EXPERIMENTAL STUDY OF PARALLEL ITERATIVE SOLUTIONS OF MARKOV CHAINS WITH BLOCK PARTITIONS

Violeta Migallón, José Penadés,
and Daniel B. Szyld

EXPERIMENTAL STUDY OF PARALLEL ITERATIVE SOLUTIONS OF MARKOV CHAINS WITH BLOCK PARTITIONS

VIOLETA MIGALLÓN*, JOSÉ PENADÉS† AND DANIEL B. SZYLD†

Abstract. Experiments are performed which demonstrate that parallel implementations of block stationary iterative methods can solve singular systems of linear equations in substantially less time that the sequential counterparts. Furthermore, these experiments illustrate the behavior of different partitions of matrices representing Markov chains, when parallel iterative methods are used for their solution. Several versions of block iterative methods are tested.

Key words. Iterative methods, parallel algorithms, linear systems, singular matrices, block methods, two-stage methods, Markov chains.

AMS(MOS) subject classification. 65F10, 65F15.

1. Introduction. In [8] it was shown that a threshold partitioning algorithm can have beneficial effects for the solution of singular linear systems arising in Markov chain modeling. The solution methods studied in [8] include standard block stationary iterative methods based on matrix splittings such as block Jacobi, block Gauss-Seidel, block SOR etc.; see, e.g., [6], [29], or section 2. More recently, an exhaustive study by Dayar and Stewart [13] demonstrates that for these problems stationary block methods based on matrix splittings give better convergence than preconditioned Krylov subspace methods [14], [25]. These experiments, as well as those in [8] were performed on sequential computers.

More explicitly, we know (e.g., from the experiments in [13]) that block stationary iterative methods, including two-stage (or inner-outer) methods are some of the most competitive methods for the solution of Markov chain problems on sequential computers. The convergence of these methods has been extensively studied in the literature (see, e.g., [2], [3], [11], [20], [22], [23], [24], [32]), but there is little experience in their use for the parallel and distributed solution of these problems. We know of the work reported in [4], and [21], where some parallel or distributed solution methods of different kind are discussed. There is a need to gain an understanding of how classical stationary block methods perform in the parallel setting.

In this contribution we show that the use of parallel implementation of block stationary iterative methods can solve singular systems of linear equations in substantially less time that the sequential counterparts. Furthermore, we explore the question on whether (permutations and) partitions such as TPABLO [8] or the ones produced in the MARCA package [19], [28] are beneficial when used in conjunction with parallel block iterative methods. Furthermore, different versions of these methods and implementation strategies are explored and their relative merit discussed.

2. The Block Methods. We use block stationary iterative methods for the solution of $n \times n$ linear systems of equations of the form

$$(1) \qquad\qquad\qquad\qquad Ax = b.$$

We have in mind applications where the matrix A in (1) is singular and the linear system is consistent. These include certain stochastic processes, queuing models [18], Markov chains [26], [29], as well as performance evaluation of computer and other critical systems [9], [33], where Petri Nets play an important role in the models [1], [10].

In the methods we study for the solution of (1), the variables (and equations) are permuted and partitioned into r groups, i.e., $Px = [x_1^T, x_2^T, \ldots, x_r^T]^T$, $x_i \in \mathbb{R}^{n_i}$, $i = 1, \ldots, r$, and $\sum_{i=1}^{r} n_i = n$, and P is a permutation matrix. Thus, the matrix A is permuted and partitioned into $r \times r$ blocks as follows,

$$(2) \qquad\qquad PAP^T = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1r} \\ A_{21} & A_{22} & \cdots & A_{2r} \\ \vdots & \vdots & & \vdots \\ A_{r1} & A_{r2} & \cdots & A_{rr} \end{bmatrix}$$

with the diagonal blocks A_{ii} being square of order n_i. In this paper, we usually consider the permutation obtained from either TPABLO [8] or from the MARCA package [19], [28]. We refer the reader to these references for descriptions on how these permutations are obtained. We also explored the use of blocks of (approximately) equal size. We have not tried the partition algorithm recently developed by Dayar [12], but we would expect results similar to those with TPABLO or MARCA; see further the discussion in section 3.2.

In the description of the algorithms that follows we assume that the permutations have been performed, and that the matrix A of (1) has already the form in the right hand side of (2). For the most part, we have chosen to divide the work among processors in an evenly fashion, maintaining in all cases the blockings obtained by the partitioning algorithm (TPABLO, MARCA, equal size blocks) in order to ensure an appropriate load balance among the processors. (See the comments in section 3.3 for the case of an uneven distribution of the blocks). In the experiments reported in this paper, the number of groups obtained, r, is larger than the number of processors p. Thus, we have assembled blocks from (2) into p groups, each group assigned to one processor. In other words, within each of the p blocks we have maintained a structure like that in the right hand side of (2) with the corresponding blocks of the partition used (TPABLO, MARCA, equal size blocks). There are r_ℓ blocks assigned to processor ℓ, $\ell = 1, \ldots, p$, and thus $\sum_{\ell=1}^{p} r_\ell = r$.

In order to describe the different versions of the iterative methods tested, we describe the Block Jacobi (BJ) and Block Gauss-Seidel (BGS) algorithms for a generic number of blocks s. The latter, shown in Algorithm 2 below, is the block version of the standard Gauss-Seidel algorithm (GS) (where $n_i = 1$ for all i, and $r = n$); see, e.g., [6], [29]. In the description of the algorithms, as well as in the rest of the paper, we concentrate on the Markov chains problem, and thus, we assume that $b = 0$ in (1), and that the solution x is normalized so that $x^T e = 1$, where e is a vector of all components equal to one. In fact, in the algorithms described below, such normalization is assumed at every iteration.

ALGORITHM 1. BLOCK JACOBI (BJ). *Given an initial vector*
$x^{(0)^T} = [x_1^{(0)^T}, x_2^{(0)^T}, \ldots, x_s^{(0)^T}]$,

For $k = 1, 2, \ldots$, until convergence.

For $i = 1$ to s

(3) $\qquad$ Solve (or approximate) $A_{ii}x_i^{(k)} = \sum_{\substack{j=1 \\ j \neq i}}^{s} A_{ij}x_j^{(k-1)}$.

The linear systems (3) can be solved independently of each other. Thus, this algorithm is inherently parallel. This parallelism is best exploited if the number of blocks matches the number of processors ($s = p$). When each solution of (3) is approximated by another iterative method, this is called a two-stage (or inner-outer) method; see, e.g., [23] and the references therein. We note that the sequential experiments reported in [8] and in [13] are of this kind.

ALGORITHM 2. BLOCK GAUSS-SEIDEL (BGS). *Given an initial vector*
$x^{(0)^T} = [x_1^{(0)^T}, x_2^{(0)^T}, \ldots, x_s^{(0)^T}]$,

For $k = 1, 2, \ldots$, until convergence.

For $i = 1$ to s

(4) $\qquad$ Solve (or approximate) $A_{ii}x_i^{(k)} = \sum_{j=1}^{i-1} A_{ij}x_j^{(k)} + \sum_{j=i+1}^{s} A_{ij}x_j^{(k-1)}$,

where the sums in (4) are not present if the lower limit is higher than the upper limit.

It is clear from the right hand side in (4) that the algorithm is inherently sequential. This is because the computation of the previous subvector $x_{i-1}^{(k)}$ has to be completed before the beginning of the solution or approximation of the system (4) can begin. We are ready to describe the different parallel two-stage (actually three-stage or nested) methods explored in this paper. We assume that there are p processors.

ALGORITHM 3. *Divide the r blocks of (2) into p groups, each assigned to a different processor.*

1. *Perform parallel BJ (with $s = p$ in Algorithm 1), i.e., each processor approximates the solution of one linear system (3). (This is the outer iteration).*
2. *Each solution of (3) in Step 1 is approximated using one step of BGS (with $s = r_\ell$ in Algorithm 2).*
3. *Each solution of (4) in Step 2 is approximated by a fixed number q of GS iterations.*

As is well known, BJ, as well as BGS and the nested Algorithm 3, may not converge, especially for singular systems. In order to guarantee convergence, one can use the customary device described, e.g., in [6], of shifting the iteration matrix from T to

(5) $$T_\delta = \delta T + (1 - \delta)I,$$

where $0 < \delta < 1$, $T = M^{-1}N$ and $A = M - N$ is a splitting representing the iteration of the corresponding algorithm; see, e.g., [5], [6], [23], [29]. This device guarantees that if λ is the eigenvalue of T_δ of largest absolute value other than 1, then $|\lambda| < 1$. This shift is performed at the end of each outer iteration. (At this point of the computation is where the vector

$x^{(k)}$ is normalized). Of course, neither the matrix T nor the shifted matrix T_δ are explicitly computed, only their action on the previous iterate $x^{(k-1)}$ is computed.

In parallel and distributed computing, the communication time can be onerous. This is the time spent sending data from one processor to another. In Algorithm 3, data is exchanged between processors at the end of each outer iteration. Since the communication time can take a significant proportion of the total time, it is natural to try to perform more computations before exchanging data. The following algorithm differs from Algorithm 3 in that Step 2 is performed a fixed number t of iterations before the outer iteration is completed.

ALGORITHM 4. *Divide the r blocks of (2) into p groups, each assigned to a different processor.*
1. *Perform parallel BJ (with $s = p$ in Algorithm 1).*
2. *Each solution of (3) in Step 1 is approximated using t steps of BGS (with $s = r_\ell$ in Algorithm 2).*
3. *Each solution of (4) in Step 2 is approximated by a fixed number q of GS iterations.*

In Algorithms 3 and 4, the stopping criterion of the innermost loop is a fixed number of iterations. This criterion has been used in several implementations of two-stage methods. Another stopping criterion is to compute the norm of the difference between two consecutive iterates and stop when it falls below a prescribed (innermost) tolerance. This criterion is what differs the following algorithm from Algorithm 4.

ALGORITHM 5. *Divide the r blocks of (2) into p groups, each assigned to a different processor.*
1. *Perform parallel BJ (with $s = p$ in Algorithm 1).*
2. *Each solution of (3) in Step 1 is approximated using t steps of BGS (with $s = r_\ell$ in Algorithm 2).*
3. *Each solution of (4) in Step 2 is approximated using GS until the norm of the difference between two consecutive iterates is below a prescribed fixed tolerance ε_{inner}.*

Implementations other than those described in Algorithms 3–5 are of course possible, and we have experimented with some of them. For example, one can use the innermost tolerance instead of a fixed number of inner iterations in Algorithm 3. As we shall see in section 3.2, for different values of ε_{inner}, Algorithm 5 was never better than the other two, and thus, we concluded that this possible implementation was not warranted. Furthermore, everywhere when we use BGS or GS, one can use the possible faster SOR (or block SOR) algorithm [29], which includes a relaxation parameter. We have obtained faster parallel computational times by varying this parameter, and for each case, one can approach experimentally an optimal value for this parameter. Nevertheless, for the sake of brevity and clarity we report only experiments with BGS or GS, i.e., with the relaxation parameter being equal to one.

We approximate the solution of each system (4), with GS (or SOR). Alternatively iterative aggregation / disaggregation (see, e.g., [29]) can be used, as done in the experiments in [13].

Another possible implementation is to use different number of inner iterations for different blocks, i.e., various values of q in Step 3 in Algorithms 3 and 4, say q_i for the ith block; see, e.g., [23] and the references therein. See also an example in section 3.3.

In our experiments, we have used as the global stopping criterion (i.e., in the outer iteration) a test for the change of two consecutive iterates (in all p blocks) to be less than a

prescribed tolerance. In other words, we test if

$$(6) \qquad \qquad \|x^{(k)} - x^{(k-1)}\|_1 < \varepsilon.$$

In all experiments reported in the next section we have used $\varepsilon = 10^{-4}$, while for the inner tolerance we used $\varepsilon_{inner} = 10^{-3}$.

3. Numerical Experiments. All matrices used in our experiments have been used in sequential experiments by several authors, including [8], [13], [25], [30], and can be obtained from [27]. We actually run our codes on all the models in [27] with matrices of various sizes, but to focus our discussion, we present here results based on two different models, and three matrices. The timings and conclusions we present here can be considered as representative of the larger set of experiments performed.

One model corresponds to an interactive computer system, described as Example 1 in [25] or in [30]. We obtained two resulting NCD matrices corresponding to 30 and 50 users. These matrices have 5,456 and 23,426 states (n, the number of variables) and 35,216 and 156,026 nonzeros, respectively (NCD stands for nearly completely decomposable; see, e.g., [29] for more details).

The other one is a telecommunications model of impatient telephone customers on a computerized telephone exchange, described, e.g., in [13], [25]. The matrix we use is of order 17,081 and has 84,211 nonzeros, and we label it TCOMM.

All experiments reported in this paper were performed on an IBM SP2 at the Universidad de Alicante, running an IBM version of the library of parallel routines PVM [15], [16]. All times are in seconds. With the stopping criterion (6), all final approximations x were such that $\|Ax\|_2$ was of the order of 10^{-4} to 10^{-5}.

TABLE 1
Algorithm 3. Matrix: NCD of order 5456. TPABLO permutation.

p	One		Four		Six	
q	Iter.	time	Iter.	time	Iter.	time
1	7843	27.42	7848	27.69	7848	26.67
5	2275	18.39	2281	10.74	2281	9.58
10	1288	17.68	1294	8.20	1294	6.78
15	917	17.94	924	7.22	924	5.71
20	719	18.21	726	6.71	726	5.16
30	508	18.68	516	6.47	516	4.68
35	445	18.93	453	6.27	453	4.56
40	397	19.01	404	6.22	404	4.48
50	327	19.55	335	6.26	335	4.46
60	279	19.87	287	6.25	287	4.35

3.1. Performance of Different Algorithms. In this subsection we show how the parallel algorithms are faster than the sequential ones (cf. [21]), and compare the performance of the three nested algorithms described in section 2.

In Table 1 we report results corresponding to Algorithm 3 for the NCD matrix of order 5456, with varying numbers of inner iterations q, and with different number of processors

TABLE 2
Matrix: NCD of order 5456. TPABLO permutation.

$p = 4$	blocks size	1309	1295	1474	1378		
	nonzeros in diag. blocks	6197	5995	6838	6266		
$p = 6$	blocks size	931	979	851	966	904	825
	nonzeros in diag. blocks	4415	4595	3871	4498	4232	3685

$p = 1$, 4, 6. The NCD matrix was first permuted with the algorithm TPABLO[1] using as blocking parameters $\alpha = \beta = 0.5$ and a threshold of 0.01, with no minimum or maximum block sizes; see [8] for an explanation of these parameters. A value of $\delta = 0.95$ was used for the shift in (5). The same values for α, β, threshold, and δ were used in all cases reported in this paper. The resulting 31 blocks are grouped into p blocks. The order of each diagonal block is reported in Table 2. In the same table, we note the number of nonzeros in the r_ℓ diagonal blocks in each group. Note that the total nonzeros in the diagonal blocks is 25,296, or about 72% of the nonzeros in the original matrix.

TABLE 3
Algorithm 3. Matrix: TCOMM of order 17081. TPABLO permutation.

p	One		Four		Six	
q	Iter.	time	Iter.	time	Iter.	time
1	2914	32.70	2931	26.73	2942	24.19
3	1078	19.97	1096	17.53	1109	10.53
5	711	18.41	730	10.74	743	8.08
8	509	18.58	529	11.61	543	6.82
10	444	19.45	464	16.88	478	6.67
15	358	22.23	378	15.13	392	6.92
20	316	25.16	336	15.56	349	6.90
25	291	28.51	310	17.13	323	7.40
30	274	31.77	293	16.04	305	7.92
35	262	35.14	281	17.20	293	8.53
40	253	38.44	271	18.14	283	9.19
50	240	46.06	258	18.71	269	10.42
60	231	51.81	248	20.55	259	11.58

Let us highlight some observations of the results in Table 1. With $q = 1$ the parallel implementations take about the same time as the sequential run ($p = 1$). The best parallel implementations reduce the computational time by about a factor of 3 for $p = 4$ and by about a factor of 4 for $p = 6$, compared to the best sequential run. In all cases, for fixed q, $q > 1$, the parallel runs are considerably faster than the uniprocessor time. Observe also that the number of iterations to reach the stopping criterion (6) remains pretty constant as the number of processors increases. Similar observations can be made on the results of the matrix TCOMM of order 17,081 in Table 3. The TPABLO permutation resulted in 86 blocks, grouped as described in Table 4. For this matrix, the TPABLO parameters for minimum

---

[1] In this paper we use the version called TPABLO1 in [8].

and maximum block size were set to 10 and 200, respectively. The number of nonzeros in the diagonal blocks is 78,499, or more than 93% of the nonzeros in the matrix.

TABLE 4
Matrix: TCOMM of order 17081. TPABLO permutation.

$p = 4$	blocks size	4222	4221	4422	4216		
	nonzeros in diag. blocks	19480	19395	20350	19274		
$p = 6$	blocks size	2815	2814	2814	2814	3015	2809
	nonzeros in diag. blocks	13007	12942	12926	12958	13853	12813

It can also be observed in Tables 1 and 3 that for a fixed number of processors p, the computational time starts to decrease as the number of inner iterations q increases until some "optimal" value of q after which the time starts to increase. This behavior can be easily explained, and it is typical of nested iterative methods; see, e.g., [7], [17], [31]. If q is small, the linear systems (4) are poorly approximated, and thus, a large number of outer iterations, with its associated computational cost, is needed for global convergence. If q is too large, even though the number of outer iterations decreases, the computational work grows since too many inner iterations are performed. Somewhere in between these two cases, lies some optimal value of q. In general, this optimal value is hard to predict and, as it can be seen in Tables 1 and 3, it is has a different value for each value of p, the number of processors.

TABLE 5
Algorithm 4. Matrix: TCOMM of order 17081. TPABLO permutation.

p/q	4 / 5		6 / 10	
t	Iter.	time	Iter.	time
1	730	10.74	478	6.67
2	393	10.65	280	5.61
3	281	10.33	214	5.73
5	192	10.21	160	6.33
6	171	8.06	146	6.77
9	134	9.47	122	8.05
15	105	11.45	101	10.70

The same rationale explains why for a fixed pair of values of p and q (and the same global stopping criterion (6)), a similar behavior is obtained when varying the number of iterations t of Step 2 in Algorithm 4. In other words, increasing t will at first decrease the computational time until some optimal value is reached, and then increasing the number of iterations t increases the computational time. This phenomenon is illustrated in Tables 5 and 6. The results in these tables are typical. The computational times always can be improved by increasing t beyond $t = 1$ (which is the same as Algorithm 3); compare with Tables 1 and 3. Sometimes, though this improvement is small. In other words, there always will be a run with Algorithm 4 which will be faster than the best run with Algorithm 3.

We also note that, as any function of two variables, the "optimal" pair of values q, t, may not correspond to first finding the best q for $t = 1$ and then varying t. Nevertheless, this strategy should give a good parallel time. To illustrate this situation, we note that the values

of q chosen for Table 5 are those which provided the fastest times in Table 3, namely $q = 5$ in the case of four processors and $q = 10$ in the case of six. We found values for q and t, which improved the performance of Algorithm 4 reported in Table 5, namely for four processors $q = 3$ and $t = 20$, converging in 8.01 sec. (112 iter.), or $q = 4$ and $t = 15$, in 7.87 sec. (114 iter.). Similarly, for six processors Algorithm 4 converged in 5.40 sec. (340 iter.) for $q = 3$ and $t = 4$, or 5.19 sec. (216 iter.) for $q = 5$ and $t = 5$.

TABLE 6
Algorithms 4 and 5. Matrix: NCD of order 5456. TPABLO permutation.

	Algorithm 4				Algorithm 5			
p	4		6		4		6	
q or ε_{inner}	20		50		.001		.001	
t	Iter.	time	Iter.	time	Iter.	time	Iter.	time
1	726	6.71	335	4.46	7312	31.45	7312	32.33
2	404	6.38	187	4.41	4464	22.85	4464	21.75
3	287	6.36	134	4.52	3282	19.93	3282	17.95
5	187	6.62	90	5.00	2198	17.58	2198	14.73
7	142	6.81	72	5.42	1678	16.50	1678	13.73
9	116	7.11	63	6.11	1369	16.95	1369	13.19
10	107	7.23	61	6.54	1256	15.36	1256	12.40
15	80	8.01	54	8.65	900	14.76	900	11.09

We end this subsection with a discussion of the performance of Algorithm 5. The results reported in Table 6 are representative of other runs we have performed. For $t = 1$, the timings are almost always worse than those of Algorithm 3. When they are better, there are always many values of q for which Algorithm 3 is better. Similarly, for the same value of t, Algorithm 4 is better for a wide range of numbers of inner iterations q. We are convinced that a fixed number of inner iterations is a better choice than a fixed tolerance.

3.2. Different Partitions. In this subsection we concentrate on the performance of the three different permutation and partition methods considered in this paper, namely TPABLO, MARCA, and equal size blocks. We emphasize that it is not our intention to do a comparison study between these three partition methods. Instead, our goal is to point to their suitability for generating partitions of the matrices representing Markov chains to be used in conjunction with the parallel algorithms described in section 2.

We first discuss computational results for the NCD matrix of order 23,426. These results can be compared to those in Tables 1 and 2 of the smaller NCD matrix. The TPABLO parameters are the same as for that matrix. The resulting 51 blocks are grouped as shown in Table 7 for the case of four processors. For this matrix, the algorithm from MARCA with the same threshold of 0.01 produced an equivalent permutation. By this we mean that the two permutations produce the same 51 groups of variables, although these groups of variables come in different order and the variables within each group (or block) are also not in the same order. This observation is consistent with the experience reported in [8]. The total number of nonzeros in the diagonal blocks is 111,826, or 71.6% of the nonzeros of the matrix. Note that this is the same proportion as observed in the case of the smaller NCD matrix in

section 3.1. The computational effort for convergence of Algorithm 3 for both partitions is shown in Table 8.

NCD Matrix of order 23426. Four Processors.

TPABLO	blocks size	5660	6019	5797	5950
	nonzeros in diag. blocks	27360	28587	27809	28070
MARCA	blocks size	5984	5496	5816	6130
	nonzeros in diag. blocks	27808	26312	28036	29670

TABLE 8
Algorithm 3. Four processors. Matrix: NCD of order 23426.

Permut.	TPABLO		MARCA	
q	Iter.	time	Iter.	time
1	8772	146.68	9257	155.73
2	5112	88.46	5353	97.60
3	3692	70.31	3848	74.65
6	2092	48.90	2162	51.27
10	1366	37.31	1401	39.29
15	971	33.39	989	33.44
20	761	30.19	770	30.70
30	538	28.19	540	29.11
50	348	28.35	343	26.23
70	261	27.87	254	25.48
90	211	27.03	203	25.27
100	193	27.22	185	26.44
110	179	27.91	169	25.12
120	166	27.47	157	25.30

As it can be expected from equivalent partitions, the times and iteration counts reported in Table 8 are similar for each value of q, with a variation of no more than 10%. The times for TPABLO are better for smaller q, and those for MARCA are better for larger values of it. For these NCD matrices, the sequential times for these two partitions were also similar [8]. Of course, this does not take into account the computer time for the partition itself which is more expensive for TPABLO.

For other matrices in [27], TPABLO and MARCA do not produce equivalent partitions, especially when a minimum and maximum block size is imposed in TPABLO, as was done for the TCOMM matrix; see section 3.1. For all these matrices, our experience with MARCA was that the blocks obtained were either too big or too small (depending of the threshold parameter used) to combine them in approximately equal size groups for distribution to the p processors. These partitions are consistent with the calculations in [13]. These groups are of course determined by the nearly completely decomposable structure of these matrices, and not by the method used by MARCA. This is why we expect similar results with the new method by Dayar [12]. The use of the maximum block size in TPABLO allows us to obtain

blocks which can be joined with others in a group for a processor, and at the same time take advantage of the connectivity of the graph of the matrix. The results in Table 3 illustrates the suitability of this approach for parallel computing.

As pointed out already in [13], there are cases where the connectivity of the graph of the matrix does not constrain the choice of nodes by TPABLO, and the resulting partition is the same as that produced by chosing the blocks of equal size (the maximum block size).

In our extensive experiments, when the TPABLO partitions are not exactly the same as equal size blocks, the latter produces either worse or about the same computational times with our parallel algorithms. This was observed for uniprocessors in [8]. Here we confirm the same in the parallel setting. Furthermore, in some cases equal size groups obtained after the matrix was permuted with MARCA (or TPABLO) is a better behaved partition than that taken from the original order. We illustrate this with the large NCD matrix in Table 9. Compare these times with those in Table 8.

TABLE 9
Algorithm 3. Four processors. Equal size blocks. Matrix: NCD of order 23426.

	MARCA						Original order			
Size	100 by 100		200 by 200		300 by 300		100 by 100		200 by 200	
q	Iter.	time	Iter.	time	Iter.	time	Iter.	time	Iter.	time
1	9198	167.96	9198	163.60	9198	163.97	8899	176.47	8899	164.69
2	6473	131.07	6371	124.62	5537	104.30	7560	163.27	7532	153.87
4	4752	107.29	4524	102.95	3260	76.21	6427	150.68	6389	146.55
6	4041	102.94	3791	100.19	2441	57.44	5869	149.52	5824	148.41
8	3615	101.30	3366	96.30	2017	56.15	5523	158.60	5473	150.75
9	3454	109.59	3210	94.59	1873	55.07	5394	156.96	5341	159.15
10	3317	108.50	3078	95.74	1757	51.42	5284	161.97	5228	161.24
15	2837	117.08	2629	112.10	1397	48.98	4914	191.31	4848	180.32
20	2535	137.10	2358	112.57	1208	53.28			4626	244.05
25	2319	172.17	2171	122.01	1086	53.22				
30	2153	294.50			999	54.81				
40					933	60.79				

3.3. Uneven Distribution of Blocks. In selected cases, we have ran experiments with an uneven distribution of the blocks, and in these cases always adjusting the number of inner iterations in an attempt to maintain a balanced load. We present in Table 10 two such runs for the two NCD matrices, for the case of four groups. The first two groups are about half the size of the other two, and thus, the number of inner iterations q for the first two are doubled. In these examples q for the first two groups is 20 and for the last two is 10. We ran these experiments with one and four processors.

If we compare these times with those corresponding to $q = 10$ and $q = 20$ in Tables 1 and 8, we see that they are about 10–20% worse than the best of these times ($q = 10$ for the smaller matrix and $q = 20$ for the larger one), but better than the other time. Note that the gain from one to four processors is similar to the case of evenly distributed groups. In general, the best timings obtained in our experiments are about 10–20% worse than the best times for the evenly distributed blocks, e.g., those reported in sections 3.1 and 3.2. We

Table 10

Algorithm 3. NCD matrices. TPABLO permutation. Uneven blocks: q = 20, 20, 10, 10.

order	blocks size				Iter.	time, $p = 1$	time, $p = 4$
5456	931	979	1817	1729	970	17.76	7.29
23426	3679	3622	8282	7843	1048	82.36	35.64

conclude therefore that the uneven partition is not generally recommended as a productive approach. An exception could be when the blocks produced by the partition are so big to make this uneven groups necessary to maintain the connected components of the graph of the matrix intact. In the latter case, our experiments suggest that similar gains can be achieved by a proper adjustment of the number of inner iterations in each block so as to maintain a balanced load among the processors.

4. Conclusions. From the experiments presented, we can conclude that the parallel implementation of classical block two-stage (or nested) methods provides a substantial reduction in computational time when solving singular systems of linear equations. Furthermore, increasing the number of processors decreases the computational time.

For the stopping criterion of the innermost iteration, a fixed number of iterations was shown to be preferable over a fixed tolerance.

All partitions discussed in the paper present computational savings in the parallel setting. Dividing the matrix in blocks of equal size provides the cheapest partition, but the solution times are in general slower than with partitions such as MARCA or TPABLO.

We observe that comparing Table 8 and Table 1, one can see that the algorithms discussed in this paper scale well in the sense that for a similar problem of larger size, the computational time is somehow proportional to the increase in size. In this case it is proportional to the increase in the number of nonzeros in the matrix as well as the number of nonzeros in the diagonal blocks. A similar observation follows from Table 10.

REFERENCES

[1] Marco Ajmone-Marsan, Gianfranco Balbo, and Giovanni Conte. A class of Generalized Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2:93–122, 1984.

[2] George P. Barker and Robert J. Plemmons. Convergent iterations for computing stationary distributions of Markov chains. *SIAM Journal on Algebraic and Discrete Methods*, 7:390–398, 1986.

[3] Vincent A. Barker. Numerical solution of sparse singular systems of equations arising from ergodic Markov chains. *Communications in Statistics. Stochastic Models*, 5:355–381, 1989.

[4] Michele Benzi, Fiorella Sgallari, and Giulia Spaletta. A parallel block projection method of the Cimmino type for finite Markov chains. In William J. Stewart, editor, *Computations with Markov Chains: Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains*, pages 65–80. Kluwer Academic, Dordrecht, 1995.

[5] Michele Benzi and Daniel B. Szyld. Existence and uniqueness of splittings for stationary iterative methods with applications to alternating methods. *Numerische Mathematik*, 76:309–321, 1997.

[6] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, New York, third edition, 1979. Reprinted by SIAM, Philadelphia, 1994.

[7] Rafael Bru, Violeta Migallón, José Penadés, and Daniel B. Szyld. Parallel, synchronous and asynchronous two-stage multisplitting methods. *Electronic Transactions on Numerical Analysis*, 3:24–38, 1995.

[8] Hwajeong Choi and Daniel B. Szyld. Application of threshold partitioning of sparse matrices to Markov chains. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium, IPDS'96, Urbana-Champaign, Illinois, September 4–6, 1996*, pages 158–165, Los Alamitos, California, 1996. IEEE Computer Society Press.

[9] Gianfranco Ciardo and Kishor S. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18:37–59, 1993.

[10] Gianfranco Ciardo, Kishor S. Trivedi, and Jogesh Muppala. SPNP: stochastic Petri net package. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models (PNPM'89)*, pages 142–151, Kyoto, Japan, 1989. IEEE Computer Society Press.

[11] Achiya Dax. The convergence of linear stationary iterative processes for solving singular unstructured systems of equations. *SIAM Review*, 32:611–635, 1990.

[12] Tugrul Dayar. Permuting Markov chains to nearly completely decomposable form. Technical Report BU-CEIS-9808, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, August 1998. Available online at `ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9808.ps.z`.

[13] Tugrul Dayar and William J. Stewart. Comparison of partitioning techniques for two-level iterative methods on large, sparse Markov chains. Technical Report BU-CEIS-9805, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, April 1998. Available online at `ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9805.ps.z`.

[14] Roland W. Freund and Marlis Hochbruck. On the use of two QMR algorithms for solving singular systems and applications in Markov Chain modelling. *Numerical Linear Algebra with Applications*, 1:403–420, 1994.

[15] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 User's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, September 1994.

[16] IBM Corporation. IBM PVMe for AIX User's Guide and Subroutine Reference. Technical Report GC23-3884-00, IBM Corp., Poughkeepsie, NY, October 1995.

[17] Mark T. Jones and Daniel B. Szyld. Two-stage multisplitting methods with overlapping blocks. *Numerical Linear Algebra with Applications*, 3:113–124, 1996.

[18] Linda Kaufman. Matrix methods for queuing problems. *SIAM Journal on Scientific and Statistical Computing*, 4:525–552, 1983.

[19] Richard L. Klevans and William J. Stewart. From queuing networks to Markov chains: The XMARCA interface. *Performance Evaluation*, 24:23–45, 1995.

[20] Ivo Marek and Daniel B. Szyld. Iterative and semi-iterative methods for computing stationary probability vectors of Markov operators. *Mathematics of Computation*, 61:719–731, 1993.

[21] P. Marenzoni, S. Caselli, and G. Conte. Analysis of large GSPN models: a distributed solution tool. In *Proceedings of the 7th Internaitonal Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 122–131, St. Malo, France, June 1997. IEEE Computer Society Press.

[22] Carl D. Meyer and Robert J. Plemmons. Convergent powers of a matrix with applications to iterative methods for singular systems. *SIAM Journal on Numerical Analysis*, 14:699–705, 1977.

[23] Violeta Migallón, José Penadés, and Daniel B. Szyld. Block two-stage methods for singular systems and Markov chains. *Numerical Linear Algebra with Applications*, 3:413–426, 1996.

[24] Dianne P. O'Leary. Iterative methods for finding the stationary vector for Markov chains. In Carl D. Meyer and Robert J. Plemmons, editors, *Linear Algebra, Markov Chains and Queuing Models*, IMA Volumes in Mathematics and its Applications, Vol. 48, pages 125–136, New York – Berlin, 1993. Springer Verlag.

[25] Bernard Philippe, Yousef Saad, and William J. Stewart. Numerical methods in Markov chain modeling. *Operations Research*, 40:1156–1179, 1992.

[26] Eugene Seneta. *Non-negative Matrices and Markov Chains*. Springer Verlag, New York – Heidelberg – Berlin, second edition, 1981.

[27] William J. Stewart. MARCA Models: A collection of Markov chain models. Available online at `http://www.csc.ncsu.edu/faculty/WStewart/MARCA_Models/MARCA_Models.html`.

[28] William J. Stewart. MARCA: Markov chain analyzer. In William J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 37–61, New York - Basel - Hong Kong, 1991. Marcel Dekker.

[29] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, New Jersey, 1994.

[30] William J. Stewart and Wei Wu. Numerical experiments with iteration and aggregation for Markov chains. *ORSA Journal on Computing*, 4:336–350, 1992.

[31] Daniel B. Szyld and Mark T. Jones. Two-stage and multisplitting methods for the parallel solution of linear systems. *SIAM Journal on Matrix Analysis and Applications*, 13:671–679, 1992.

[32] Kunio Tanabe. Characterization of linear stationary iterative processes for solving a singular system of linear equations. *Numerische Mathematik*, 22:349–359, 1974.

[33] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.