3) *Compute first direction* $v^0 := s^0$ *and* (s^0, r^0)
(4) *Iterations: FOR* $n = 0, 1, \ldots$ *UNTIL convergence*
(4a) *Compute new residual* (r)
compute and broadcast $(v_j^n, A_j v^n)$
compute $\alpha = (s^n, r^n) / \sum_{j=1}^{p} (v_j^n, A_j v^n)$
$r_j^{n+1} = r_j^n + \alpha A_j v^n$
(4b) *Preconditioner: solve* $Ms = r$ *(Equal step (2))*
(4c) *Compute new solution* (x)
$x_j^{n+1} = x_j^n - \alpha v_j^n$
(4d) *Convergence test*
compute and broadcast (s_j^{n+1}, r_j^{n+1})
compute $(s^{n+1}, r^{n+1}) = \sum_{j=1}^{p} (s_j^{n+1}, r_j^{n+1})$
if $(s^{n+1}, r^{n+1}) < \epsilon$ *then*
compute (r^{n+1}, r^{n+1}) *(similar to the previous test)*
if $(r^{n+1}, r^{n+1}) < \epsilon$ *then END*
(4e) *Compute new direction* (v)
$\beta := (s^{n+1}, r^{n+1}) / (s^n, r^n)$
$v_j^{n+1} := s_j^{n+1} + \beta v_j^n$
send v_j^{n+1}.

After the computation of v_j^{n+1} in step **(4e)**, to exploit sparsity for communication, only the elements needed by other processors are sent. The same holds in steps **(2,7)** and **(2,5)** for the application of the preconditioner shown bellow. The solution of the system in step **(2)** and **(4b)** for the splittings (10) and (11) is performed in parallel. Denoting by the superscript (j) the corresponding matrix row block stored in processor j, and (jj) the corresponding square diagonal block, processor j has to perform, in the step i of the preconditioner, the following computations.

Algorithm 3 *Parallel Preconditioner: splittings (10)*

(2,1) *Compute* $v^{(j)} = Q_D^{(j)} s_i + r^{(j)}$
(2,2) *Compute* $\omega^{(j)} = Q_L^{(j)} s_i + r^{(j)}$
(2,3) *Compute* $z^{(j)} = Q_U^{(j)} s_i + r^{(j)}$
(2,4) *Solve system* $D^{(jj)} s_D^{(j)} = v^{(j)}$
(2,5) *Solve system* $L^{(jj)} s_L^{(j)} = \omega^{(j)}$
(2,6) *Solve system* $U^{(jj)} s_U^{(j)} = z^{(j)}$
(2,7) *Compute and send* $s_{i+1}^{(j)} = \frac{1}{2p} \{ s_L^{(j)} + s_U^{(j)} + 2(p-1) s_D^{(j)} \}$

In the algorithm $D^{(jj)}$, $L^{(jj)}$, and $U^{(jj)}$ are respectively the diagonal, lower and upper part of the square diagonal block $A^{(jj)}$ stored by each processor (see, for instance, Figure 1). Therefore, $Q_D^{(j)} = A^{(j)} - D^{(jj)}$, and similarly for $Q_L^{(j)}$ and $Q_U^{(j)}$. For the second type of splittings (11), the diagonal system is not computed. Therefore steps **(2,1)** and **(2,4)** are omitted.

Algorithm 4 *Parallel Preconditioner: splittings (11)*

(2,1) *Compute* $\omega^{(j)} = Q_L^{(j)} s_i + r^{(j)}$
(2,2) *Compute* $z^{(j)} = Q_U^{(j)} s_i + r^{(j)}$
(2,3) *Solve system* $L^{(jj)} s_L^{(j)} = \omega^{(j)}$
(2,4) *Solve system* $U^{(jj)} s_U^{(j)} = z^{(j)}$
(2,5) *Compute and send* $s_{i+1}^{(j)} = \frac{1}{2}\{s_L^{(j)} + s_U^{(j)}\}$

3.2 Complexity

As in the sequential algorithm, we shall compute only the complexity by iteration.
Phase 4a. To compute α one matrix-vector product, one partial inner product and one division are required $(2dn^2/p + n/p)$. Moreover the partial inner product is broadcast to all the other processors and they are added on each processor $(p - 1 + g(p-1) + \ell)$. Finally to calculate the residual one multiplication and one addition are performed per element $(2n/p)$. Hence, the total cost for this step is $2dn^2/p + 3n/p + p - 1 + g(p-1) + \ell$.
Phase 4c. One multiplication and one addition are required per element, hence the cost is $2n/p$.
Phase 4d. In the worst case two partial inner products, two broadcasts, $2(p-1)$ additions and two conditions are required. Then the cost for this phase is at most $4n/p + 2p + 2 + 2g(p-1) + 2\ell$.
Phase 4e. The computation of β only requires one division. To compute v_j one multiplication and one addition are required per element $(2n/p)$. Hence, the arithmetic cost is $2n/p + 1$. The cost of sending v_j, assuming the worst case corresponding to broadcast all the elements, is $g(p-1)n/p + \ell$. For structured sparsity patterns it is possible to specify smaller bounds. In section 5, the special structure of banded systems is studied. The total cost of this phase is $2n/p + 1 + g(p-1)n/p + \ell$.

Phases (**2**) and (**4b**) correspond to the computation of the preconditioner. For the first type of splittings (10) its complexity analysis is as follows.
Phase (2,1). The computation of the matrix-vector product requires $(2dn - 1)n/p$ operations and the addition of the two vectors of length n/p requires n/p arithmetic operations.

Phases (2,2) and (2,3). The cost of computing $\omega^{(j)}$ ($z^{(j)}$) is the same of the step (2,1), but substracting the cost of computing a triangular matrix-vector product. Hence, we have a total cost of $2(2dn^2/p - (d(n/p)^2 + (d-1)n/p))$.
Phase (2,4). There is one division per element. Hence, the cost is n/p.
Phases (2,5) and (2,6). The solution of the two triangular systems requires $2(d(n/p)^2 + (d-1)n/p)$ arithmetic operations.
Phase (2,7). The computation of s_{i+1} requires two multiplications and two additions per element ($4n/p$). Moreover, each processor broadcast s_{i+1} in the worst case. Hence, the total cost of this step is $4n/p + g(p-1)n/p + \ell$.

The total cost for this preconditioner is $m(6dn^2/p+(4d+1)n/p+g(p-1)n/p+\ell)$. For the second type (11), proceeding in a similar way the complexity is computed as $m(4dn^2/p + (4d-1)n/p + g(p-1)n/p + \ell)$. Adding the cost of each step we can stablish the following result.

Theorem 2 *Each iteration of the parallel implementation of the PCG (algorithm 2) requires at most* $(6m+2)dn^2/p+((4d+1)m+11)n/p+3p+2+g(p-1)(3+(m+1)n/p)+(m+4)\ell$ *arithmetic operations with the preconditioner defined by the multisplitting (10), and* $(4m+2)dn^2/p+((4d-1)m+11)n/p+3p+2+g(p-1)(3+(m+1)n/p)+(m+4)\ell$ *arithmetic operations with the preconditioner defined by the multisplitting (11).*

These costs for large matrices can be approximated by

$$(6m+2)dn^2/p + g(p-1)(m+1)n/p + (m+4)\ell \tag{12}$$

$$(4m+2)dn^2/p + g(p-1)(m+1)n/p + (m+4)\ell \tag{13}$$

Hence, each iteration of the parallel algorithm is performed in $O(mdn^2/p)+$ $O(mng)+$ $O(m\ell)$ time. Moreover, if we consider that the matrix density is $O(n^{-1})$ then the time required is $O(mn/p)+O(mng)+O(m\ell)$. Based on theorems (1) and (2), the speedup expressions are given by

$$S_{PP1} \approx \frac{d(4m+2)n^2}{(6m+2)dn^2/p + g(p-1)(m+1)n/p + (m+3)\ell} \approx \frac{p}{1.5 + \frac{pg}{4dn}} \tag{14}$$

for the preconditioner (10), and by

$$S_{PP2} \approx \frac{d(4m+2)n^2}{(4m+2)dn^2/p + g(p-1)(m+1)n/p + (m+3)\ell} \approx \frac{p}{1 + \frac{pg}{4dn}} \tag{15}$$

for the preconditioner (11). Thus, a good BSP efficiency depends on keeping low the pg/nd ratio. Thus, it depends of the matrix sparsity. Note that when d is $O(n^{-1})$ and g is $O(1)$, $S_P \approx \dfrac{p}{p+1}$ and no actual speedup can be gained.

Moreover, from (14) we see that with the preconditioner given by (10) the maximum efficiency of the parallel program can never be more than 0.67.

4 Modifying the preconditioner

As we have already seen, the communication and synchronisation costs relative to solve the system (3) in step **(2)** and **(4b)** of the parallel algorithm are proportional to m. One way to improve the execution time of the algorithm is modifying the preconditioner in such a way that the communication in step **(2,7)** for the first type of preconditioner (**(2,5)** for the second type) can be avoided.

Let A be a symmetric and positive definite matrix. Consider the matrix A partitioned in square blocks of size $q = n/p$, $A = (A_{ij})$, $i, j = 1, 2, \ldots, p$. Let $A_0 = \mathrm{diag}(A_{ii})$ be the matrix obtained from A canceling the off diagonal blocks. Obviously, A_0 is also symmetric and positive definite. Therefore, if an additive polynomial preconditioner of m steps is defined for the matrix A_0, it could be also used as preconditioner for the matrix A.

Theorem 3 *Let A be a symmetric and positive definite matrix, let $A_0 = \mathrm{diag}(A_{ii})$ and let $A_0 = P_l - Q_l$, $l = 1, 2, \ldots, 2k$, be a multisplitting of A_0 satisfying $P_{2i} = P_{2i-1}^T$ for $i = 1, 2, \ldots, k$, and suppose that the matrices P_l are positive definite. Then the matrix M^{-1} given by (8) is symmetric and positive definite if*
(i) m is odd, or
(ii) m is even and $\rho(H) < 1$.

PROOF. Apply theorems 1, 2 and 3 in [5] to A_0. □

With this preconditioner, the matrices Q_l have zero elements in the off diagonal blocks. Thus, each processor computes q elements of the vector s independently, and the communication is reduced. Note that, whith the new multisplitting chosen, the matrices $Q_l^{(j)}$ are triangular. The modified algorithm remains almost unchanged. Only in the preconditioning phase s_i is replaced by $s_i^{(j)}$, and the steps **(2,7)**,**(2,5)** in Algorithms 3 and 4 are replaced by:

(2,7')(2,5') Compute $s_{i+1}^{(j)}$. If $i + 1 = m$ then send $s_m^{(j)}$.

In general, with this modification M is less close to A than the original preconditioner, and therefore an increment in the number of iterations is expected. Thus, the execution time can be worse, depending on the speed of the computer communication network. Nevertheless, if $A \approx A_0$, or the zeroed elements of A are very small compared with the other ones, the increase in the number of iterations could be insignificant, such that it could be balanced by the reduction in communication time. In this context, the modification can be advantageous.

4.1 Complexity

With the described modification, only the terms corresponding to the communication and synchronisation costs will change. For both preconditioners this cost is $g(p-1)n/p+\ell$. Therefore, the complexity for the parallel algorithm is given by

Theorem 4 *Each iteration of the parallel implementation of the PCG (algorithm 2) requires at most* $(6m+2)dn^2/p + ((4d+1)m+11)n/p + 3p + 2 + g(p-1)(3+2n/p) + 5\ell$ *arithmetic operations with the preconditioner defined by the multisplitting (10) and the matrix* A_0*, and* $(4m+2)dn^2/p + ((4d-1)m+11)n/p + 3p + 2 + g(p-1)(3+2n/p) + 5\ell$ *arithmetic operations with the preconditioner defined by the multisplitting (11) and the matrix* A_0*.*

As in the previous section, these costs can be approximated by

$$(6m+2)dn^2/p + 2g(p-1)n/p + 5\ell \tag{16}$$

$$(4m+2)dn^2/p + 2g(p-1)n/p + 5\ell. \tag{17}$$

Each iteration is performed in $O\ (mdn^2/p)+\ O(ng)+\ O(\ell)$. If d is $O(n^{-1})$ then the time is $O(mn/p)+\ O(ng)+\ O\ (\ell)$. The speedup expressions in this case are similar to the equations (14) and (15) but setting $m=1$ in the communication and synchronisation terms. Thus, for the modified versions we have

$$S_P \approx \frac{p}{1.5 + \frac{pg}{4dmn}} \tag{18}$$

for the splitting (10), and

$$S_P \approx \frac{p}{1 + \frac{pg}{4dmn}} \tag{19}$$

for the splitting (11). In this case, if d is $O(n^{-1})$ and $m \gg pg$, then $S_P \approx p$ for the second splitting. Unfortunately, for the type of matrices studied in this paper the number of steps that yields the best execution time is usually $O(1)$, as we will see in the next section, and therefore that optimality is hard to reach. Comparing the new modified preconditioners with the original ones, one observes that there is an improvement in the communication time, proportional to both, m the number of steps of the preconditioner, and g, the cost of communicating a message, i.e. the speed of the communication network. Therefore, this change will be interesting for high values of m and g , since in this case we can expect a smaller execution time. In the next section we study the use of these preconditioners for a large, sparse and banded matrix. More concretely, we report the observed times for computers with different values of the parameter g, when A corresponds to the five point discretization of the *Laplace* equation on a square domain. Moreover, we will show how to exploit the sparsity pattern of the matrix to improve the parallel algorithm.

5 Experimental results

When the matrix is sparse and banded, some savings in the computation and specially in the communication can be obtained. In this section, a modification of the parallel algorithm for this type of matrices is studied. Moreover, the results obtained on a IBM SP2 and a CONVEX SPP1000 are presented. The problem considered corresponds to the discretization of the *Laplace* equation, which is known to be a block tridiagonal matrix $A = \text{tridiag}(-I, B, -I)$, where I is the identity matrix and $B = \text{tridiag}(-1, 4, -1)$. Thus, A is a banded and pentadiagonal matrix. In this case, A_0 is close but not equal to A. For a given matrix size, in all cases the same initial aproximation and solution was used, and the same holds for the stopping criterion. The parallel code was implemented using the Oxford BSP Library [12].

The preconditioners described by (10) and (11) are symmetric and positive definite matrices since the matrix satisfies the conditions of the following result.

Theorem 5 *Let A be a symmetric diagonally dominant matrix with positive diagonal entries, then the splittings of A_0 defined by (10) and (11) are P-regular (that is the symmetric part of $P + Q$ is positive definite). In addition the matrix M^{-1} defined by (8) is a symmetric positive definite matrix.*

PROOF. A_0 is also a symmetric diagonally dominant matrix, and then [6, Theorem 2.7] or [5, Theorem 4] can be applied. Then by [13, Theorem 1 (b)] the conditions in Theorem 3 are satisfied and the preconditioner is a symmetric positive definite matrix. $\square$

Banded systems usually arise from the discretization of PDE's. Due to theirs special structure, it is possible to reduce the amount of messages communicated in the preconditioning step. In fact, if δ is the matrix bandwidth, to compute a matrix-vector product the processor j only needs the elements of the vector with indexes $\in \{q(j-1)+1-\delta, \ldots, qj+\delta\}$. Thus, assuming that the bandwidht is smaller than the number of elements per processor, i.e, $\delta < n/p$, the steps **(2,7)** (Algorithm 3) and **(2,5)** (Algorithm 4) can be changed as follows,

(2,7),(2,5) Compute $s_{i+1}^{(j)}$ and send it to processors $j-1,\ j+1$.

Note that processors 1 and p only communicate with processor 2 and $p-1$ respectively. The communication cost changes to $2mg\delta$. Similarly, for the modified preconditioners the algorithm changes to

(2,7'),(2,5') Compute s_{i+1}^{j}. If $i+1 = m$ then send s_{i+1}^{j} to processors $j-1,\ j+1$.

This step has a cost of $2g\delta$. On the other hand, for most banded matrices the bandwidth δ is $O(\sqrt{n})$ (for instance, for the *Laplace* equation $\delta = \sqrt{n}$). If we assume that the matrix density is $O(n^{-1})$, we can establish the following result.

Theorem 6 *Let A be a large and sparse banded matrix with bandwidth $O(\sqrt{n})$ and density $O(n^{-1})$. Then, each iteration of the parallel algorithm 2 can be performed in $O(mn/p)$+ $O(m\sqrt{n}g)$+ $O(m\ell)$ time for the preconditioners (10) and (11), and in $O(mn/p)$+ $O(\sqrt{n}g)$+ $O(\ell)$ time for the modified preconditioners respectively.*

PROOF. Taking $d \approx n^{-1}$, $\delta \approx \sqrt{n}$ and the communication cost equal to $2mg\delta$ in the approximate cost expressions given by equations (12) and (13). For the modified preconditioners we proceed similarly but taking the communication cost equal to $2g\delta$ in the equations (16) and (17). $\square$

The speedup can be approximated by

$$S_P \approx \frac{p}{1.5 + \frac{pg}{2dn\sqrt{n}}} \qquad \text{and} \qquad S_P \approx \frac{p}{1 + \frac{pg}{2dn\sqrt{n}}} \tag{20}$$

for the preconditioners (10) and (11) respectively. Again, if d is $O(n^{-1})$ and $\sqrt{n} \gg pg$, $S_P \approx p$ for the preconditioner (11). Thus, the algorithm is optimal and good speedups can be obtained (see Figure 3). Note that these conditions hold in most banded systems. For the modified versions we have

$$S_P \approx \frac{p}{1.5 + \frac{pg}{2dmn\sqrt{n}}} \qquad \text{and} \qquad S_P \approx \frac{p}{1 + \frac{pg}{2dmn\sqrt{n}}}, \tag{21}$$

N	m	IBM SP2 Number Iterat.	IBM SP2 time(sec)	CONVEX SPP1000 time(sec)
6400	1	121	2.1	3.90
	2	49	2.41	3.60
20736	1	216	11.4	16.06
	2	77	12.4	16.78
40000	1	351	36.01	46.52
	2	106	32.68	39.51
65536	1	489	83.02	114.9
	2	134	68.8	93.96

Table 1
Observed time on a IBM SP2 and a CONVEX SPP1000 for 1 processor. N stands for the matrix size and m for the number of preconditioner steps.

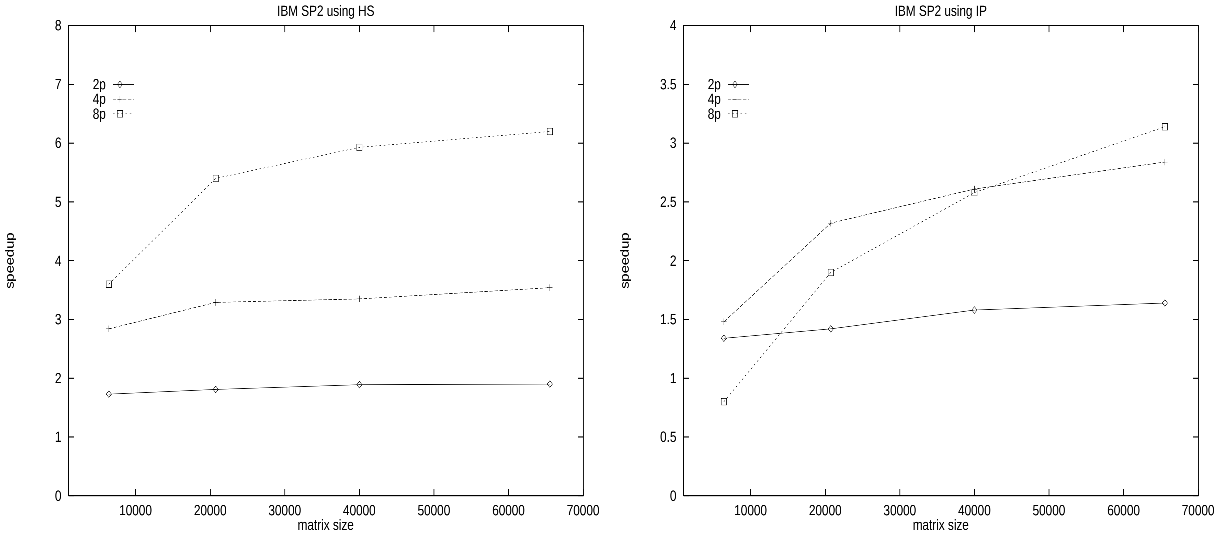

Fig. 3. Observed speedup for the parallel algorithm on an IBM SP2 using the high performance switch (left-hand side) and the ethernet type network (right-hand side).

and similar conclusions hold. In this case, due to the presence of the parameter m, a better speed up is expected to reach. An important constraint to be mentioned is that, assuming $\delta < n/p$, for $\delta = \sqrt{n}$ the number of processors should be less than $\sqrt{n}$.

Tables 1 and 2 show the execution times measured on a IBM SP2 and a CONVEX SPP1000 for the preconditioner given in (11). The observed time for the preconditioner given in (10) has not been shown since it performed worse. For the case of the IBM SP2, two different communication networks were used. The high performance switch, denoted by HS, and the ethernet type network, denoted by IP. The values of the parameter g range from $O(1)$*flops/word* for IBM SP2 using switch, to $O(10)$*flops/word* and $O(100)$*flops/word* for the CONVEX SPP1000 and IBM SP2 using ethernet respectively. These values of g explain the observed times from the point of view of the BSP model. One can see that for the SP2 using HS, the best time corresponds to the original algorithm in all cases. Due to the small value of g in this case, the increment in the number of iterations for the modified algorithm leads to an increase in the arithmetic time bigger than the reduction in the communication cost. Note that the smallest time is obtained for 8 processors and a number of steps of the preconditioner $m = 2$ in almost all cases. On the other hand, when the network IP was used, we observe that the modified algorithm gives always the best results, due mainly to both, the high value of g and the fact that the values of m leading the best time are bigger. Thus, the reduction in the communication time compensates the increase of the arithmetic one. This fact can be observed also on the CONVEX SPP1000. It is worth noticing that, unlike in the SP2 HS case in which the best time is always obtained for a number of eight processors, when using the ethernet type network, the number of processors that yields the best execution time is not always the same, and decrease with the matrix size (see Figure 3). This is also true on the CONVEX, and it shows that for not scalable machines (with high value of g) a bigger number of processors is not synonym of a better execution time. On the other hand, the speedup tends to be p when the matrix size increases on IBM SP2 using HS. In contrast, using the IP network the observed speedup is worse. The explanation comes from the values of the g parameter and the equations (20) and (21). These equations show that a high value of g (as for the ethernet type) leads to smaller speedup. A similar conclusion holds on Convex SPP1000.

Another interesting thing to be observed is that the best execution time does not match with the preconditioner which gives the smallest number of iterations (which is obviously the preconditioner with more number of steps). Note also that for the same matrix size different preconditioners has been considered, since the number of blocks defining the preconditioner depends on the number of processors. This fact explains the different number of iterations for a given matrix size.

6 Conclusions

In this paper the complexity of some parallel preconditioners based on those presented in [4] and [5] has been analized using the BSP model. Moreover, a new way for constructing parallel preconditioners for the PCG method has been introduced, reducing the amount of the communication cost. Its convergence properties has been studied theoretically. An optimal realisation of these preconditioners for banded systems has been made, showing that for this type of systems good speedups can be obtained. From our experimental results it follows that the new preconditioners are competitive if the ratio between the computation and communication speed is smaller than some threshold (or, using the BSP model, if g is bigger than some quantity, which seems to be $O(10)$

flops/word). We conclude that, for parallel computers, the choice of the preconditioner depends on how fast is the communication network. For modern parallel arquitectures with small values of g, the original preconditioners seem to be the best choice. On the other hand, for slow communication networks with bigger values of g, as workstations networks, the modified preconditioners presented can lead to an improvement on the execution time.

Acknowledgements

The authors would like to thank to Centre de Computaciò i Comunicacions de Catalunya for the use of its IBM SP2 machine.

References

[1] L. Adams, m-step preconditioned conjugate gradient methods, *SIAM J. Sci. Stat. Comput.* **6(2)** (1985) 452–462.

[2] L. Adams and E. Ong, Additive polynomial preconditioners for parallel computers, *Parallel Comp.* **9** (1988) 333–345.

[3] R. H. Bisseling and W. F. McColl. Scientific computing on bulk synchronous parallel architectures. Technical Report 836, Department of Mathematics, University of Utrecht, December 1993.

[4] R. Bru, C. Corral, A. Martínez and J. Mas, Multisplitting preconditioners based on incomplete Choleski factorizations, *SIAM J. Mat. Anal.* **16(4)** (1995) 1210–1222.

[5] R. Bru, C. Corral and J. Mas, A preconditioned conjugate gradient method on a distributed memory multiprocessor, *Applied Mathematics Letters* **8(3)** (1995) 49–53.

[6] C. Corral, Precondicionadores paralelos para el método del gradiente conjugado, Ph.D. Thesis, Departament de Matemàtica Aplicada, Universitat Politècnica de València, April 1995.

[7] G.H. Golub and C.F. Van Loan, *Matrix Computations*, (The Johns Hopkins Press, Baltimore, Maryland, 1989).

[8] M. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.* **49** (1952) 409–436.

[9] J.A. Meijerink and H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Mathematics of Computation* **31(137)** (1977) 148-162.

[10] W.F. McColl, BSP Programming, in: G.E. Blelloch, K.M. Chandy and S. Jagannathan, eds., *Proceedings of the DIMACS Workshop* (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1994) **18** 21–35.

[11] W. F. McColl, Scalable Computing, in: J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments* (number 1000 in Lecture Notes in Computer Science, Springer Verlag, 1995) 46–61.

[12] R. Miller and J.L. Reed, The Oxford BSP library users' guide, Technical report, Programming Research Group, University of Oxford, 1993.

[13] D.P. O'Leary and R.E. White, Multisplittings of matrices and parallel solution of linear systems, *SIAM J. Alg. Discr. Met.* **6** (1985) 630-640.

[14] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems* (Plenum Press, New York, 1988).

[15] Y. Saad, *Iterative Methods for Sparse Linear Systems*, (PWS Publishing Company, Boston, 1996).

[16] L.G. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM* **33(8)** (1990) 103–111 .

N	p	m	IBM SP2						CONVEX SPP1000	
			Preconditioner						Preconditioner	
			Original			Modified			Original	Modified
			Number	time(sec)		Number	time(sec)		time	time
			Iterat.	HS	IP	Iterat.	HS	IP	(sec)	(sec)
6400	2	2	50	1.39	1.74	57	1.57	1.84	4.45	$\boxed{4.15}$
		3	42	1.53	1.89	50	1.76	2.03	4.68	4.20
		4	35	1.58	1.95	47	2.05	2.89	4.86	4.52
	4	2	50	0.85	2.14	58	0.93	$\boxed{1.63}$	7.38	7.13
		3	42	0.90	2.40	52	1.04	1.67	7.52	6.48
		4	35	0.92	2.80	47	1.13	1.81	7.66	6.34
	8	2	51	0.67	3.48	59	0.67	3.33	16.32	15.17
		3	43	$\boxed{0.64}$	3.81	52	0.68	2.89	15.52	14.2
		4	36	0.69	3.50	49	0.74	3.07	14.33	13.65
20736	2	2	78	6.82	7.75	88	7.65	8.69	14.37	13.83
		3	56	7.60	8.27	83	9.48	10.4	14.7	16.45
		4	35	7.93	8.54	76	10.67	11.51	15.1	17.86
	4	2	78	3.75	6.47	89	4.17	$\boxed{5.33}$	14.77	14.2
		3	56	4.10	6.56	77	4.66	5.71	15.23	13.98
		4	65	4.27	6.82	70	5.27	6.23	15.34	$\boxed{13.69}$
	8	2	79	$\boxed{2.28}$	7.84	91	2.60	6.99	27.3	27.95
		3	66	2.43	8.12	79	2.72	6.56	24	22.17
		4	56	2.50	8.14	72	2.96	6.46	24.07	25.04
40000	2	2	106	17.29	18.38	120	19.91	20.68	34.25	36.63
		3	90	19.64	20.73	104	22.5	23.24	37.56	40.14
		4	75	20.0	21.28	94	25.33	25.82	38.91	44.48
	4	2	107	9.76	13.13	122	10.76	$\boxed{12.5}$	29.96	29.9
		3	90	10.48	14.76	105	12.02	13.65	29.89	$\boxed{29.48}$
		4	75	10.74	14.28	94	13.14	14.80	30.27	30.45
	8	2	107	$\boxed{5.51}$	13.55	122	6.18	12.66	38.9	39.13
		3	90	6.01	14.82	106	6.99	14.05	37.89	35.11
		4	76	6.05	14.57	96	7.34	12.89	37.12	34.44
65536	2	2	135	38.82	37.66	153	41.0	41.94	74.63	81.7
		3	114	41.21	41.94	132	46.5	47.27	81.43	90.33
		4	95	41.98	43.33	118	51.5	52.23	84.11	97.49
	4	2	135	19.45	22.75	153	21.9	24.2	55.14	54.68
		3	114	21.65	29.91	132	24.5	27.61	55.7	56.29
		4	96	22.24	26.57	119	27.3	29.27	56.09	58.21
	8	2	136	$\boxed{11.10}$	22.29	155	12.9	30.17	58.39	57.98
		3	115	12.19	23.97	135	13.7	$\boxed{21.9}$	58.4	55.6
		4	96	12.22	24.41	122	15.1	22.63	56.58	$\boxed{53.96}$

Table 2
Observed time on a IBM SP2 and a CONVEX SPP1000. $\boxed{}$ indicates the best time for a given matrix size. N stands for the matrix size, p for the number of processors and m for the number of preconditioner steps.