Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 263–268
ISBN 978-83-60810-14-9
ISSN 1896-7094

Merging Jacobi and Gauss-Seidel Methods for Solving Markov Chains on Computer Clusters

Jarosław Bylina
Institute of Mathematics
Marie Curie-Sklodowska University
plac Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: jmbylina@hektor.umcs.lublin.pl

Beata Bylina
Institute of Mathematics
Marie Curie-Sklodowska University
plac Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: beatas@hektor.umcs.lublin.pl

***Abstract*—The authors consider the use of the parallel iterative methods for solving large sparse linear equation systems resulting from Markov chains—on a computer cluster. A combination of Jacobi and Gauss-Seidel iterative methods is examined in a parallel version. Some results of experiments for sparse systems with over 3×10^7 equations and about 2×10^8 nonzeros which we obtained from a Markovian model of a congestion control mechanism are reported.**

I. Introduction and Motivation

DISCRETE-STATE models are widely employed for modeling and analysis of communication networks and computer systems. It is often convenient to model such a system as a continuous time Markov chain, provided probability distrbutions are assumed to be exponential (or combinations of exponential ones).

A CTMC (Coninuous-Time Markov Chain) may be represented by a set of states and a transition rate matrix $\mathbf{Q}$ containing state transition rates as coefficients, and can be analysed by using probabilistic model checking. Such an analysis proceeds by specifying desired performance properties as some temporal logic formulae, and by verifying these properties automatically, using the appropriate model checking algorithms. A core component of these algorithms is a computation of the steady-state probabilities of the CTMC. This is reducible to the classical problem of solving a (homogeneous) sparse system of linear equations, of the form $\mathbf{Ax} = \mathbf{b}$, of size equal to the number of states in the CTMC.

A limitation of the Markovian modeling approach is the fact that the CTMC models tend to grow extremely large due to the state space explosion problem. This is caused by the fact that a system is usually composed of a number of concurrent subsystems, and that the size of the state space of the overall system is generally exponential in the number of subsystems. A realistic system can give rise to a large state space, typically over 10^6. As a consequence, much research is focused on the development of techniques, that is, methods and data structures, which minimise the computational (space and time) requirements for analysing large and complex systems.

One of such techniques is parallelization. Problems of parallel computations for such systems and finding its steady-state probabilities in parallel is brought up in [3], [8], [13], [15]. In [2], [4], [7], [17] a manner of distributed generation of the matrix $\mathbf{Q}$ in a network environment is desribed and in [6] a similar algorithm—on a computer cluster—is presented. Paper [5] describes solving sparse linear systems with the use of the GMRES algorithm [18] in a network environment. The parallel Jacobi method was discussed and a parallel method for the CTMC steady-state solution is presented in [15]. The Gauss-Seidel method is used for parallel solving of Markov chains in [14], [20].

In this paper a combination of two classical iterative methods for solving linear equation systems, namely Jacobi method and Gauss-Seidel method is presented. These methods were chosen because the presented algorithm is intended for computer clusters and Jacobi method is inherently parallel (Gauss-Seidel method has not got such a property and its parallelization requires a lot of communication), but Gauss-Seidel method usually converges much faster than Jacobi method [1]. Properties of such a combined method are experimantally examined in this paper. We try to study relative speedup and efficiency of the algorithm—as the traditional characteristics of parallel algorithms.

The rest of the paper is organized as follows. Section II presents the problem. In Section III classical iterative methods are reminded. Section IV shows used data distribution. Section V presents traditional block/parallel iterative methods and (in Section V-C) an approach employed by the authors. Section VI describes conducted experiments and Section VII contains some conclusions.

II. CTMCs and the Steady-State Solution

While modeling with Markov chains, in a steady state (independent of time), we obtain a linear equation system like follows;

$$\mathbf{Q}^T\mathbf{x} = \mathbf{0}, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{x}^T\mathbf{e} = 1 \tag{1}$$

where $\mathbf{Q}$ is a transition rate matrix, $\mathbf{x}$ is an unknown vector of states probabilities and $\mathbf{e} = (1, 1, ..., 1)^T$. The matrix $\mathbf{Q}$

This work was partially supported within the project *Metody i modele dla kontroli zatłoczenia i oceny efektywności mechanizmów jakości usług w Internecie następnej generacji* (N517 025 31/2997). This work was also partially supported by Marie Curie-Sklodowska University in Lublin within the project *Równoległe algorytmy generacji i rozwiązywania mechanizmów kontroli przeciążenia w protokole TCP modelowanych przy użyciu łańcuchów Markowa*.

is a square one of size $n \times n$, usually a big one, of rank $n-1$, sparse, with dominant diagonal. It is also a singular matrix demanding adequate methods to solve the equation. Markovian models solving demands overcoming both numerical and algorithmic problems. Solving the equation system (1) generally requires applying iterative methods, projection methods or decomposition methods but occasionally (for the need of an accurate solution) direct methods are used as well. The rich material concerning the methods mentioned above can be found in [19].

III. Iterative methods—Jacobi and Gauss-Seidel

In this section classical iterative methods are reminded. The general form of such a method step is

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} \tag{2}$$

where

$$\mathbf{M} - \mathbf{N} = \mathbf{Q}^T. \tag{3}$$

A. The Method of Jacobi

The method of Jacobi is a classical iterative method with the coefficient matrix $\mathbf{Q}^T$ split as following:

$$\mathbf{Q}^T = \mathbf{D} - (\mathbf{L} + \mathbf{U}) \tag{4}$$

which corresponds to assigning:

$$\mathbf{M} = \mathbf{D}, \qquad \mathbf{N} = \mathbf{L} + \mathbf{U} \tag{5}$$

in (3). The matrix $\mathbf{D}$ is a diagonal matrix, the matrix $\mathbf{L}$ is a strictly lower triangular matrix (with zeroes on its diagonal) and the matrix $\mathbf{U}$ is a strictly upper triangular matrix (with zeroes on its diagonal). So in this method the step (2) looks as following:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)} \tag{6}$$

and in scalar form (for $i = 1, \ldots, n$):

$$x_i^{(k+1)} \leftarrow \frac{1}{d_{ii}} \left(\sum_{j=1}^{i-1} l_{ij} x_j^{(k)} + \sum_{j=i+1}^{n} u_{ij} x_j^{(k)} \right). \tag{7}$$

The method of Jacobi is very convinient to vectorize and to parallelize. It can also be seen in the Jacobi equation (7) that the new approximation of the solution vector ($\mathbf{x}^{(k+1)}$) is calculated by using only the old approximation of the vector ($\mathbf{x}^{(k)}$). This method, therefore, possess a high degree of natural parallelism. However, Jacobi methods has relatively slow convergence.

B. The Method of Gauss-Seidel

In this method we have the same splitting of the matrix $\mathbf{Q}^T$ but with a different grouping of components:

$$\mathbf{Q}^T = (\mathbf{D} - \mathbf{L}) - \mathbf{U} \tag{8}$$

(the matrices $\mathbf{D}$, $\mathbf{L}$, $\mathbf{U}$ are defined as in the previous section). Here we have $\mathbf{M} = (\mathbf{D} - \mathbf{L})$ and $\mathbf{N} = \mathbf{U}$, so the step (2) is:

$$\mathbf{x}^{(k+1)} \leftarrow (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(k)} \tag{9}$$

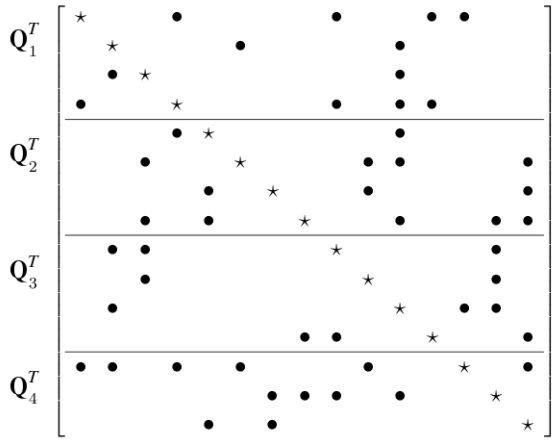

Fig. 1. An example of the matrix $\mathbf{Q}^T$ division

and in scalar form (for $i = 1, \ldots, n$):

$$x_i^{(k+1)} \leftarrow \frac{1}{d_{ii}} \left(\sum_{j=1}^{i-1} l_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} u_{ij} x_j^{(k)} \right). \tag{10}$$

The method of Gauss-Seidel is not so convinient for vectorization or parallelization, but it converges faster—what can be explained by the fact that although both methods use the same scalar formulas (7) and (10), but in the method of Gauss-Seidel the newly computed approximation of a solution component is used as soon as it is available.

The Gauss-Seidel method typically converges faster than the Jacobi method by using the most recently available approximations of the elements of the iteration vector. The other advantage of the Gauuss-Seidel algorithm is that it can be implemented using only one iteration vector, which is important for large linear equation systems where storage of single iteration vector alone may require 10GB or more. However, a consequence of using the most recently available solution approximation is that the method is inherently sequential—it does not possess natural parallelism.

IV. Data Distribution

In parallel programming a very crucial issue is a division of data among computational nodes (machines, processors etc.). In the algorithm described later (in Section V-C) the matrix $\mathbf{Q}^T$ is divided among cluster nodes as in Figure 1—that is the matrix is divided into p rectangular submatrices $\mathbf{Q}_i^T$, each stored in the ith cluster node.

Each submatrix is a sparse matrix and takes part in the computations on its node (where it is stored).

V. A Parallel Method of Jacobi/Gauss-Seidel

The description of the parallel Jacobi/Gauss-Seidel method will be started from the presentation of the well-known block iterative methods, namely block Jacobi method and block Gauss-Seidel method.

Block-based formulations of the iterative methods which perform matrix computations on block-by-block basis usually

turn out to be more efficient and easier to parallelize. Generally iterative block methods demand more calculations per iteration, which is recompensed by a faster convergence rate (and sometimes better cache utilization).

It is possible in Markov chains to divide the transition rate matrix into blocks (or even generate it in blocks straight away [4], [7]) and develop iterative methods basing on that division.

In Markov chains problems it is often the case that the state space can be meaningfully partitioned into subsets and thus, it is possible to partition the transition rate matrix respectively and to base the solution method on blocks implied by such a partition.

The homogeneous equations system (1) is divided into K^2 square blocks of the same size in the following way:

$$\begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} & \dots & \mathbf{Q}_{1K} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} & \dots & \mathbf{Q}_{2K} \\ \dots & \dots & \dots & \dots \\ \mathbf{Q}_{K1} & \mathbf{Q}_{K2} & \dots & \mathbf{Q}_{KK} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_K \end{bmatrix} = \mathbf{0}. \tag{11}$$

We introduce block splitting:

$$\mathbf{Q}^T = \mathbf{D}_K - (\mathbf{L}_K + \mathbf{U}_K), \tag{12}$$

where $\mathbf{D}_K$ is a block-diagonal matrix, $\mathbf{L}_K$ is a strictly block lower triangular matrix, $\mathbf{U}_K$ is a strictly block upper triangular matrix with form:

$$\mathbf{D}_K = \begin{bmatrix} \mathbf{D}_{11} & 0 & \dots & 0 \\ 0 & \mathbf{D}_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \mathbf{D}_{KK} \end{bmatrix}, \tag{13}$$

$$\mathbf{L}_K = \begin{bmatrix} 0 & 0 & \dots & 0 \\ \mathbf{L}_{21} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \mathbf{L}_{K1} & \mathbf{L}_{K2} & \dots & 0 \end{bmatrix}, \tag{14}$$

$$\mathbf{U}_K = \begin{bmatrix} 0 & \mathbf{U}_{12} & \dots & \mathbf{U}_{1K} \\ 0 & 0 & \dots & \mathbf{U}_{2K} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}, \tag{15}$$

where $\mathbf{D}_{ii} = \mathbf{Q}_{ii}$, $\mathbf{L}_{ij} = -\mathbf{Q}_{ij}$, $\mathbf{U}_{ij} = -\mathbf{Q}_{ij}$.

A. A Block Jacobi Algorithm

The iterative method of Jacobi was described in Section III-A. In this section, a block parallel Jacobi algorithm for the solution of the linear equation system (1) is presented.

Block Jacobi method is given by (for $i = 1, \dots, K$):

$$\mathbf{Q}_{ii}^T \mathbf{x}_i^{(k+1)} = -\sum_{j<>i} \mathbf{Q}_{ij}^T \mathbf{x}_j^{(k)} \tag{16}$$

where blocks are as in (11).

Having K computational nodes (computers or processors), every equation of (16) can be solved independently by a computational node. The equtions (16) can be solved by an arbitrary method.

B. A Block Gauss-Seidel Algorithm

Just like the scalar Gauss-Seidel algorithm (see Section III-B), the block Gauss-Seidel algorithm can be written:

$$(\mathbf{D}_K - \mathbf{L}_K) x^{(i+1)} = \mathbf{U}_K x^{(i)}. \tag{17}$$

Describing the equation mentioned above in a scalar-like form we get (for $j = 1, \dots, K$):

$$\mathbf{Q}_{jj} \mathbf{x}_j^{(i+1)} = -\left(\sum_{l=1}^{i-1} \mathbf{Q}_{jl} \mathbf{x}_l^{(i+1)} + \sum_{l=j+1}^{K} \mathbf{Q}_{jl} \mathbf{x}_l^{(i)} \right) \tag{18}$$

where $\mathbf{x}_j^{(i)}$ is the jth (n/K)-element subvector of the vector $\mathbf{x}^{(i)}$ (as in (11)).

As a result of above in every step we must solve K equation systems of n/K size each in the following form (for $j = 1, \dots, K$):

$$\mathbf{Q}_{jj} \mathbf{x}_i^{(i+1)} = \mathbf{z}_j^{(i+1)} \tag{19}$$

where

$$\mathbf{z}_j^{(i+1)} = -\left(\sum_{l=1}^{i-1} \mathbf{Q}_{jl} \mathbf{x}_l^{(i+1)} + \sum_{l=j+1}^{K} \mathbf{Q}_{jl} \mathbf{x}_l^{(i)} \right). \tag{20}$$

We can apply different direct and iterative methods to the solve equation (19). There is a small number of iterations demanded to obtain convergence for a small number of blocks (i.e. submatrices are big). The difficulty is that it is very hard to parallelize effectively—it is caused by the use of the newly computed values just after their computation.

C. A Modified Block Jacobi Algorithm

In this section we present an algorithm, which takes advantage of the division of the matrix $\mathbf{Q}$ between computational nodes described in Section IV. The algorithm proposed here is a combination of Jacobi and Gauss-Seidel iterative methods.

To start from a usual Jacobi method, there is

$$\mathbf{Q}^T = \mathbf{D} - (\mathbf{L} + \mathbf{U}) \tag{21}$$

(see Section III-A).

Let

$$\mathbf{H} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}). \tag{22}$$

Thus, the (6) can be written

$$\mathbf{x}^{(k+1)} = \mathbf{H}\mathbf{x}^{(k)}. \tag{23}$$

In a block iterative method the linear system (23) is divided into some subsystems. In this section the matrix $\mathbf{H}$ is divided into p blocks (p is the number of computatinal nodes), each block $\mathbf{H}_i$ $(i = 1, \dots, p)$ of the size of n columns and n/p rows (last one, $\mathbf{H}_p$ can be shorter—as in Figure 1—it does not influences general considerations). Such a division corresponds to the division of the matrix $\mathbf{Q}^T$ proposed in Section IV, because to obtain elements of the matrix $\mathbf{H}_i$ one needs only elements of the matrix $\mathbf{Q}_i^T$.

Similarly, the vector $\mathbf{x}$ (and some auxiliary vectors in the implementation) is divided into p subvectors, each of the size n/p.

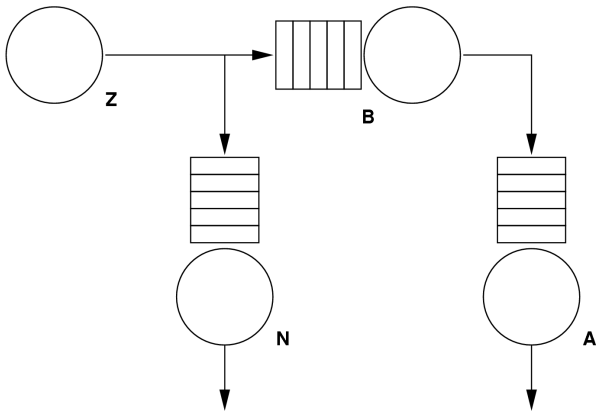

Fig. 2. A Markovian queuing model of the tail-drop mechanism

Now, the (23) can be written:

$$\begin{bmatrix} \mathbf{x}_1^{(k+1)} \\ \mathbf{x}_2^{(k+1)} \\ \vdots \\ \mathbf{x}_p^{(k+1)} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_p \end{bmatrix} \mathbf{x}^{(k)} \tag{24}$$

and, in another manner, for $i = 1, \ldots, p$:

$$\mathbf{x}_i^{(k+1)} = \mathbf{H}_i \mathbf{x}^{(k)}. \tag{25}$$

(25) is a formula for the block Jacobi method (Section III-A). All the equations in (25) are solved independently, so it is very suitable to parallelize for p processors – each of them solves one equation and then they exchange the resulting vectors $\mathbf{x}_i^{(k+1)}$ to build its new version $\mathbf{x}^{(k+1)}$.

However, the Jacobi method is rather slowly convergent, so in the presented algorithm, every computational node can employ the formula of the Gauss-Seidel algorithm (10) instead of the Jacobi algorithm (7)—using the newly obtained elements of the vector $\mathbf{x}_i^{(k+1)}$ (although only those which are stored in the same computational nodes) and hoping for the better convergency.

In other words, the algorithm can be described as a block Jacobi iterative method with solving inner blocks with the Gauss-Seidel iterative method.

In borderline cases the presented algorithm reduces to the pure Jacobi algorithm (for $p = n$) and to the pure Gauss-Seidel algorithm (for $p = 1$).

D. Implementation details

We propose an algorithm for homogeneous cluster environments. This algorithm is based on a message-passing paradigm and consists of one module for each of nodes. The algorithm presented below is just a skeleton, and the detailed implementation, such as data preparation, parameters passing, and so forth, might be different according to requirements of various applications.

The algorithm is composed of several steps. First, the starting information is acquired. Next, the matrix $\mathbf{Q}$ is generated in parts so that every node keeps only needed states (as in Figure 1) Next, in loop, in every node for its block we make a step of the Gauss-Seidel method.

The algorithm for every node (in pseudo-code) is described as follows:

```
Initialization(&MyNumber);

FindBeginAndEnd(MyNumber, &n0, &n1);
  /* computes starting index n0 and
     ending index n1 for the block of
     the node MyNumber */
QT=Generate(n0, n1);
  /* generates adequate block */
FillVector(X, 1.0/n);
IterNo=0;

do
{
  IterNo++;
  InnerGaussSeidel(QT, X);
    /* the node computes only its own
       part of the vector */
  GatherVector(X)
    /* every node needes the whole vector
       for further computations */
  Normalize(X);
} while(Remainder(QT, X)>EPSILON &&
        IterNo<MAXITER);

Finalizetion();
```

VI. Experimental Results

A. Properties of the Used Matrices

The matrices obtained for tests were transition rate matrices for a very simple model of a tail-drop mechanism (a very simple congestion control mechanism in a router's buffer). The queuing model for such a mechanism is shown in Figure 2. It is a passive mechanism, that is, it does not make any decision, as far as the moment of dropping packets is concerned—they are always dropped when there is no room for new packets. Moreover, in the tail-drop mechanism the packet that just arrived, is dropped. A Markovian queuing model for such a mechanism consists of: a source **Z** (which sends packets with variable rates), a service station **B** (which corresponds to a router's buffer) and two auxiliary service stations **A** and **N** (which correspond to confirmations and rejections, respectively, returning to the source **Z**).

The rate of the source **Z** is not constant but it increases (not above a given maximum) as confirmations leave the station **A** and it decreases (not below a given minimum) as rejections leave the station **N** (both events represent reaching the source by the information about the packet's fate). The states of such a model can be written as vectors of numbers. In our example it would be (l, b, a, m), where l is an integer between 1 and l_{max} and it means the current relative intensity of the source **Z** and b, a and m are integers between 0 and b_{max}, a_{max},

TABLE I
TEST MATRICES AND THE ALGORITHM PERFORMANCE

l_{max}	b_{max}	a_{max}	m_{max}	n	p	T_p [s]	S_p	E_p
40	40	40	40	2 756 840	1	145	—	—
					10	96	1.51	0.15
50	50	50	50	6 632 550	1	480	—	—
					10	210	2.29	0.23
60	60	60	60	13 618 860	1	812	—	—
					10	399	2.04	0.20
70	70	70	70	25 053 770	1	3 373	—	—
					10	714	4.72	0.47
72	72	72	72	28 009 224	1	4 283	—	—
					10	829	5.17	0.52
74	74	74	74	31 218 750	1	5 232	—	—
					10	929	5.63	0.56

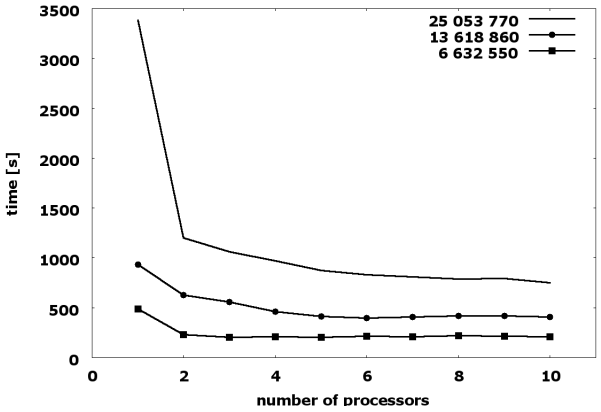

Fig. 3. Performance time of the algorithm for three different sizes of matrices

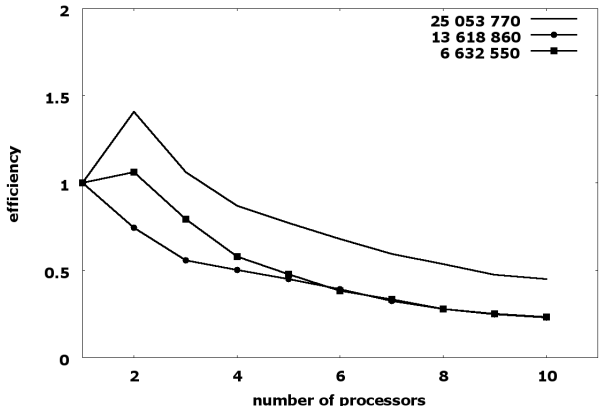

Fig. 5. Efficiency of the algorithm for three different sizes of matrices

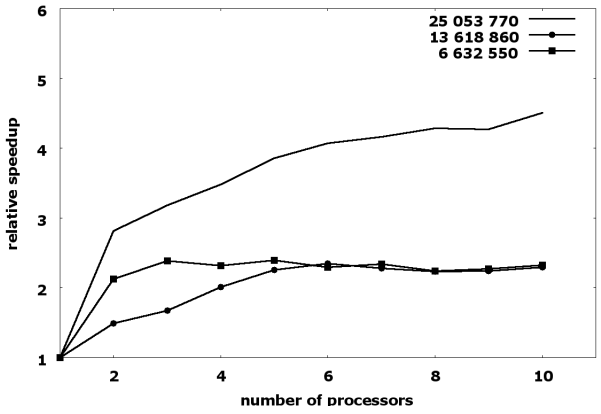

Fig. 4. Relative speedup of the algorithm for three different sizes of matrices

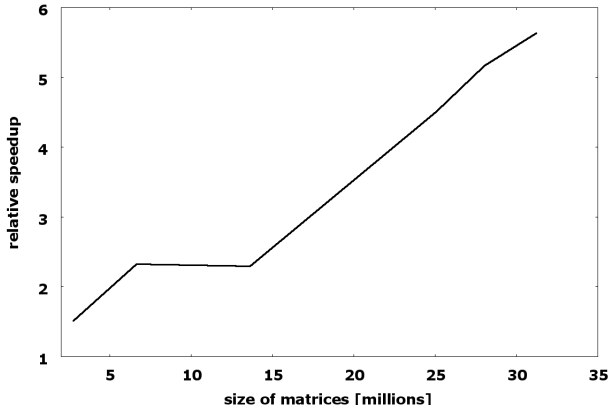

Fig. 6. Relative speedup of the algorithm working on 10 processors for various matrices

m_{max} (respectively) and they mean the number of packets waiting or being processed currently in the stations **B**, **A**, **N** (respectively).

B. Computational Environment

Experiments were carried out in a cluster environment (part of CLUSTERIX – a cluster of Linux machines consisting of more than 800 machines distributed all over Poland and connected with a fast optical network; one of the fastest European distributed supercomputers [21]) dedicated for computing.

The CLUSTERIX itself is built as a cluster of local clusters. So up to 10 machines were used—all belonging to a single local cluster— as we were interested in investigating behavior

of the algorithm in such an environment. Each of the machines was equipped with two 64-bit processors Intel Itanium 2 1.4 GHz and 4 GB RAM, but we used only one processor in every machine, because we were interested in distributed results. The cluster nodes were used when they were idle—in order to assure credible performance times.

The algorithm was implemented with the use of the MPI (message passing interface) standard and MPICH library.

C. Numerical Results

The properties of the models and the matrices used in tests are shown in Table I. All the matrices have about 5–6 elements in a row (very sparse matrices). There are also some characteristics of the performance—T_p, S_p and E_p presented there. The distributed run-time of 50 iterations of the algorithm is measured as the maximum run-time from the start and we denote it as T_p (for p machines). The speedup for p processors is denoted by S_p and is given by $S_p = T_1/T_p$. The efficiency for p processors is denoted by E_p and is defined as $E_p = S_p/p$.

We can see in Figures 3–6 and in Table I that the relative speedup and the efficiency grows with the growth of the size of matrices, so we can expect that for bigger matrices we can get better results.

VII. Conclusion

The authors have developed a parallel and somewhat scalable algorithm that computes the vector $\mathbf{x}$ from the equation (1) for a very large n; the matrix $\mathbf{Q}$ is distributed among the computational nodes. To find $\mathbf{x}$, a parallel combination of Jacobi and Gauss-Seidel iterative methods was used. The numerical experiments on a parallel system have been carried out in order to assess the effectiveness of the distributed algorithm on a computer cluster. The numerical tests indicate that some efficiency is possible if the sufficient amount of work per processor is provided (for small sizes of the matrix the scalability is worse). The parallel implementation was benchmarked using a Markovian model of congestion control.

The results suggest an important area for future research—writing numerical algorithms (which find probability vectors) for huge matrices modeling Markov chains distributed in row-striped manner among many processor (as in our algorithm) and performance optimization—perhaps using preconditioned iterative methods [2], [9], [10], [11], [12], [17].

The proposed method is tested on matries connected with Markov chains, but we can use this method to different application to other problems with sparse matrices.

In the future we want to examine the combination of Jacobi and Gauss-Seidel methods theoretically and compare our approach with others.

References

[1] J. M. Bahi, S. Contessot-Vivier, R. Coutier: Parallel iterative algorithms. From sequential to Grid Computing. Chapman & Hall/CRC, 2007.

[2] M. Benzi, M. Tuma: A parallel solver for large-scale Markov chains. *Applied Numerical Mathematics*, 41 (2002), pp. 135–153.

[3] P. Buchholz, M. Fischer, P. Kemper: Distrbuted steady state analysis using Kronecker algebra. *Proceedings of the Third International Conference on the Numerical Solution of Markov Chains (NSNC '99)*, Zaragoza, Spain, September 1999, pp. 76–95.

[4] J. Bylina: A distributed approach to solve large Markov chains. *Proceedings from EuroNGI Workshop: New Trends in Modeling, Quantitive Methods and Measurements*, Jacek Skalmierski Computer Studio, Gliwice 2004, pp. 145–154.

[5] J. Bylina, B. Bylina: A Markovian model of the RED mechanism solved with a cluster of computers. *Annales UMCS Informatica*, 5 (2006), pp. 19–27.

[6] J. Bylina, B. Bylina: Analysis of a Parallel Algorithm of Distributed of Matrix for Markovian Models of Congestion Control. Wydawnictwa Komunikacji i Łączności, Warszawa 2008,

[7] J. Bylina, B. Bylina: Distributed generation of Markov chains infinitesimal generators with the use of the low level network interface. *Proceedings of 4th International Conference Aplimat 2005*, part II, pp. 257–262.

[8] N. J. Dingle, P. G. Harrison, W. J. Knottenbelt: Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models. *Journal of Parallel and Distributed Computing*, 64 (2004) pp. 908–920.

[9] K. M. Giannoutakis, G. A. Gravvanis, B. Clayton, A. Patil, T. Enright, J. P. Morrison: Matching high performance approximate inverse preconditioning to architectural platforms, *The Journal of Supercomputing*, 42(2), 2007, pp. 145–163.

[10] G. A. Gravvanis: Explicit Approximate Inverse Preconditioning Techniques, *Archives of Computational Methods in Engineering*, 9(4), 2002, pp. 371–402.

[11] G. A. Gravvanis: Explicit preconditioned generalized domain decomposition methods, *International Journal of Applied Mathematics*, 4(1), 2000, pp. 57–71.

[12] G. A. Gravvanis, V. N. Epitropou, K. M. Giannoutakis: On the performance of parallel approximate inverse preconditioning using Java multithreading techniques, *Applied Mathematics and Computation*, Vol. 190, 2007, pp. 255–270.

[13] W. Knottenbelt: Distributed disk-based solution techniques for large Markov models. *Proceedings of the Third International Conference on the Numerical Solution of Markov Chains (NSNC '99)*, Zaragoza, Spain, September 1999.

[14] M. Kwiatkowska, D. Parker, Y. Zhang, R. Mehmood: Dual-processor parallelisation of symbolic probabilistic model checking. In *Proceedings of 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, 2004, pp. 123–130.

[15] R. Mehmood, J. Crowcroft: Parallel Iterative Solution Method for Large Sparse Linear Equation Systems Technical Report UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, UK October 2005.

[16] Ng Chee Hock: Queuing Modelling Fundamentals. Wiley, New York, 1996.

[17] A. Platis, G. Gravvanis, K. Giannoutakis, E. Lipitakis: A two-phase cyclic non-homogeneous markov chain performability evaluation by explicit approximate inverses applied to a replicated database system, *Journal of Mathematical Modelling and Algorithms*, 2, 2003, pp. 235–249.

[18] Y. Saad, M. H. Schultz: GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7, 1986, pp. 856–869.

[19] W. Stewart: Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Chichester, West Sussex 1994.

[20] Y. Zhang, D. Parker, M. Kwiatkoska: A wavefront parallelisation of CTMC solution using MTBDDs. In *Proceedings of International Conference on Dependable Systems and Networks (DSN'05)*, IEEE Computer Society Presss, 2005, pp. 732–742.

[21] http://clusterix.pcz.pl/