Numer Algor (2013) 63:127–142
DOI 10.1007/s11075-012-9616-4

ORIGINAL PAPER

A parallel implementation of the CMRH method for dense linear systems

Sébastien Duminil

Received: 31 January 2012 / Accepted: 20 June 2012 /
Published online: 7 July 2012

Abstract This paper presents an implementation of the CMRH (Changing Minimal Residual method based on the Hessenberg process) iterative method suitable for parallel architectures. CMRH is an alternative to GMRES and QMR, the well-known Krylov methods for solving linear systems with non-symmetric coefficient matrices. CMRH generates a (non orthogonal) basis of the Krylov subspace through the Hessenberg process. On dense matrices, it requires less storage than GMRES. Parallel numerical experiments on a distributed memory computer with up to 16 processors are shown on some applications related to the solution of dense linear systems of equations. A comparison with the GMRES method is also provided on those test examples.

Keywords Linear systems · Krylov method · Hessenberg process · Dense matrix · Parallel implementation · MPI · CMRH · GMRES · Preconditioned CMRH

1 Introduction

Many large linear systems can be solved efficiently by iterative methods, especially those based on a Krylov subspace. In this paper, we are interested in the solution of large dense linear systems that appear in many engineering and scientific applications such as boundary element methods, quantum mechanical problems, large least squares problems, etc...

S. Duminil (✉)
Laboratoire de Mathématiques Pures et Appliquées, Université du Littoral,
Centre Universitaire de la Mi-Voix, Batiment H. Poincaré, 50 Rue F. Buisson,
BP 699, 62228 Calais cedex, France
e-mail: duminil@lmpa.univ-littoral.fr

Because they are quite time consuming, they require an efficient parallel implementation on powerful supercomputers. This is the main reason why we develop a parallel distributed dense CMRH method [6, 10] based on MPI [8]. MPI is a message passing library used on machines with distributed memory. We choose a Single Program Multiple Data (SPMD) programming style where each processor executes the same code on different data. Our code implements a CMRH algorithm with a row data distribution, using Level 1 and 2 BLAS routines.

The CMRH algorithm is described as an alternative method to the generalized minimal residual (GMRES) algorithm [9]. The GMRES method uses the Arnoldi process whence the CMRH method is based on the Hessenberg process. This method requires less work and storage than Arnoldi's process. The parallelization of the GMRES method has been studied by several authors [1–4]. In [6, 14] it was shown that CMRH is indeed a valid alternative to GMRES for certain situations. These include the cases where the matrix A is explicitly stored and is not very sparse, e.g, those arising from the discretization of boundary element methods or integral equations. Furthermore, it has been observed that in practice, CMRH behaves like GMRES (same behaviour of convergence). We have found no case in which CMRH suffers from instability.

Section 2 presents the mathematical formulation of CMRH together with the Hessenberg process. This presentation follows the previous work in [6, 10, 11]. A comparison is done with the GMRES method in terms of floating-point operations and memory requirements for dense matrices. Section 3 gives the parallel implementation with a SPMD programming style for distributed-memory architectures. The data distribution follows a 1D rowwise partitionning from the input matrix. The matrix-vector product and the parallel permutation of rows/columns are discussed. In Section 4, parallel and numerical performances are analyzed on some linear systems of equations. Conclusions are drawn in Section 5.

Throughout the paper, we define the Krylov subspace by

$$K_k(A, v) = \text{span}\left\{v, Av, \cdots, A^{k-1}v\right\}$$

for any given vector $v \in \mathbb{R}^n$ and an $n \times n$ non singular matrix A where k is the iteration number. For a vector v, $\|v\|$ refers to the Euclidean norm and $\langle x, y\rangle$ to the Euclidean inner product. $\|v\|_\infty$ denotes the maximum norm $\|v\|_\infty = \max_{i=1,\cdots,n} |(v)_i|$, where $(v)_i$ is the ith component of the vector v. If L is a matrix, we denote by l^i the ith column of L. By $e_1^{(k)}, \cdots, e_k^{(k)}$ we denote columns of the $k \times k$ identity matrix. When the length of these vectors is clear from the context, we drop the superscript.

We also use MATLAB-like notations $A_{i:j,k:l}$ to denote the submatrix of A consisting of rows i to j and columns k to l, $triu(A)$ and $tril(A)$ to denote the upper and lower, respectively, triangular part of the matrix A and $diag(v)$ to denote the square matrix of order n with the elements of v on the diagonal. The symbol $\leftrightarrow$ means swapping contents : $x \leftrightarrow y \Leftrightarrow t = x\ ;\ x = y\ ;\ y = t$.

2 Sequential version

2.1 The GMRES method

In this section, we give a short description of the GMRES algorithm. For a full description of this method and its standard implementation, see [9, 12]. Let A be a matrix, b and x_0 given vectors of size n.We consider the linear system of equations

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

The GMRES algorithm uses the Arnoldi process to construct an orthonormal basis $V_k = [v^1, \cdots, v^k]$ for $K_k(A, r_0)$ where $r_0 = b - Ax_0$ is the initial residual. The Arnoldi process begins with $v_1 = r_0/\|r_0\|$, and at step $j+1$, Av_j is orthogonalized with respect to$v_1, v_2, \cdots, v_j$,thus obtaining a vector in the direction of the new vector v_{j+1}. The GMRES algorithm is recalled in Algorithm 1.

Algorithm 1: GMRES method

```
Let x_0 and tol ;
Compute r_0 = b − Ax_0;
β = ‖r_0‖;
Arnoldi process :
 v^1 = r_0/β, ;
for k = 1, ··· , until convergence do
    u = Av^k ;
    for j = 1, ··· , k do
        h_{j,k} = v^{jT} u ;
        u = u − h_{j,k} v^j ;
    end
    h_{k+1,k} = ‖u‖;
    v^{k+1} = u/h_{k+1,k};
    Update the QR factorization of H̄_k ;
    Apply previous rotations to H̄_k and βe_1 ;
end
Solve H_k d_k = βe_1 ;
Update x_k = x_0 + V_k d_k ;
```

The matrix $\bar{H}_k$ is an upper Hessenberg matrix of order $(k+1) \times k$. We let $\beta = \|r_0\|$. GMRES algorithm computes

$$x = x_0 + V_k y_k$$

where y_k solves the least-squares problem

$$\min_{y \in \mathbb{R}^k} \|\beta e_1^{(k+1)} - \bar{H}_k y\|.$$

2.2 The CMRH method

In [10], Sadok proposed an algorithm to solve non symmetric linear systems: the CMRH method. This method uses the Hessenberg process to compute

at the kth step, a $n \times k$ unit trapezoidal matrix $L_k = \left[l^1, l^2, \cdots, l^k\right]$ whose columns form a basis of the Krylov subspace $K_k(A, r_0)$. The basis $\left\{l^1, l^2, \cdots, l^k\right\}$ is such that $(l^i)_i = 1$ and $(l^i)_j = 0,\ j = 1, \cdots, i-1$. In practice, the kth step in the Hessenberg process is as follows : Compute $u = Al^k$, then substract successively a multiple of l^1 to annihilate $(u)_1$, a multiple of l^2 to annihilate $(u)_2, \cdots$ and finally a multiple of l^k to annihilate $(u)_k$.

In [6], Heyouni and Sadok proposed an implementation which minimised the use of memory. At the $k+1$st iteration, the matrix-vector product with A is Al_k, and since the first $k-1$ entries of l_k are zeros, this product can be implemented without the need to access the first k columns of A. This fact also allows for an implementation in which the first k columns of the matrix A are used to store the basis vectors of L_k. We give the CMRH algorithm in Algorithm 2.

Algorithm 2: CMRH method

Let x_0 and tol be given;
Compute $b = b - Ax_0$;
*Hessenberg process: $p = [1, 2, \cdots, n]^T$; ;
Determine i_0 such that $|(b)_{i_0}| = \|b\|_\infty$;
$\alpha = (b)_{i_0},\ b = b/\alpha$, ;
$(p)_1 \leftrightarrow (p)_{i_0},\ (b)_1 \leftrightarrow (b)_{i_0}$;
$A_{1,:} \leftrightarrow A_{i_0,:},\ A_{:,1} \leftrightarrow A_{:,i_0}$;
for $k = 1, \cdots,$ *until convergence* **do**
  $u = A_{:,k} + A_{:,k+1:n}(b)_{k+1:n}$;
  $A_{k+1:n,k} = (b)_{k+1:n}$;
  for $j = 1, \cdots, k$ **do**
    $A_{j,k} = (u)_j,\ (u)_j = 0$;
    $(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k}A_{j+1:n,j}$;
  end
  Determine $i_0 \in \{k+1, \cdots, n\}$ such that $\left|(u)_{(p)_{i_0}}\right| = \left\|(u)_{(p)_{k+1}:(p)_n}\right\|_\infty$;
  $h = (u)_{(p)_{i_0}}$;
  $b = u/h$;
  $(p)_{k+1} \leftrightarrow (p)_{i_0},\ (b)_{k+1} \leftrightarrow (b)_{i_0}$;
  $A_{k+1,:} \leftrightarrow A_{i_0,:},\ A_{:,k+1} \leftrightarrow A_{:,i_0}$;
  Update the QR factorization of $\bar{H}_k$;
  Apply previous rotations to $\bar{H}_k$ and βe_1 ;
end
Solve $H_k d_k = \alpha e_1,\ (H_k = triu(A_{1:k,1:k}))$;
Update $x_k = x_0 + L_k d_k,\ (L_k = diag(ones(k,1)) + tril(A_{:,1:k}, -1))$; ;
Reorder the components of x_k ;
for $i = 1, \cdots, n$ **do**
  $(b)_{(p)_i} = (x_k)_i$;
end

Now we give some remarks: In Algorithm 2, computing $u = A_{:,k} + A_{:,k+1:n}$ $(b)_{k+1:n}$, requires $n(n-k)$ multiplications or additions . Furthermore, for each iteration k the computation of L_{k+1} requires $nk - k(k+1)/2$ multiplications or additions. The CMRH method requires for step k about $n^2 - k(k+1)/2$ multiplications or additions, while the GMRES algorithm requires $2nk + n^2$. Hence the CMRH method is less expensive than the GMRES method. On the

other hand, the CMRH method uses a permutation matrix, hence it involves some data movement. However this method is less expensive in terms of floating point operations per iteration and needs slightly less storage than the GMRES method per iteration. For more details, see [7, 10].

2.3 Convergence results

Now, we recall some results on convergence between the CMRH and the GMRES methods. For details on proofs, see [10, 11].

Let $\alpha = \|r_0\|_\infty$. The CMRH approximation at the kth step is

$$x_k = x_0 + L_k y_k$$

where y_k solves

$$\min_{y \in \mathbb{R}^k} \|\alpha e_1^{(k+1)} - \bar{H}_k y\|.$$

It was proved that the CMRH approximation minimizes a quasiresidual

$$\min_{z \in K_k} \|L_{k+1}^+ (Az - r_0)\|.$$

Theorem 1 *Let r_k^C and r_k^G be the CMRH and GMRES residuals at the kth iteration beginning with the same initial residual r_0, respectively. Then*

$$\|r_k^C\| \leq \kappa(L_{k+1}) \|r_k^G\|$$

where $\kappa(L_{k+1}) = \|L_{k+1}\| \|L_{k+1}^+\|$ is the condition number of L_{k+1}.

If we consider the QR factorization of L_k:

$$L_k = V_k R_k, \tag{2}$$

we obtain an other results of convergence.

Theorem 2 *Let r_k^C and r_k^G be the CMRH and GMRES residuals at the kth iteration beginning with the same initial residual r_0, respectively. Then*

$$\|r_k^G\| \leq \|r_k^C\| \leq \kappa(R_{k+1}) \|r_k^G\|.$$

2.4 Preconditioned CMRH method

As for other iterative methods, it is usually to combine the CMRH method with an efficient preconditioning technique. Unlike the GMRES method, we can only use the left-preconditioned algorithm to keep the structure of the vector u for the matrix-vector product. In this section, we give an implementation of the left-preconditioned CMRH method.

Let M be a given nonsingular $n \times n$ matrix which approximates in some sense the coefficient matrix A, such that systems $Mx = z$ can easily be solved.

Instead of solving the original system (1), we apply the CMRH algorithm to the equivalent linear system

$$M^{-1}Ax = M^{-1}b\,.$$

The Hessenberg process constructs a basis of the left-preconditioned Krylov subspace

$$K_k^m = K_k(M^{-1}A, r_0) = \operatorname{span}\left\{r_0, M^{-1}Ar_0, \cdots, (M^{-1}A)^{k-1}r_0\right\}.$$

All residual vectors and their norms that are computed by the algorithm correspond to the preconditioned residuals, namely, $z_k = M^{-1}(b - Ax_k)$, instead of the original (unpreconditioned) residuals $b - Ax_k$.

The CMRH algorithm 2 combined with preconditioning can be sketched as in Algorithm 3.

Algorithm 3: Preconditioned CMRH method

Let x_0 and tol be given;
Compute $r_0 = b - Ax_0$;
Solve $Mb = r_0$ *Hessenberg process: $p = [1, 2, \cdots, n]^T$; ;
Determine i_0 such that $|(b)_{i_0}| = \|b\|_\infty$;
$\alpha = (b)_{i_0},\ b = b/\alpha,$;
$(p)_1 \leftrightarrow (p)_{i_0},\ (b)_1 \leftrightarrow (b)_{i_0}$;
$A_{1,:} \leftrightarrow A_{i_0,:},\ A_{:,1} \leftrightarrow A_{:,i_0}$;
for $k = 1, \cdots,$ *until convergence* **do**
  $v = A_{:,k} + A_{:,k+1:n}(b)_{k+1:n}$;
  Solve $Mu = v$;
  $A_{k+1:n,k} = (b)_{k+1:n}$;
  for $j = 1, \cdots, k$ **do**
    $A_{j,k} = (u)_j,\ (u)_j = 0$;
    $(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k}A_{j+1:n,j}$;
  end
  Determine $i_0 \in \{k+1, \cdots, n\}$ such that $\left|(u)_{(p)_{i_0}}\right| = \left\|(u)_{(p)_{k+1}:(p)_n}\right\|_\infty$;
  $h = (u)_{(p)_{i_0}}$;
  $b = u/h$;
  $(p)_{k+1} \leftrightarrow (p)_{i_0},\ (b)_{k+1} \leftrightarrow (b)_{i_0}$;
  $A_{k+1,:} \leftrightarrow A_{i_0,:},\ A_{:,k+1} \leftrightarrow A_{:,i_0}$;
  Update the QR factorization of $\bar{H}_k$;
  Apply previous rotations to $\bar{H}_k$ and βe_1 ;
end
Solve $H_k d_k = \alpha e_1,\ (H_k = triu(A_{1:k,1:k}))$;
Update $x_k = x_0 + L_k d_k,\ (L_k = diag(ones(k,1)) + tril(A_{:,1:k}, -1))$;
Reorder the components of x_k ;
for $i = 1, \cdots, n$ **do**
  $(b)_{(p)_i} = (x_k)_i$;
end

For the numerical examples in Section 4, we have used the incomplete Cholesky preconditioning. This preconditioner is based on a decomposition of a matrix E into the product of a sparse lower triangular matrix C and its conjugate transpose such that $E \approx C^T C$. For our experiments, we have used the incomplete Cholesky preconditioning on the matrix $A^T A$.

3 Parallel implementation of the CMRH method

In this section, we give a parallel implementation of the CMRH method. We want to analyse the cost of one iteration of CMRH with a specific data distribution. We also propose a comparison with the corresponding cost associated with GMRES. The computation of the basis L_k contains three main steps: matrix-vector products, the factorization and swaps in the matrix. We now analyse the parallelization of these three procedures. We begin by the data distribution.

3.1 Data distribution

Algorithm 2 only involves the original matrix A and three vectors b, p and u to compute the CMRH iterates. A parallelization of the algorithm on distributed memory processors requires the distribution of the matrix A. We do not distribute the vector p on each processor. We store it on processor 0. We only need to swap two scalars per iteration.

Assume that the matrix A is distributed among Np processors. We deal with the natural case in which each processor holds a set of rows of the matrix. Let $nl = \frac{n}{Np}$ be the number of rows in each processor. We obtain Np matrices of size $nl \times n$.

The most communication costs metric addressed in this parallelization problems are the matrix-vector product and the swap of rows. We will see the advantages to use this distribution to obtain a good performance. Indeed, we will see that this distribution allows to have the same number of operations in each processor for the matrix-vector product. Then we will see that this distribution allows a permutation of the columns of the matrix A without any data transfer, only the permutation of the rows will involve a transfer of data. Otherwise, others communications are negligible. We must also do collective communications for some real numbers.

3.2 Parallel matrix-vector product

In this section, the distributed matrix-vector product is studied. With our data distribution, the matrix-vector product uses the same number of computations on each processor. Each processor computes nl rows of the vector u.

If we would have a column distribution for the matrix A, the number of processors used to compute u will decrease and the number of computations per processor will increase. For a cyclic block distribution of A, the number of computations per processor will be unequal. With our distribution, all processors are kept busy to compute u. At iteration k, the matrix-vector product requires $(n - k) \times nl$ multiplications on each processor. For the GMRES method, this step requires $n \times nl$ multiplications on each processor. Since the vector b is updated in parallel in a previous step, it induces communication to compute u. A straightforward implementation would broadcast the vector b to each processor so that each processor gets a complete copy.

Table 1 Summarize of maximum costs and maximum transfers at iteration k

	CMRH		GMRES	
	Operations	Transfer	Operations	Transfer
Matrix-vector product	$2nl \times (n-k)$		$2nln$	
Dot product			$2k \times nl + Np$	$k \times Np$
$u = u - h_{j,k} v_j$	$2k \times nl$	k	$2k \times nl$	
$\|u\|_2$ or $\|u\|_\infty$	nl	Np	$2nl + Np + 1$	Np
Swap column	$nl <->$			
Swap row	$n <->$	n		

3.3 Parallel factorization

The vector u obtained is a local vector of size nl distributed on each processor. This local distribution of u allows the computation of the next part of the Algorithm 2 without any vector transfer:

$$(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k} A_{j+1:n,\,j}$$

We use a daxpy subroutine in the BLAS library. We must only transfer $A_{j,k}$ using a global communication.

For this step, the GMRES method computes k scalar products. Global communications are still required for these products. To compute the norm, euclidean norm for GMRES and infinity norm for CMRH, both methods requires same communications. Finally, costs for Givens rotations are similar for both methods.

3.4 Parallel swap

This step only relates to the CMRH method. At each iteration, Algorithm 2 requires two swaps of two vectors: One swap of rows and one swap of columns of the matrix A. Our distribution of the matrix A allows to decrease the time of the second swap. This swap takes place on each processor without data transfer. We use the swap subroutine of BLAS library applicated on a vector of size nl.

Instead, the parallel swap of rows performs the step with the biggest number of real which must be transferred. If the swap takes place in two different processor, we must transfer two rows of the matrix A.

Table 1 gives a summarize of all costs and transfers for one iteration of the CMRH and the GMRES methods.

4 Numerical experiments

We have implemented a parallel code for the proposed matrix scaling algorithm in Fortran 90 using MPI [8] on a Bull Novascale 5160 with 16 processors Itanium II 1,5 GHz and 64 GB of RAM. In this section, we compare the CMRH method and the GMRES method on some examples. We compare the

residual and the error norms, the time of each process and the maximum size of matrices. Then, we give times for different values of n and different numbers of processors Np. For the tests, we use different matrices. We now define all matrices.

$$B_{i,j} = \begin{cases} -\log|z_i - z_j|, & i \neq j, \\ -\log|r_i| \end{cases} \tag{3}$$

where z_i are n somehow randomly distributed points in a unit square centered at the origin in the complex plane and where each r_i is a number in $(0, d_i]$, d_i being the distance between the point z_i and its nearest neighbour. We observe that computing an off-diagonal entry B_{ij} corresponds, up to a factor of $-\frac{1}{2\pi}$, to evaluating the free space Green's function for the Laplacian in two dimensions with argument $z_i - z_j$. For more details, see [5]. We use one matrix coming either from University of Florida Matrix collection [13]: TSOPF. It is of order $n = 38120$ with $nz = 16171169$ nonzeros entries. We treated this matrix as dense and nonsymmetric matrix. The last matrix coming from the integral formulation of the Helmholtz equation. We denote C this matrix. C is defined as follows

$$C = I + \kappa^2 K$$

where κ represent the frequency of the wave oscillations and the entries of the K matrix as $K_{ij} = h^2 g(x_i - x_j)$. h is the step size of the discretized domain and g is the freespace Green's function in $\mathbb{R}^2$. The matrix K (and therefore A) is a dense non-Hermitian square matrix. We choose $n = 20{,}736$.

For all examples, the right hand side is chosen such that the exact solution is $x^* = 1$ and we choose $x_0 = 0$. The iteration was stopped as soon as the convergence criterion

$$\|r_k\|/\|r_0\| < 10^{-10}$$

was satisfied.

In figures, the solid line is used for the CMRH method while the dashed line represents the GMRES curve.

4.1 Convergence analysis

We compare the convergence of the two methods on all matrices. Figures 1, 2 and 3 show the convergence behaviour of the residual norm using a logarithmic scale. As the plots indicate, the curves for CMRH and GMRES are very close to each other. For the matrix B, the GMRES method converges after 8851 iterations whereas the CMRH method converges in 8,868 iterations. For the matrix TSOPF, the CMRH method converges in 5,029 iterations and the GMRES method converges in 4,985 iterations. Finally, for the last one, the GMRES method converges in 3,871 whereas the CMRH method converges in 3,924 iterations.

Next, we ran the matrix B with the Cholesky preconditioning (cf. Section 2.4). For this example, I choose $n = 20000$. Figure 4 shows the convergence curves

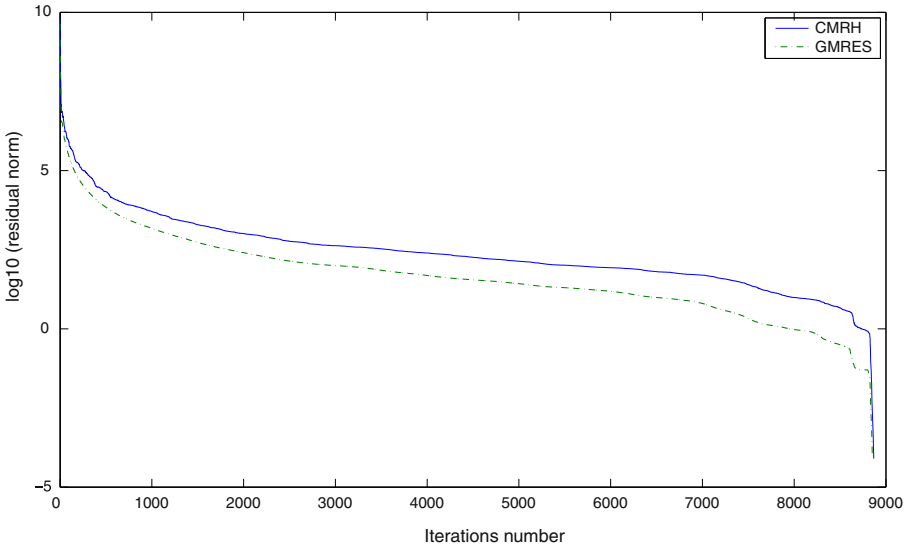

Fig. 1 Convergence Residual Norm for matrix B

of the relative residual norm $\|r_k\|/\|r_0\|$ for CMRH and preconditioned CMRH. The solid line is always used for the CMRH method while the dash-dotted line represents the preconditioned CMRH method curve. Clearly, the convergence of the preconditioned method is better than no preconditioning method.

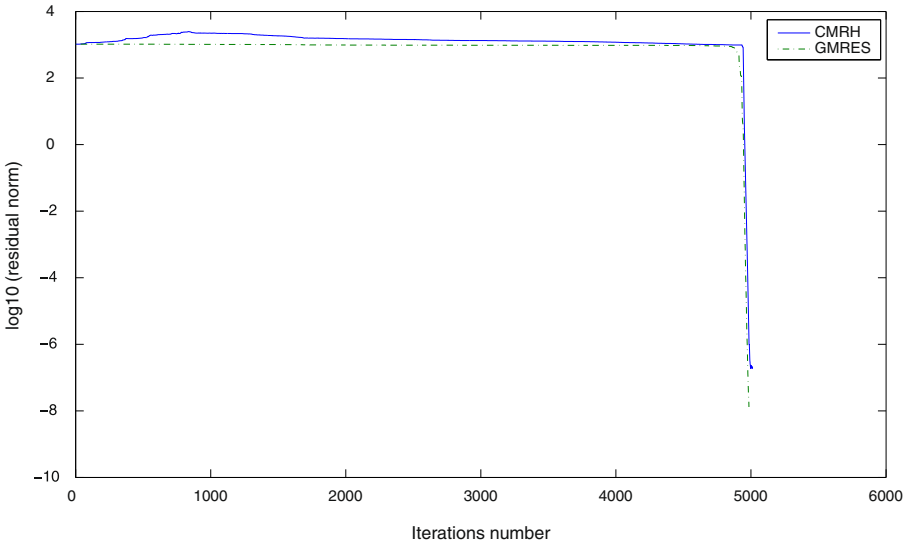

Fig. 2 Convergence Residual Norm for matrix TSOPF

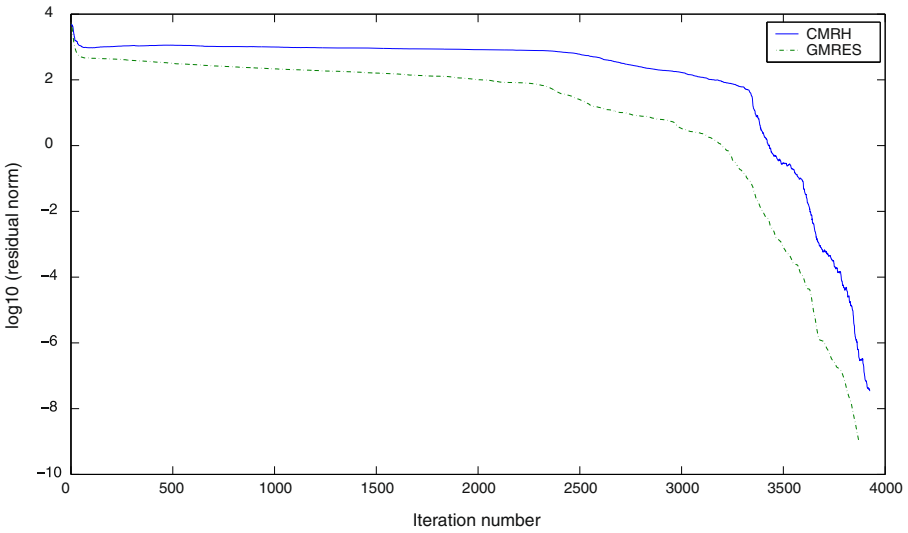

Fig. 3 Convergence Residual Norm for matrix C

4.2 Comparison of computing times

Now we compare the CPU time of both methods when we use the parallel implementation. The computational experiments presented here were executed on 16 processors. Results are presented for different values of n and have been computed using the matrix B.

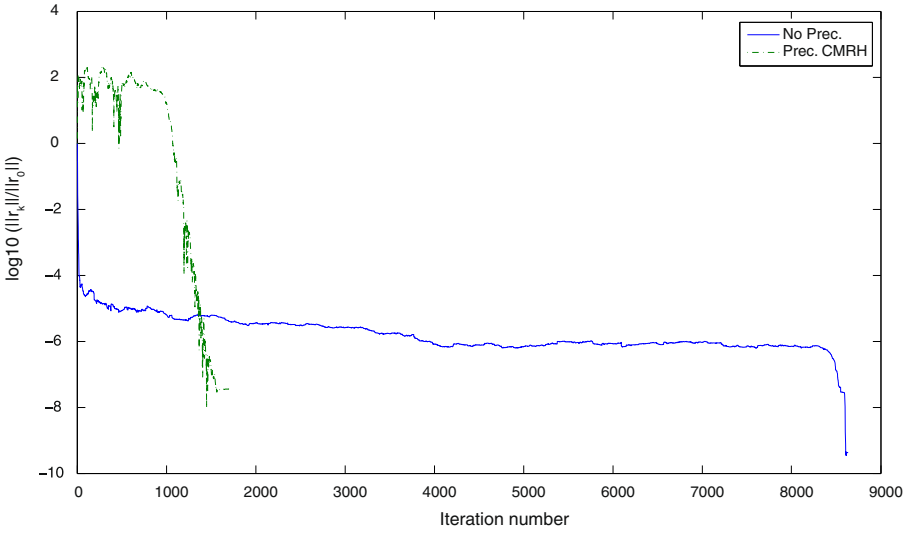

Fig. 4 Convergence curve for matrix B with preconditioned CMRH

Table 2 CPU time (s) of CMRH method and GMRES method with different values of n

n	Iterations	CMRH	GMRES
16,000	6,688	941	1,525
32,000	12,846	6,713	8,229
48,000	18,181	20,873	28,217

Table 2 gives the time for different sizes of the matrix B. We consider the same iteration number for both methods. This table shows that the CMRH method is faster than the GMRES method for this example. Note that we must compute a scalar product at each iteration in the GMRES method. This scalar product involves nl multiplications per processor and a data transfer with a global reduction. It is not the case for the CMRH method.

Figures 5, 6 and 7 give the CPU time in seconds for a varying number of processors for matrices B, TSOPF and C. For the matrix B, we choose $n =$ 32,000. The CPU time is comparable with 1 processor. Then, we see that the CMRH method is faster than GMRES method which confirm precedent observations. For the second matrix, curves have the same behaviour. The cost of swaps is comparable as the cost of dot products. Finally, for the third matrix, the CMRH method is clearly faster than the GMRES method.

4.3 Comparison of the maximum size

In Section 2, we said that the CMRH method minimized the use of memory in comparison with the GMRES method. We give the maximum feasible size of matrix with 64 GB of RAM for the CMRH method. We always choose the matrix B.

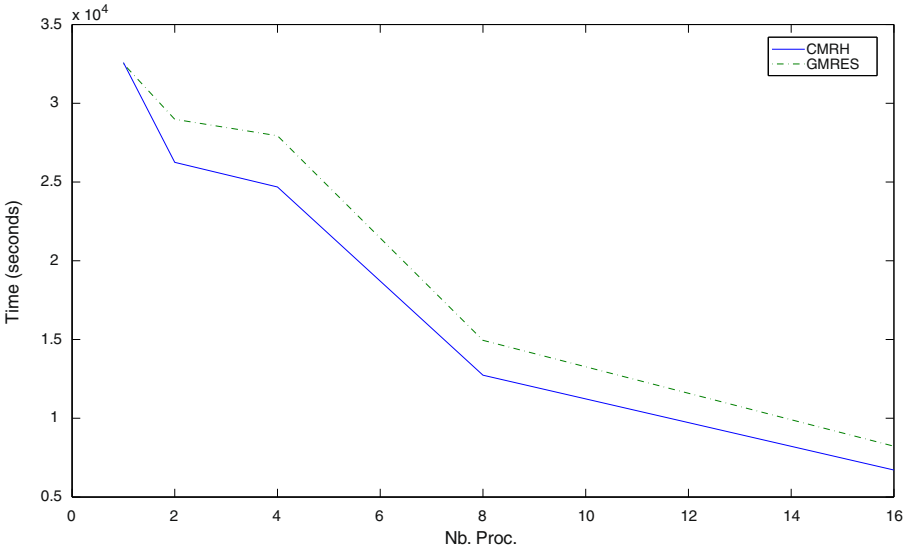

Fig. 5 CPU Time (seconds) matrix B ($n = 32{,}000$)

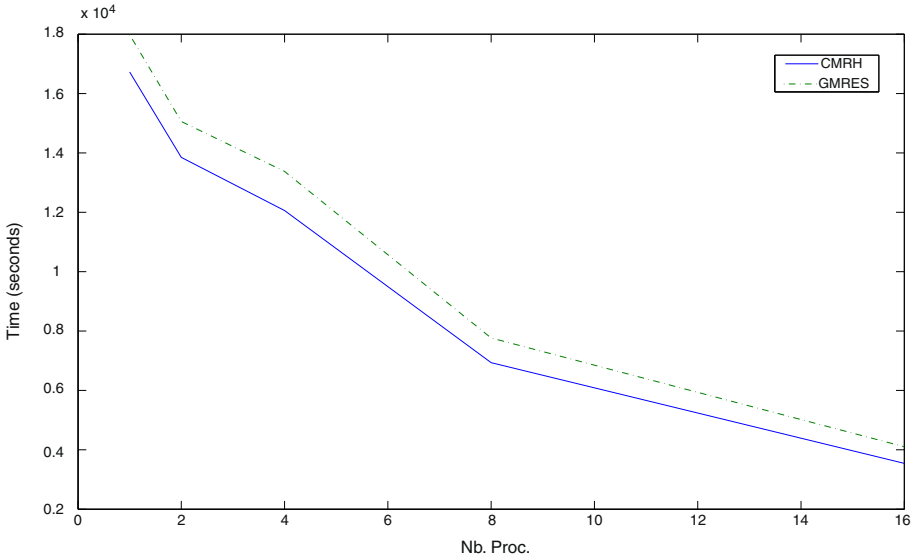

Fig. 6 CPU Time matrix TSOPF

We test that with 64 GB of RAM, we can use our code with $n = 89{,}600$. It requires 31,559 steps. We recall that the storage of matrix L_k et $\bar{H}_k$ in the original matrix B.

We recall that the major drawback of GMRES is that the amount of storage required per iteration rises with the iteration number.

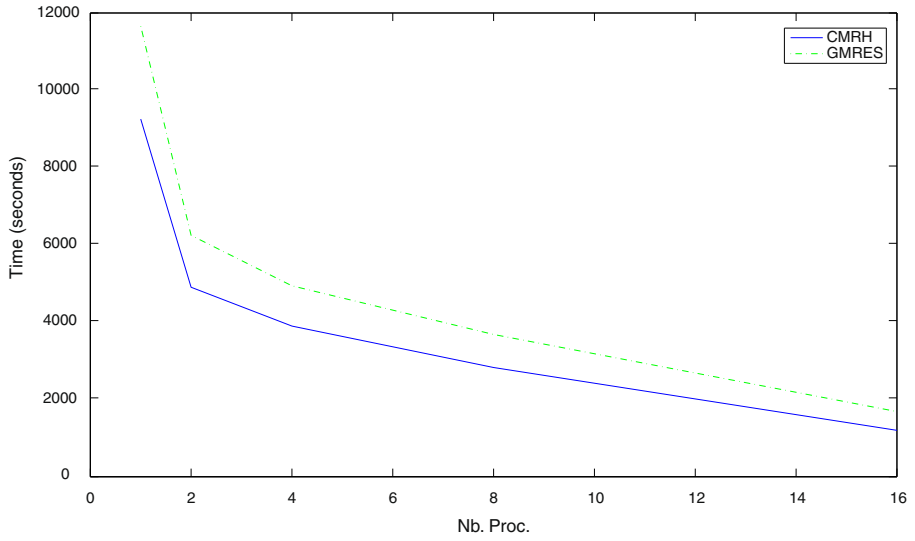

Fig. 7 CPU Time matrix C

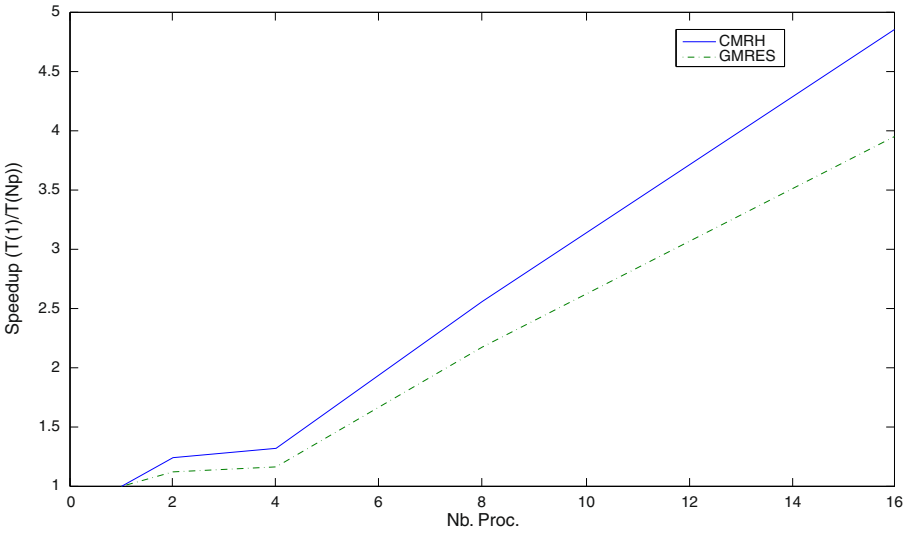

Fig. 8 Speedup matrix B

We must create two extra matrices of size $n \times nmax$ for the GMRES method.

4.4 Ratio between computing times for the CMRH code

Here, time gains are presented for different values of the number of processors. We studied the effect of the parallel implementation on our three matrices: B,

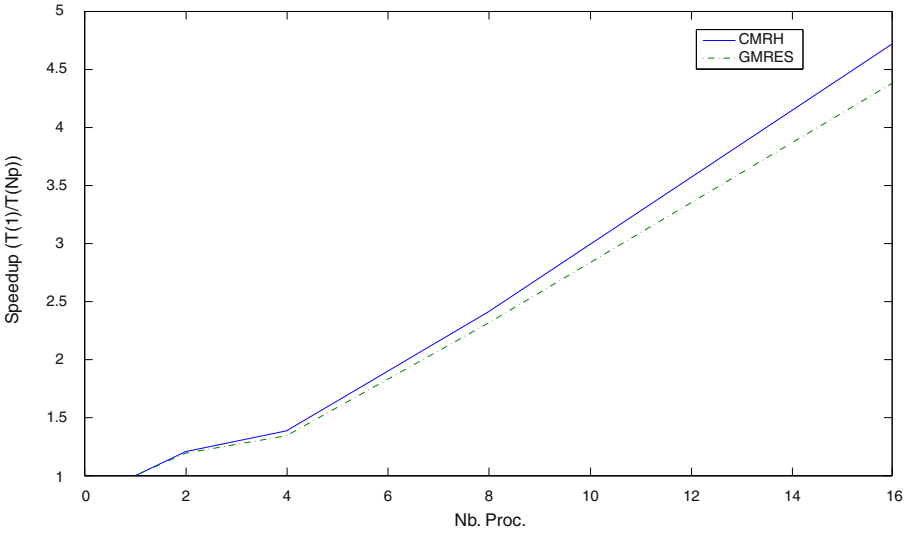

Fig. 9 Speedup matrix TSOPF

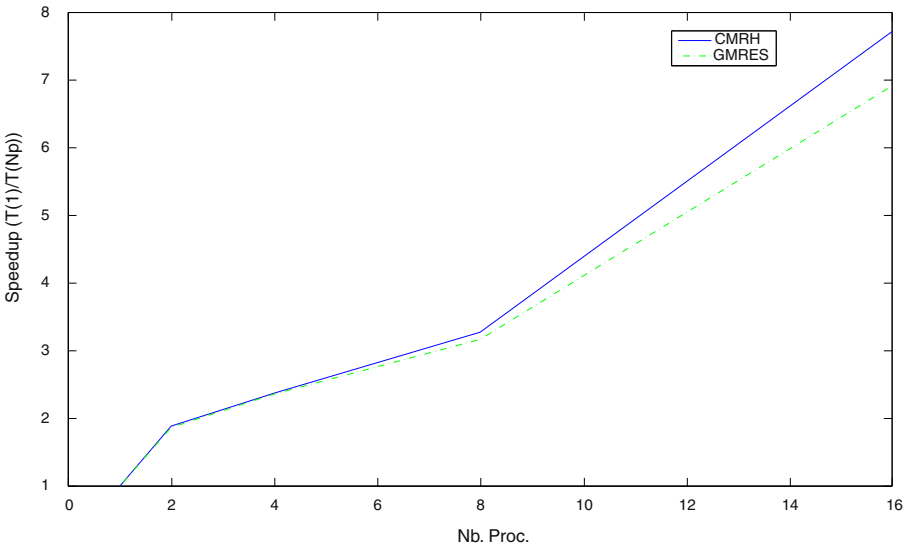

Fig. 10 Speedup matrix C

TSOPF and C. Results are given in Figs. 8, 9 and 10. These results confirm that the CMRH method is more scalable than GMRES method when the processor number grow up.

5 Conclusion

We recalled an iterative method for solving dense non-symmetric linear systems: the CMRH method, and briefly mentioned the algorithm. We proposed a left-preconditioned algorithm of the CMRH method and presented an example using the Cholesky preconditioning. We discussed the parallelization of the no preconditioned algorithm. We proposed a classic data distribution to decrease the number of data transfers and to decrease the number of operations on each processor. We compared our algorithm with a parallel GMRES algorithm. We used the same data distribution than for the CMRH method. Results on our method show good performances. The convergence of the residual norm is comparable to the GMRES method.If we compare the CPU time, the CMRH method is 20 to 30 percent faster than the GMRES method. The use of our method on computers with more processors and larger memories would probably to confirm our results.

Acknowledgements I would like to thank Prof. Hassane Sadok (my advisor, Université du Littoral, Calais) for this help with the CMRH method and Philippe Marion (Université du Littoral, Calais) for his help in using the parallel computer.

References

1. Bai, Z., Hu, D., Reichel, L.: A Newton basis GMRES implementation. IMA J. Numer. Anal. **14**, 563–581 (1994)
2. Bai, Z., Hu, D., Reichel, L.: Implementation of GMRES method using QR factorisation. In: Proc. Fifth SIAM Conference on Parallel Processing for Scientific Computing, pp. 84–91 (1992)
3. Di Brozolo, J.J., Robert, Y.: Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor. Parallel Comput. **11**, 84–91 (1989)
4. Erhel, J.: A parallel GMRES version for general sparse matrices. Electron. T. Numer. Anal. **3**, 160–176 (1995)
5. Helsing, J.: Approximate inverse preconditioners for some large dense random electrostatic interaction matrices. BIT Numer. Math. **46**, 307–323 (2006)
6. Heyouni, M., Sadok, H.: A new implementation of the CMRH method for solving dense linear systems. J. Comput. Appl. Math. **213**, 387–399 (2008)
7. Hessenberg, K.: Behandlung der linearen Eigenwert-Aufgaben mit Hilfe der Hamilton-Cayleychen Gleichung. Darmstadt dissertation (1940)
8. Message Passing Interface Forum: MPI: a message-passing interface standard. Int. J. Supercomputing Applications and High Performance Computing (1994)
9. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Statist. Comput. **7**, 856–869 (1986)
10. Sadok, H.: CMRH: A new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm. Numer. Algorithms **20**, 303–321 (1999)
11. Sadok, H., Szyld, D.B.: A new look at CMRH and its relation to GMRES. BIT Numer. Math. **52**, 485–501 (2012)
12. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. PWS Publishing, Boston (1996). SIAM, Philadelphia (2003)
13. Davis, T.A., Hu, Y.: The University of Florida Sparse Matrix Collection. ACM T. Math. Software **38**, 1–25 (2011)
14. Alia, A., Sadok, H., Souli, M.: CMRH method as iterative solver for boundary element acoustic systems. Eng. Anal. Bound. Elem. **36**, 346–350 (2012)