


Experimental Study of a Parallel Iterative Solver for Markov Chain Modeling^{*}

V. Besozzi¹, M. Della Bartola¹, and L. Gemignani¹^[0000-0001-8000-4906] 

Dipartimento di Informatica, Università di Pisa, 56127, Pisa, Italy

 Corresponding author: luca.gemignani@unipi.it

Abstract. This paper presents the results of a preliminary experimental investigation of the performance of a stationary iterative method based on a block staircase splitting for solving singular systems of linear equations arising in Markov chain modelling. From the experiments presented, we can deduce that the method is well suited for solving block banded or more generally localized systems in a parallel computing environment. The parallel implementation has been benchmarked using several Markovian models.

Keywords: Iterative methods · parallel algorithms · Markov chains.

1 Introduction

The solving of linear algebraic systems lies at the core of many scientific and engineering simulations. Discrete-state models are widely employed for modeling and analysis of large networks and systems such as communication networks, allocation schemes, computer systems and population processes. If the future evolution of the system depends only on the current state of the system and not on the past history, the system may be represented by a Markov chain. For a homogeneous, irreducible, continuous time Markov chain with N states, the long-term behavior of the system is determined by the stationary probability vector $\boldsymbol{\pi} \in \mathbb{R}^N$ such that

$$Q^T \boldsymbol{\pi} = \mathbf{0}, \quad \boldsymbol{\pi} \geq \mathbf{0}, \quad \mathbf{e}^T \boldsymbol{\pi} = 1, \quad (1)$$

where $Q \in \mathbb{R}^{N \times N}$ is the transition rate matrix, or the infinitesimal generator of the Markov chain, and $\mathbf{e} = [1, \dots, 1]^T$. Since Q is irreducible, by the Perron-Frobenius Theorem [17] we find that Q has rank $N - 1$ and, therefore, $\boldsymbol{\pi}$ spans the kernel of Q^T . The computation of $\boldsymbol{\pi}$ amounts to solve the homogeneous

^{*} This work has been supported by the project PRA_2020_61 of the University of Pisa and by the Spoke 1 “FutureHPC & BigData” of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di ”campioni nazionali di R&S (M4C2-19)” - Next Generation EU (NGEU).

linear system (1). A review of numerical methods for solving (1) can be found in [26, 20]. Active research in this area is focused on the development of techniques, methods and data structures, which minimize the computational (space and time) requirements for solving the linear system (1) when Q is large and sparse. One of such techniques is parallelization.

Iterative methods are generally preferred for solving large linear systems of equations because they are insensitive to fill-in and accuracy issues [20]. Stationary iterative methods like Gauss-Seidel (GS), Jacobi, and Successive Over-Relaxation (SOR) are interesting on their own and have further applications as preconditioners for projection methods like CG and GMRES. Experimental studies demonstrated that for Markov chain problems (1) block methods based on matrix splittings such as block Jacobi and block Gauss-Seidel give better convergence than other projection methods (see [27] and the references given therein).

Among classical iterative methods, the Gauss-Seidel method has several interesting features. It is a classical result that on a nonsingular M-matrix the Gauss-Seidel method converges faster than the Jacobi method [5, Corollary 5.22]. Moreover it can be implemented just using one iteration vector which is an important feature for huge systems. The SOR method with the optimal relaxation parameter can be better yet, but, however, choosing an optimal SOR relaxation parameter is difficult for many problems. Therefore, the Gauss-Seidel method is very attractive in practice and it is also used as preconditioner in combination with other iterative schemes. A classical example is the multigrid method for partial differential equations, where using Gauss-Seidel or SOR as a smoother typically yields good convergence properties [28].

Parallel implementations of Gauss-Seidel method have been designed for certain regular problems, for example, the solution of Laplace's equations by finite differences, by relying upon red-black coloring or more generally multi-coloring schemes to provide some parallelism [19]. In most cases, constructing efficient parallel true Gauss-Seidel algorithms is challenging and Processor-Block (or localized) Gauss-Seidel is often used [25]. Recent examples with applications to Markov chain modeling are the methods proposed in [6] and [1]. Here, each processor performs Gauss-Seidel as a subdomain solver for a block Jacobi method. While Processor-Block Gauss-Seidel methods are easy to parallelize, the overall convergence of the resulting iterative scheme can suffer.

In order to cope with the parallelization of Gauss-Seidel type methods while retaining the same convergence rate, in [14] staircase splittings were introduced by proving that, for consistently ordered matrices [21], the iterative scheme based on such partitionings splits into independent computations and at the same time exhibits the same convergence rate as the classical Gauss-Seidel iteration. A specialization of this result for block tridiagonal matrices had already appeared in [2]. More recently, in [10] the computational interest of staircase splittings has been broadened by showing that for a nonsingular M-matrix A in block lower Hessenberg form the asymptotic rate of convergence of the block staircase method is better than the asymptotic rate of convergence of the block Gauss-

Seidel method applied to A . A further extension with applications to accelerating certain fixed point iterations for Markov chain modeling is given in [9]. These results are quite surprising since the matrix M in the block staircase partitioning of $A = M - N$ is much more sparse than the corresponding block lower triangular matrix M of the Gauss-Seidel splitting.

The contribution of this paper is twofold. The matrix $A = Q$ in (1) is singular and the comparison theorems proved in [10] do not extend to the singular case, while classical results for singular systems [15, 16] do not apply to our methods. The first aim is to gain an understanding of how (block) staircase and (block) Gauss-Seidel type methods compare when applied for solving large and sparse Markov Chain problems. In particular, we are interested in the case where A is banded or localized around the main diagonals. The second goal is to perform this comparison in a parallel computing environment. To do this we have implemented a block staircase iterative solver for the parallel computation of the vector π . The properties of this method are examined experimentally. In particular, numerical experiments are performed to compare our method with an implementation of the composite solver proposed in [6] in terms of traditional efficiency measures for parallel algorithms. Our experimental evidence indicates that the block staircase partitioning generally works quite well when compared to block Gauss-Seidel for block banded or more generally localized matrices [4] with entries decaying away from the main diagonals. A discussion of the results is presented together with some conclusions and insights for future work.

2 Mathematical Background

Let $P = (p_{i,j}) \in \mathbb{R}^{N \times N}$ be a transition probability matrix of a homogeneous ergodic Markov Chain with N states. Then P is irreducible and row-stochastic, that is, $p_{i,j} \geq 0$, $1 \leq i, j \leq N$, and $Pe = e$ with $e^T = [1, \dots, 1]$. The matrix $Q^T = I_N - P^T$ is a singular M-matrix. Observe that $e^T Q^T = \mathbf{0}^T$. Since Q^T is also irreducible, by the Perron-Frobenius Theorem it follows that the kernel of Q^T is spanned by a vector π such that $\pi > \mathbf{0}$ and $e^T \pi = 1$. This vector is called the stationary probability distribution vector of the Markov Chain.

The computation of π amounts to solve the homogeneous linear system $Q^T \pi = \mathbf{0}$ under the normalization $e^T \pi = 1$. Iterative methods based on the power iteration can be used [20]. The computational efficiency and the convergence properties of these algorithms can benefit of a block partitioning of the matrix Q^T . Let us assume that

$$Q^T = \begin{bmatrix} Q_{1,1} & \cdots & Q_{1,n} \\ \vdots & \vdots & \vdots \\ Q_{n,1} & \cdots & Q_{n,n} \end{bmatrix},$$

where $Q_{i,j} \in \mathbb{R}^{n_i \times n_j}$, $1 \leq i, j \leq n$, $\sum_{i=1}^n n_i = N$. A regular splitting of the matrix Q^T is a partitioning $Q^T = M - N$ with $M^{-1} \geq 0$ and $N \geq 0$. Since

$Q^T \pi = \mathbf{0}$ we find that $M\pi = N\pi$ which gives $M^{-1}N\pi = \pi$. It is well known that the spectral radius of $M^{-1}N$ is equal to 1 and $\lambda = 1$ is a simple eigenvalue of $M^{-1}N$ [24]. This does not immediately imply that $\lambda = 1$ is the dominant eigenvalue of $M^{-1}N$, that is, that for the remaining eigenvalues λ of $M^{-1}N$ it holds $|\lambda| < 1$.

Example 1. Let $Q^T = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1/2 & 1 & -1/2 & 0 \\ 0 & -1/2 & 1 & -1/2 \\ 0 & 0 & -1 & 1 \end{bmatrix}$. The Jacobi splitting with

$M = I_4$ is a regular splitting but the iteration matrix $M^{-1}N$ has eigenvalues $\{-1, 1, -1/2, 1/2\}$. The Gauss-Seidel splitting is a regular splitting and the corresponding iteration matrix has eigenvalues $\{1, 1/4, 0, 0\}$.

By graph-theoretic arguments [24] it follows that for a regular splitting the matrix $M^{-1}N$ is permutationally similar to a block matrix $T = \begin{bmatrix} 0 & T_{1,2} \\ 0 & T_{2,2} \end{bmatrix}$ where $T_{2,2}$ is square, irreducible and non-negative and every row of the possibly nonempty matrix $T_{1,2}$ is nonzero. A non-negative square matrix A is primitive if there is $m \geq 1$ such that $A^m > 0$. By the Perron-Frobenius Theorem we obtain that $\lambda = 1$ is the dominant eigenvalue of $M^{-1}N$ if $T_{2,2}$ is primitive. Hereafter, this condition is always assumed. Under this assumption the classical power iteration is eligible for determining a numerical approximation of the vector π .

The method based on the (block) Jacobi splitting is very convenient to vectorize and to parallelize. As shown in the simple example above it can suffer from convergence problems. In this respect, the (block) Gauss-Seidel iteration generally outperforms the Jacobi algorithm. Processor-Block (or localized) Gauss-Seidel schemes provide a reliable compromise between parallelization and convergence issues. One such hybrid adaptation is described in [6]. Suppose that the matrix Q^T is partitioned as

$$Q^T = \begin{bmatrix} Q^{(1)T} \\ \vdots \\ Q^{(p)T} \end{bmatrix}, \quad Q^{(j)T} = \begin{bmatrix} Q_{m_j,1} & \cdots & Q_{m_j,n} \\ \vdots & \vdots & \vdots \\ Q_{m_j+r_j-1,1} & \cdots & Q_{m_j+r_j-1,n} \end{bmatrix},$$

with $m_1 = 1$, $m_j = \sum_{i=1}^{j-1} r_i + 1$, $2 \leq j \leq p$, $\sum_i^p r_i = n$. The iterative scheme in [6] exploits the regular splitting where

$$M = \text{diag} [M_1, \dots, M_p], \quad M_j = \begin{bmatrix} Q_{m_j,m_j} & & & \\ Q_{m_j+1,m_j} & Q_{m_j+1,m_j+1} & & \\ \vdots & & \ddots & \\ Q_{m_{j+1}-1,m_j} & \cdots & & Q_{m_{j+1}-1,m_{j+1}-1} \end{bmatrix}$$

and, hence, M is a block diagonal matrix with block lower triangular blocks. The resulting scheme proceeds as follows:

$$\begin{cases} M\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} \\ \mathbf{x}^{(k+1)} = \frac{\mathbf{x}^{(k+1)}}{e^T \mathbf{x}^{(k+1)}} \end{cases}, \quad k \geq 1. \quad (2)$$

1. The solution of a linear system $M\mathbf{x} = \mathbf{b}$ can be carried out in two parallel steps since all even and all odd components of \mathbf{x} can be computed concurrently.
2. In terms of convergence these splittings inherit some advantages of the block Gauss-Seidel method. If Q^T is block tridiagonal, then it can be easily proved that the iteration matrices associated with block Gauss-Seidel and block staircase splittings have the same eigenvalues and therefore the same convergence rate. In [10] it is shown that the spectral radius of the iteration matrix generated by the staircase splitting of an invertible M-matrix in block lower Hessenberg form is not greater than the spectral radius of the corresponding iteration matrix in the block Gauss-Seidel method. For singular matrices the convergence of the scheme (2) depends on the spectral gap between the dominant eigenvalue equal to 1 and the second eigenvalue $\gamma = \max\{|\lambda|: \lambda \text{ eigenvalue of } M^{-1}N \text{ and } \lambda \neq 1\}$. Experimentally (see Example 3 below) the block staircase iteration still performs similarly to the block Gauss-Seidel method when applied for solving linear systems with singular M-matrices in block Hessenberg form. The same behavior is observed for matrices that are localized around the main diagonals. Examples are the covariance matrices with application to the spatial kriging problem (compare with [11]). Other non artificial examples are discussed in Section 3.

Example 2. For the matrix of Example 1 the staircase splitting of the first type gives an iteration matrix having the same eigenvalues $\{1, 1/4, 0, 0\}$ of the Gauss-Seidel scheme.

Example 3. We have performed several numerical experiments with randomly generated singular M-matrices in banded block lower Hessenberg form having the profile depicted in Figure 1.

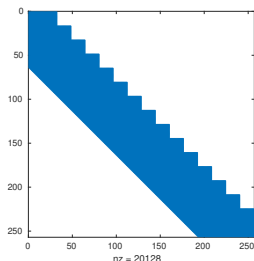


Fig. 1: Spy plot

$kp \backslash k$				
		16	32	64
16	ρ_1			1.6
	ρ_2			1.5
32	ρ_1			1.6
	ρ_2			1.4
64	ρ_1	∞	33.6	2.0
	ρ_2	1.0	1.0	1.4

Table 1: Convergence comparisons

The size of the blocks is k , the block size of the matrix is n and the lower bandwidth is $4k$. We compare the performance of the block Jacobi (BJ), block Gauss-Seidel (BGS) and block staircase (BS) methods for different sizes of the block partitioning denoted as kp . In the following Table 1 we show the maximum value of $\rho_1 = \log(1/\gamma_{BGS})/\log(1/\gamma_{BJ})$ and of $\rho_2 = \log(1/\gamma_{BGS})/\log(1/\gamma_{BS})$ – where γ_M is the second eigenvalue of the iteration matrix of size $N = k \times n$ generated by the method $M \in \{BGS, BJ, BS\}$ over 1000 experiments with

$k = 64$ and $n = 16$. The value ∞ indicates that in some trials the block Jacobi method does not converge due to the occurrence of two or more eigenvalues equal to 1 in magnitude. The results demonstrate that the spectral gap of the iteration matrix in the block staircase method remains close to that one of BGS.

The following **Algorithm 2** provides an implementation of (2) using $M = M_1$. For $i = 1, \dots, n$ let

$$\mathcal{J}_i = \begin{cases} \{i-1, i+1\} \cap \{1, \dots, n\}, & i \text{ even} \\ i, & i \text{ odd.} \end{cases}$$

Algorithm 2 This algorithm approximates the vector $\boldsymbol{\pi}$ by means of the method (2) with $M = M_1$

```

1: Initialization
2: while  $err \geq tol$  &  $it \leq maxit$  do
3:   parfor  $i = 1, \dots, n$  do
4:      $\mathbf{z}_i \leftarrow -\sum_{k \notin \mathcal{J}_i} Q_{i,k} \mathbf{x}_k$ 
5:   end parfor
6:   parfor  $i = 1, 3, \dots, n$  do
7:      $\mathbf{z}_i \leftarrow Q_{i,i}^{-1} \mathbf{z}_i$ 
8:   end parfor
9:   parfor  $i = 2, 4, \dots, n$  do
10:     $\mathbf{z}_i \leftarrow Q_{i,i}^{-1} \left( \mathbf{z}_i - \sum_{k \in \mathcal{J}_i} Q_{i,k} \mathbf{z}_k \right)$ 
11:  end parfor
12:   $\mathbf{z} \leftarrow \frac{\mathbf{z}}{\mathbf{e}^T \mathbf{z}}$ 
13:   $err \leftarrow \|\mathbf{z} - \mathbf{x}\|_1$ ;  $\mathbf{x} \leftarrow \mathbf{z}$ ;  $it \leftarrow it + 1$ 
14: end while
15: return  $\mathbf{x}$ 

```

In the next section numerical experiments are performed to compare the performance of **Algorithm 1** and **Algorithm 2** for solving singular systems of linear equations arising in Markov chain modeling.

3 Numerical Experiments

The experiments have been run on a server with two Intel Xeon E5-2650v4 CPUs with 12 cores and 24 threads each, running at 2.20GHz. Hyper-threading was disabled so that the number of logical processors is equal to physical processors (cores). To reduce the variance of execution times, the number of threads is set to the number of physical cores and each thread is mapped statically to one core. The parallel implementations of **Algorithm 1** –referred to as JGS algorithm– and **Algorithm 2** –referred to as STAIR1 or STAIR2 algorithm depending on the staircase splitting– are based on OpenMP. Specifically, we have used C++20 with the help of Armadillo [22, 23] which also provides integration with LAPACK [3] and OpenBLAS [13].

Our test suite consists of the following transition matrices:

1. The transition rate matrix associated with the queuing model described in [7]. This is a complex queuing model, a BMAP/PHF/1/N model with retrial system with finite buffer and non-persistent customers. We do not describe in detail the construction of this matrix, as it would take some space, but refer the reader to [6, Sections 4.3 and 4.5]. The buffer size is denoted as n . The only change with respect to the paper is that we fix the orbit size to a finite capacity k (when the orbit is full, customers leave the queue forever). We set $n = 15$, which results in a block upper Hessenberg matrix Q of size $N = 110400$ with $k \times k$ blocks of size 138.
2. The transition rate matrix for the model described in Example 1 of [20]. The model describes a time-sharing system with n terminals which share the same computing resource. The matrices are nearly completely decomposable (NCD) so that classical stationary iterative methods do not perform satisfactorily as the spectral gap is pathologically close to unity. We set $n = 80$ which gives a matrix of size $N = 91881$.
3. The transition rate matrix for the model described in Example 3 of [20]. The model describes a multi-class, finite buffer, priority system. The buffer size is denoted as n . This model can be applied to telecommunications modeling, and has been used to model ATM queuing networks as discussed in [26, 20]. We note that the model parameters can be selected so that the resulting Markov chain is nearly completely decomposable. We set $n = 100$ which results in a matrix of size $N = 79220$.
4. The transition rate matrix generated by the set of mutual-exclusion problems considered in [8]. In these problems, n distinguishable processes share a certain resource. Each of these processes alternates between a sleeping state and a resource using state. However, the number of processes that may concurrently use the resource is limited to r , where $1 \leq r \leq n$, so that when a process wishing to move from the sleeping state to the resource using state finds r processes already using the resource, that process fails to access the resource and returns to the sleeping state. We set $n = 16$ and $r = 12$ so that the transition matrix has size $N = 64839$.

All the considered transition matrices are sparse matrices. In Figure 2 we show the spy plots of the matrices generated in tests 1-4. The matrices are stored using the compressed sparse column format. With this method, only nonzero entries are kept in memory. However, despite evident merits this solution has also some drawbacks. In particular, we notice that certain operations such as `submat` calls become relatively expensive because they could be creating temporary matrices. Clearly, the size of the block partitioning of the matrix seriously affects the performance of iterative methods. There is an extensive literature on this topic (see for instance [18, 12] and the references given therein). We have not tried the partition algorithms described in [18, 12] and in this paper we explore the use of blocks of equal size $n_i = \ell$, $1 \leq i \leq n-1$. A test for the change of two consecutive iterates to be less than a prescribed tolerance or the number of iterations to be greater than a given bound is used as stopping criterion. In other words, we stop

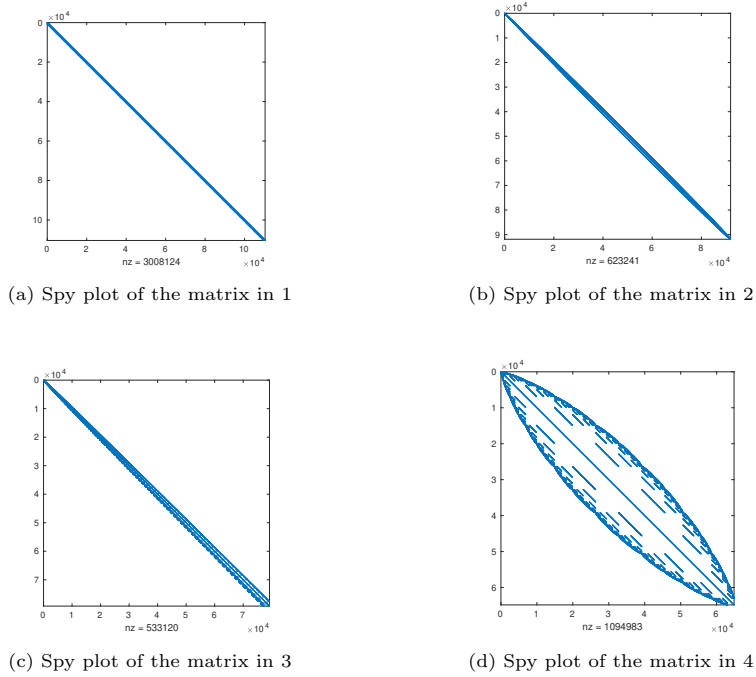


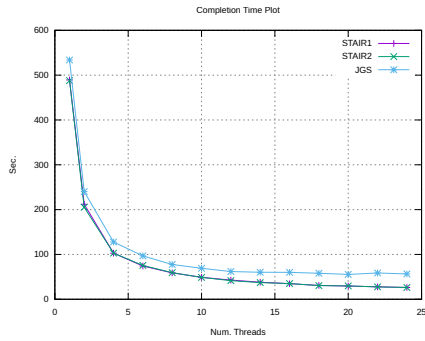
Fig. 2: Illustration of the sparsity pattern of the matrices in our test suite

the iteration if

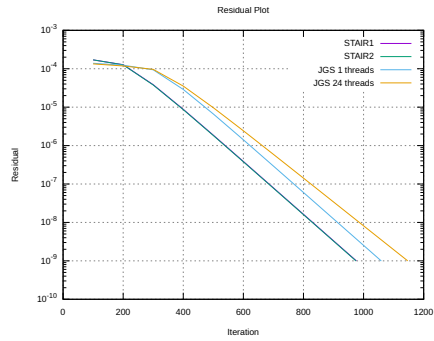
$$\| \mathbf{x}^{(k)} - \mathbf{x}^{(k+1)} \|_1 \leq \epsilon \vee k \geq \text{maxit}.$$

In all the experiments reported below we have used $\epsilon = 1.0e-9$ and $\text{maxit} = 1000$. A larger number of iterations is carried out in some cases to check for divergence of the iterative process. For each algorithm and experiment we measure the sequential completion time T_{seq} , the parallel completion time on m threads $T_{par}(m)$, the speedup $S_p(m) = T_{seq}/T_{par}(m)$ and the efficiency $E(m) = S_p(m)/m$. We also report a plot of the residual $\| \mathbf{x}^{(k)} - \mathbf{x}^{(k+1)} \|_1$ to analyze the convergence of the iterative scheme.

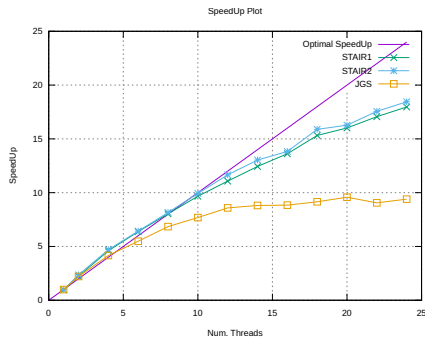
In the first experiment we consider the transition rate matrix generated in 1 with $n = 15$ and $k = 800$. The matrix has size $N = 110400$ and the block partitioning is determined by setting $\ell = 512$. The matrix is block lower Hessenberg and block banded. In Figure 3 we show the plots of completion time, residual, speedup and efficiency generated for this matrix. The convergence of block staircase methods is better than the convergence of the block Gauss-Seidel method. The superlinear speedup is a cache effect. The best sustained computation rate is about 6.5 Gflops on 24 cores. We think that the attained level of performance is quite low with respect to the theoretical peak performance due to the use of sparse linear algebra functionalities.



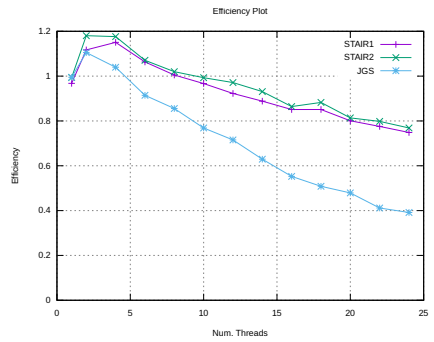
(a) Completion-time plot for the matrix in 1



(b) Residual plot for the matrix in 1



(c) Speedup plot for the matrix in 1

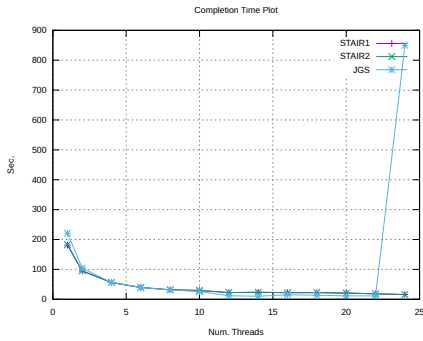


(d) Efficiency plot for the matrix in 1

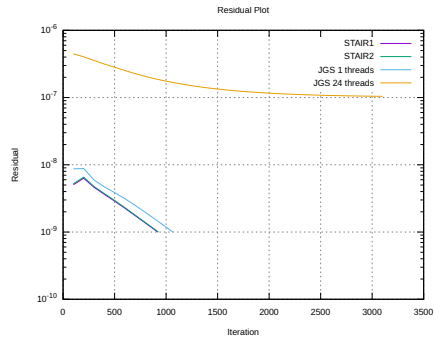
Fig. 3: Illustration of the performance of algorithms JGS, STAIR1 and STAIR2 for the matrix in 1

In Figure 4 we show the plots generated for the matrix in 2 with $n = 80$. The size is $N = 91881$. The matrix is symmetric with bandwidth 1135. We set $\ell = 2048$ so that the matrix is block tridiagonal. The crazy behavior of JGS in Figure 4 and in Figure 5 depends on the fact that the block partitioning varies with the number of threads. In particular, for the nearly completely decomposable example this makes possible divergence or false convergence cases. In the course of experimenting with the JGS method, we have tried various policies for the distribution of blocks to different processors. The chosen policy also goes to affect the behavior and convergence of the algorithm as the number of threads varies. It was chosen in the end to use as fair a distribution policy as possible, trying in general to allocate in each thread the same number of blocks and eventually, in the case of any remaining blocks, evenly distributing the remainder.

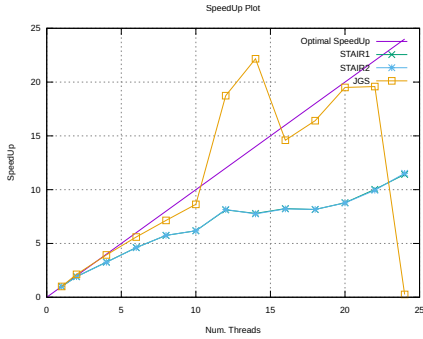
In Figure 5 we show the plots generated for the matrix in 3 with $n = 100$. The matrix has size $N = 79220$, lower bandwidth 2370 and upper bandwidth 1585. We set $\ell = 2048$ so that the matrix is block banded in block lower Hessenberg form.



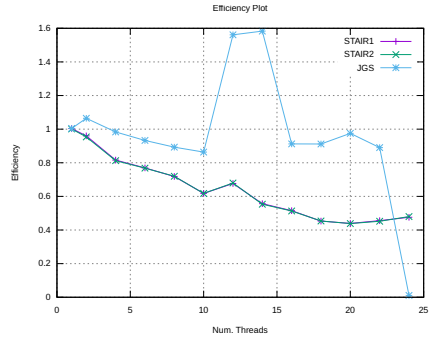
(a) Completion-time plot for the matrix in 2



(b) Residual plot for the matrix in 2



(c) Speedup plot for the matrix in 2

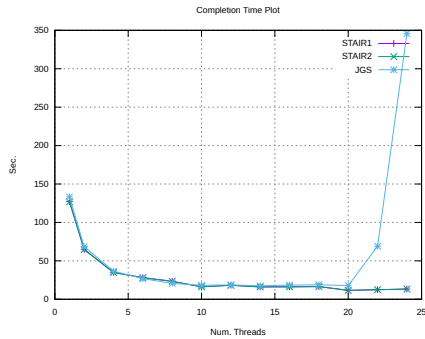


(d) Efficiency plot for the matrix in 2

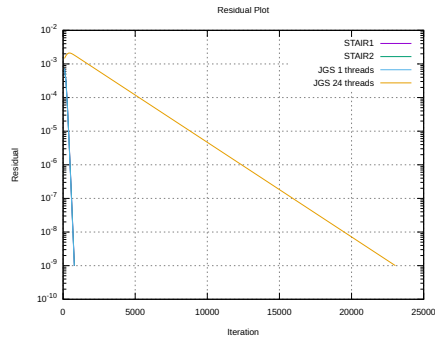
Fig. 4: Illustration of the performance of algorithms JGS, STAIR1 and STAIR2 for the matrix in 2

The results in Figure 4, 5 clearly highlight that the convergence of the JGS method may deteriorate as the number of threads increase since the iteration becomes close to a pure block Jacobi method. In particular for $m = 24$ threads in both examples the method does not converge and the process stops as the maximum number of iterations has been reached. Finally, in Figure 6 we illustrate the plots of completion time and speedup for the matrix generated by 4 with $n = 16$, and $r = 12$. The matrix has size $N = 64839$ and bandwidth 13495. We set $\ell = 256$ so that the matrix is block banded. Since the entries are very rapidly decaying away from the main diagonal all methods perform quite well and the number of iterations in the JGS method is quite insensitive to the number of threads.

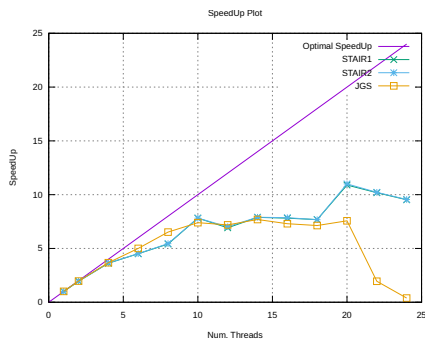
Concerning the parallel performance, we recall that the server has only 24 combined physical cores, and going above 12 required communication between the different CPUs, which inevitably reduces the efficiency of the parallelization. When the number of threads is quite small it is generally observed that the bigger the block size, the shorter is the execution time. Differently, as the number



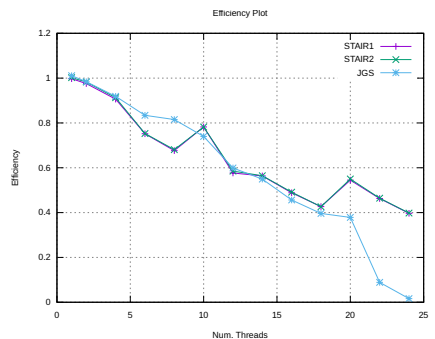
(a) Completion-time plot for the matrix in 3



(b) Residual plot for the matrix in 3



(c) Speedup plot for the matrix in 3



(d) Efficiency plot for the matrix in 3

Fig. 5: Illustration of the performance of algorithms JGS, STAIR1 and STAIR2 for the matrix in 3

of threads increases small blocks promote the parallelism. The reason can be attributed to two main factors: load balancing and communication overhead. In Figure 7 we show the the speedup plot for the test 3 with $\ell = 512$ and $\ell = 1024$, respectively. The comparison with the results reported in Figure 5 with $\ell = 2048$ indicates some improvements. These effects are highlighted in all the conducted experiments. Considering that most consumer hardware has between 2 and 8 or 16 cores, this shows that the proposed method is generally well tuned for the currently available architectures.

4 Conclusions

This paper presents the results of a preliminary experimental investigation of the performance of a stationary iterative method based on a block staircase splitting for solving singular systems of linear equations arising in Markov chain modeling. From the experiments presented, we can deduce that the method is

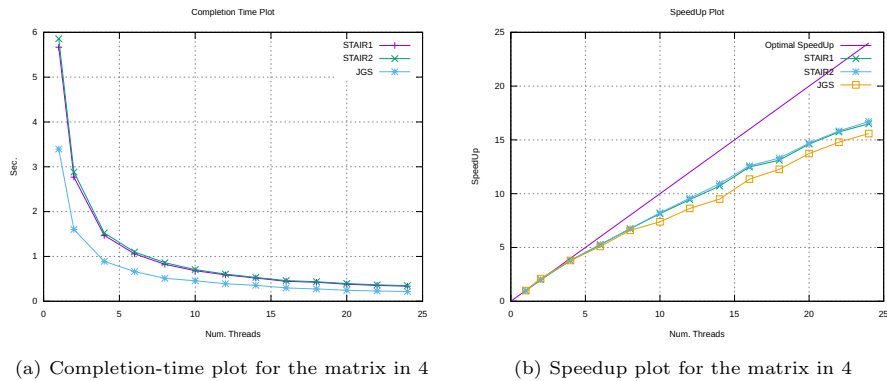
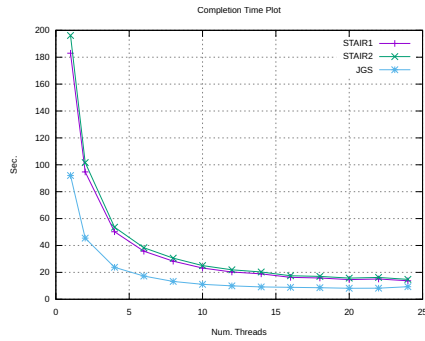


Fig. 6: Illustration of the performance of algorithms JGS, STAIR1 and STAIR2 for the matrix in 4

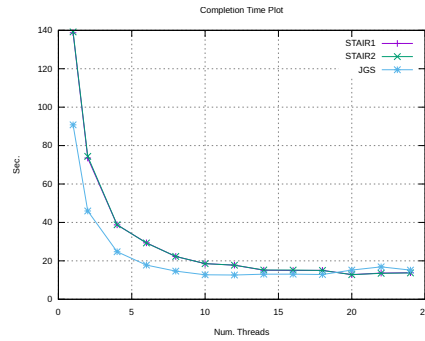
well suited for solving block banded or more generally localized systems in a shared-memory parallel computing environment. The parallel implementation has been benchmarked using several Markovian models. In the future we plan to examine the performance of block staircase splittings in a distributed computing environment and, moreover, their use as preconditioners for other iterative methods.

References

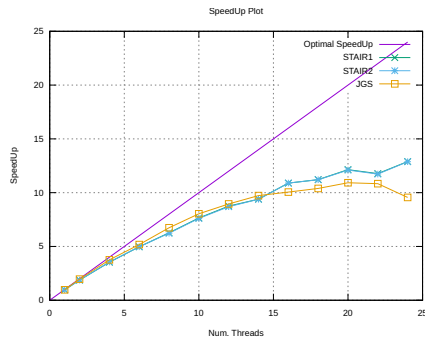
1. Ahmadi, A., Manganiello, F., Khademi, A., Smith, M.C.: A parallel Jacobi-embedded Gauss-Seidel method. *IEEE Transactions on Parallel and Distributed Systems* **32**, 1452–1464 (2021)
2. Amodio, P., Mazzia, F.: A parallel Gauss-Seidel method for block tridiagonal linear systems. *SIAM J. Sci. Comput.* **16**(6), 1451–1461 (1995), <https://doi.org/10.1137/0916084>
3. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Dongarra, J.D.J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: *LAPACK Users' Guide*. SIAM, Philadelphia, Pennsylvania, USA, third edn. (1999)
4. Benzi, M.: Localization in matrix computations: theory and applications. In: *Exploiting hidden structure in matrix computations: algorithms and applications*, *Lecture Notes in Math.*, vol. 2173, pp. 211–317. Springer, Cham (2016)
5. Berman, A., Plemmons, R.J.: *Nonnegative matrices in the mathematical sciences*, *Classics in Applied Mathematics*, vol. 9. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (1994), <https://doi.org/10.1137/1.9781611971262>, revised reprint of the 1979 original
6. Bylina, J., Bylina, B.: Merging Jacobi and Gauss-Seidel methods for solving Markov chains on computer clusters. In: *2008 International Multiconference on Computer Science and Information Technology*. pp. 263–268 (2008). <https://doi.org/10.1109/IMCSIT.2008.4747250>



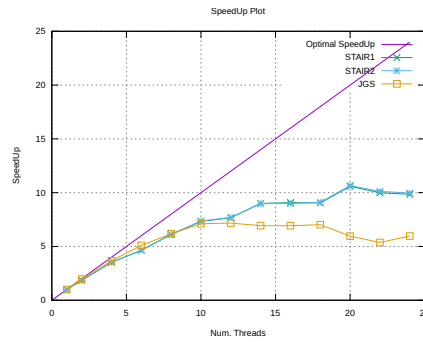
(a) Completion-time plot with $\ell = 512$



(b) Completion-time plot with $\ell = 1024$



(c) Speedup plot with $\ell = 512$



(d) Speedup plot with $\ell = 1024$

Fig. 7: Illustration of the completion time and speedup plot for the matrix 3 with different block sizes

7. Dudin, S., Dudin, A., Kostyukova, O., Dudina, O.: Effective algorithm for computation of the stationary distribution of multi-dimensional level-dependent Markov chains with upper block-Hessenberg structure of the generator. *J. Comput. Appl. Math.* **366**, 112425, 17 (2020), <https://doi.org/10.1016/j.cam.2019.112425>
8. Fernandes, P., Plateau, B., Stewart, W.J.: Efficient descriptor-vector multiplications in stochastic automata networks. *J. ACM* **45**(3), 381–414 (1998), <https://doi.org/10.1145/278298.278303>
9. Gemignani, L., Meini, B.: Relaxed fixed point iterations for matrix equations arising in Markov chain modeling. *Numerical Algorithms* (2023). <https://doi.org/https://doi.org/10.1007/s11075-023-01496-y>
10. Gemignani, L., Poloni, F.: Comparison theorems for splittings of M-matrices in (block) Hessenberg form. *BIT* **62**(3), 849–867 (2022), <https://doi.org/10.1007/s10543-021-00899-4>
11. Ghadiyali, H.S.: Partial Gauss-Seidel Approach to Solve Large Scale Linear Systems. Master's thesis, Florida State University (2016), http://purl.flvc.org/fsu/fd/FSU_2016SP_Ghadiyali_fsu_0071N_13280

12. Klevans, R.L., Stewart, W.J.: From queueing networks to markov chains: The XMARCA interface. *Performance Evaluation* **24**(1), 23–45 (1995). [https://doi.org/https://doi.org/10.1016/0166-5316\(95\)00007-K](https://doi.org/https://doi.org/10.1016/0166-5316(95)00007-K)
13. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Basic linear algebra sub-programs for Fortran usage. *ACM Trans. Math. Softw.* **5**(3), 308–323 (sep 1979), <https://doi.org/10.1145/355841.355847>
14. Lu, H.: Stair matrices and their generalizations with applications to iterative methods. I. A generalization of the successive overrelaxation method. *SIAM J. Numer. Anal.* **37**(1), 1–17 (1999), <https://doi.org/10.1137/S0036142998343294>
15. Marek, I., Szyld, D.B.: Iterative and semi-iterative methods for computing stationary probability vectors of Markov operators. *Math. Comp.* **61**(204), 719–731 (1993), <https://doi.org/10.2307/2153249>
16. Marek, I., Szyld, D.B.: Comparison of convergence of general stationary iterative methods for singular matrices. *SIAM J. Matrix Anal. Appl.* **24**(1), 68–77 (2002), <https://doi.org/10.1137/S0895479800375989>
17. Meyer, C.: *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (2000), <https://doi.org/10.1137/1.9780898719512>
18. O’Neil, J., Szyld, D.B.: A block ordering method for sparse matrices. *SIAM J. Sci. Statist. Comput.* **11**(5), 811–823 (1990), <https://doi.org/10.1137/0911048>
19. Ortega, J.M., Voigt, R.G.: Solution of partial differential equations on vector and parallel computers. *SIAM Rev.* **27**(2), 149–240 (1985), <https://doi.org/10.1137/1027055>
20. Philippe, B., Saad, Y., Stewart, W.J.: Numerical methods in Markov chain modeling. *Operations Research* **40**(6), 1156–1179 (1992), <http://www.jstor.org/stable/171728>
21. Saad, Y.: *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edn. (2003), <https://doi.org/10.1137/1.9780898718003>
22. Sanderson, C., Curtin, R.: Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software* **1**, 26 (2016)
23. Sanderson, C., Curtin, R.: A user-friendly hybrid sparse matrix class in C++. In: Davenport, J.H., Kauers, M., Labahn, G., Urban, J. (eds.) *Mathematical Software – ICMS 2018*. pp. 422–430. Springer International Publishing, Cham (2018)
24. Schneider, H.: Theorems on M -splittings of a singular M -matrix which depend on graph structure. *Linear Algebra Appl.* **58**, 407–424 (1984), [https://doi.org/10.1016/0024-3795\(84\)90222-2](https://doi.org/10.1016/0024-3795(84)90222-2)
25. Shang, Y.: A distributed memory parallel Gauss-Seidel algorithm for linear algebraic systems. *Comput. Math. Appl.* **57**(8), 1369–1376 (2009), <https://doi.org/10.1016/j.camwa.2009.01.034>
26. Stewart, W.J.: *Introduction to the numerical solution of Markov chains*. Princeton University Press, Princeton, NJ (1994)
27. Touzene, A.: A new parallel algorithm for solving large-scale Markov chains. *The Journal of Supercomputing* **67**(1), 239–253 (2014)
28. Wallin, D., Löf, H., Hagersten, E., Holmgren, S.: Multigrid and Gauss-Seidel smoothers revisited: parallelization on chip multiprocessors. In: Egan, G.K., Muraoka, Y. (eds.) *Proceedings of the 20th Annual International Conference on Supercomputing, ICS 2006*, Cairns, Queensland, Australia, June 28 - July 01, 2006. pp. 145–155. ACM (2006), <https://doi.org/10.1145/1183401.1183423>