DOI: 10.14529/jsfi180202

Scalability Evaluation of Cimmino Algorithm for Solving Linear Inequality Systems on Multiprocessors with Distributed Memory*

Irina M. Sokolinskaya[1], *Leonid B. Sokolinsky*[1]

The paper is devoted to a scalability study of Cimmino algorithm for linear inequality systems. This algorithm belongs to the class of iterative projection algorithms. For the analytical analysis of the scalability, the BSF (Bulk Synchronous Farm) parallel computation model is used. An implementation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* is presented. An analytical estimation of the upper scalability bound of the algorithm for cluster computing systems is derived. An information about the implementation of Cimmino algorithm on lists in C++ language using the BSF program skeleton and MPI parallel programming library is given. The results of large-scale computational experiments performed on a cluster computing system are demonstrated. A conclusion about the adequacy of the analytical estimations by comparing them with the results of computational experiments is made.

Keywords: system of linear inequalities, iterative algorithm, projection algorithm, Cimmino algorithm, parallel computation model, bulk synchronous farm, scalability estimation, speedup, parallel efficiency, cluster computing systems.

Introduction

The problem of solving systems of linear inequalities arise in numerous fields. As examples, we can mention linear programming [1, 2], image reconstruction from projections [3], image processing in magnetic resonance imaging [4], intensity-modulated radiation therapy (IMRT) [5]. At the present time, a lot of methods for solving systems of linear inequalities are known, among which we can mark out a class of self-correcting iteration methods that allow efficient parallelization. In this field, pioneer works are papers [6, 7], in which the Agmon–Motzkin–Schoenberg relaxation method for solving systems of linear inequalities was proposed. The relaxation method belongs to the class of projection methods, which use the operation of orthogonal projection onto a hyperplane in Euclidean space. One of the first iterative algorithms of projection type was the Cimmino algorithm [8], intended for solving systems of linear equations and inequalities. Cimmino algorithm had a great influence on the development of computational mathematics [9]. A considerable number of papers have been devoted to the generalizations and extensions of the Cimmino algorithm (for example, see [3, 10–13]).

In many cases, systems of linear inequalities arising in the solution of practical problems can involve up to tens of millions of inequalities and up to hundreds of millions of variables [2]. In this case, the issue of developing scalable parallel algorithms for solving large-scale systems of linear inequalities on multiprocessor systems with distributed memory becomes very urgent. When one creates parallel algorithms for large multiprocessor systems, it is important at an early stage of the algorithm design (before coding) to obtain an analytical estimation of its scalability. For this purpose, one can use various models of parallel computation [14]. Nowadays, a large number of different parallel computation models are known. The most famous models among

*The article is recommended for publication by the Program Committee of the International Scientific Conference "Russian Supercomputing Days 2018".

[1]South Ural State University, Chelyabinsk, Russian Federation

them are PRAM [15], BSP [16] and LogP [17]. Each of these models generated a large family of parallel computation models, which extend and generalize the parent model (see, e.g., [18–20]). The problem of developing new parallel computation models is still important today. The reason is that it is impossible to create a parallel computation model, which is good in all respects. To create a good parallel computation model, the designer must restrict the set of target multiprocessor architectures and class of algorithms. In paper [21], the parallel computation model BSF (Bulk Synchronous Farm) intended for cluster computing systems and iterative algorithms was proposed. The BSF model makes it possible to predict the upper scalability bound of an iterative algorithm with great accuracy before coding. An example of using the BSF model is given in [22].

The purpose of this article is to investigate the scalability of the Cimmino algorithm for solving large-scale systems of linear inequalities on multiprocessor systems with distributed memory by using the BSF parallel computation model. The rest of the article is organized as follows. Section 1 gives a formal description of the Cimmino algorithm. In Section 2, the representation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* defined in the Bird–Meertens formalism is constructed. Section 3 is dedicated to an analytical investigation of the scalability of the Cimmino algorithm on lists using the BSF model cost metrics; the equations for estimating the speedup and parallel efficiency are given; the upper bound of the algorithm scalability depending on the problem size is calculated. In Section 4, a description of the implementation of the Cimmino algorithm on lists in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library is presented; a comparison of the results obtained analytically and experimentally is given. In conclusion, the obtained results are summarized and directions for further research are outlined.

1. Cimmino Algorithm for Inequalities

Let us consider the system of linear inequalities

$$l_i(x) = \langle a_i, x\rangle - b_i \leqslant 0 \qquad (i = 1, \ldots, m), \tag{1}$$

where $\langle a_i, x\rangle$ is the Euclidean inner product of a_i and x in $\mathbb{R}^n$, $b_i \in \mathbb{R}$. To avoid triviality, we assume $m \geqslant 2$. We also assume that the system (1) is consistent. It is necessary to find a solution of the system of linear inequalities (1). To solve this problem, it is convenient to use a geometric language. Thus, we look upon $x = (x_1, \ldots, x_n)$ as a point in n-dimensional Euclidean space $\mathbb{R}^n$, and each inequality $l_i(x) \leqslant 0$ as a half-space P_i. Therefore, the set of solutions of system (1) is the convex polytope $M = \bigcap_{i=1}^{m} P_i$. Each equation $l_i(x) = 0$ defines a hyperplane H_i:

$$H_i = \{x \in \mathbb{R}^n | \, \langle a_i, x\rangle = b_i\}\,. \tag{2}$$

Let the orthogonal projection of $x \in \mathbb{R}^n$ onto the hyperplane $H_i \subset \mathbb{R}^n$ be denoted by $\pi_{H_i}(x)$. The orthogonal projection $\pi_{H_i}(x)$ can be calculated by the following equation:

$$\pi_{H_i}(x) = x + \frac{b_i - \langle a_i, x\rangle}{\|a_i\|^2} a_i, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm. Let us define the orthogonal reflection of x with respect to hyperplane H_i as follows:

$$\rho_{H_i}(x) = \pi_{H_i}(x) - x = \frac{b_i - \langle a_i, x\rangle}{\|a_i\|^2} a_i. \tag{4}$$

The *Cimmino algorithm for equally weighted inequalities* consists of the following steps:

Step 1: $k := 0$; $x_0 := \mathbf{0}$.

Step 2: $x_{k+1} := x_k + \frac{\lambda}{m} \sum_{i=1}^{m} \rho_{H_i}(x_k)$.

Step 3: If $\|x_{k+1} - x_k\|^2 < \varepsilon$ then go to Step 5.

Step 4: $k := k + 1$; go to Step 2.

Step 5: Stop.

Cimmino's method starts with an arbitrary point x_0 in $\mathbb{R}^n$ as an initial approximation, and then calculates at each step the centroid of a system of masses placed at the reflections of the previous iterate with respect to the hyperplanes $H_1, \ldots, H_m$ defined by the system of inequalities. This centroid is taken as the new iterate:

$$x_{k+1} = x_k + \frac{\lambda}{m} \sum_{i=1}^{m} \rho_{H_i}(x_k). \tag{5}$$

In equation (5), λ is a *relaxation parameter*. It is known [10] that for $0 < \lambda < 2$ the iteration process (5) converges to a point belonging to the polytope M.

2. Cimmino Algorithm in the Form of Operations on Lists

In order to obtain analytical estimations of an algorithm using the cost metrics of the BSF model, it must be represented in the form of operations on lists using higher-order functions *Map* and *Reduce* defined in the Bird-Meertens formalism [23]. The higher-order function *Map* applies the given function $F : \mathbb{A} \to \mathbb{B}$ to each element of the given list $[a_1, \ldots, a_m]$ and returns a list of results in the same order:

$$Map(F, [a_1, \ldots, a_m]) = [F(a_1), \ldots, F(a_m)]. \tag{6}$$

The higher-order function *Reduce* reduces the given list $[b_1, \ldots, b_m]$ to a single value by iteratively applying the given binary associative operation $\oplus : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ to each pair of elements:

$$Reduce(\oplus, [b_1, \ldots, b_m]) = b_1 \oplus \ldots \oplus b_m. \tag{7}$$

In the context of the Cimmino algorithm, we define the list L_{Map} as follows:

$$L_{Map} = [i_1, \ldots, i_m], \tag{8}$$

where $i_k \in \{1, \ldots, m\}$ and $i_k \neq i_l$ for $k \neq l$ $(k, l = 1, \ldots, m)$. In other words, L_{Map} – is the list of numbers of inequalities (1) ordered in an arbitrary way. For an arbitrary point $x \in \mathbb{R}^n$, let us define the function $F_x : \{1, \ldots, m\} \to \mathbb{R}^n$ as follows:

$$F_x(i) = \rho_{H_i}(x) \tag{9}$$

for all $i \in \{1, \ldots, m\}$. In other words, the function $F_x(i)$ calculates the orthogonal reflection of x with respect to the hyperplane H_i. For an arbitrary point $x \in \mathbb{R}^n$, let us define the list $L^{(x)}_{Reduce} \subset \mathbb{R}^n$ as follows:

$$L^{(x)}_{Reduce} = [F_x(i_1), \ldots, F_x(i_m)]. \tag{10}$$

The list $L^{(x)}_{Reduce}$ holds orthogonal reflections of the point x with respect to the hyperplanes $H_1, \ldots, H_m$ in the order determined by the list L_{Map}. Thus, the list $L^{(x)}_{Reduce}$ is obtained from the list L_{Map} by applying to it the higher-order function Map using as a parameter the function F_x:

$$L^{(x)}_{Reduce} = Map(F_x, L_{Map}). \tag{11}$$

Let us define the binary associative operation $\oplus : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ as follows:

$$x \oplus y = x + y \tag{12}$$

for all $x, y \in \mathbb{R}^n$. In this case, the $\oplus$ operator performs the conventional composition of vectors. Then the sum of orthogonal reflections of the point x can be obtained by applying to the list $L^{(x)}_{Reduce}$ the higher-order function *Reduce* using as a parameter the vector composition operation $\oplus$:

$$\sum_{i=1}^{m} \rho_{H_i}(x) = Reduce(\oplus, L^{(x)}_{Reduce}). \tag{13}$$

Now we can write the *Cimmino algorithm in the form of operations on lists*:

Step 1: $k := 0$; $x_0 := \mathbf{0}$; $L_{Map} := [1, \ldots, m]$.

Step 2: $L^{(x_k)}_{Reduce} := Map(F_{x_k}, L_{Map})$.

Step 3: $s := Reduce(\oplus, L^{(x_k)}_{Reduce})$.

Step 4: $x_{k+1} := x_k + \frac{\lambda}{m} s$

Step 5: If $\|x_{k+1} - x_k\|^2 < \varepsilon$ then go to Step 7.

Step 6: $k := k + 1$; go to Step 2.

Step 7: Stop.

The BSF model assumes that the algorithm is executed by a computing system consisting of one master-node and K worker-nodes ($K > 0$). Step 1 of the algorithm is performed by both the master and the workers during the initialization of the iterative process. Step 2 (*Map*) is performed only on the worker-nodes. Step 3 (*Reduce*) is performed on the worker-nodes and partially on the master-node. Steps 4–6 are performed only on the master-node. The BSF model assumes that all arithmetic operations (addition and multiplication) as well as comparison operations on floating-point numbers take the same time τ_{op}.

3. Analytical Evaluation of Scalability

Let us introduce the following notation for the scalability evaluation of the Cimmino algorithm:

c_s : the quantity of float numbers transferred from the master to one worker;

c_{Map} : the quantity of arithmetic operations performed in the *Map* step (Step 2 of the algorithm);

c_{Reduce_W} : the quantity of arithmetic operations performed by one worker in the *Reduce* step (Step 3 of the algorithm);

c_r : the quantity of float numbers transferred from one worker to the master;

c_{Reduce_M} : the quantity of arithmetic operations performed by the master in the *Reduce* step (Step 3 of the algorithm);

c_a : the quantity of arithmetic operations performed by the master in Steps 4 and 5 of the algorithm (aggregation).

Let us calculate the indicated values. At the beginning of each iteration, the master sends to all the workers the current approximation x_k, which is a vector of dimension n. Hence:

$$c_s = n. \tag{14}$$

Let us calculate the number of arithmetic operations performed in the *Map* step. For each element of the list L_{Map}, one vector is calculated by equation (4). Note that the values of $\|a_i\|^2$ $(i = 1, \ldots, m)$ do not depend on x_k, and therefore can be calculated in advance at the initialization stage. Taking this into account, the quantity of operations for calculating one orthogonal reflection of the point x_k is $n^2 + 2$. Multiplying this number by the number of inequalities, we obtain

$$c_{Map} = m(n^2 + 2). \tag{15}$$

During the execution of *Reduce* step, the list L_{Reduce} consisting of m vectors is divided into equal parts, each of them assigned to a single worker. Everywhere below we assume that $K \leqslant m$. For simplicity we assume that m is a multiple of number of workers K. In that case, each worker has to process a list of length of m/K. The composition of vectors of dimension n requires x arithmetic operations. The composition of m/K vectors of dimension n requires $((m/K) - 1)n$ arithmetic operations. Hence:

$$c_{Reduce\_W} = ((m/K) - 1)n. \tag{16}$$

After execution of *Reduce* step, each worker sends the resulting vector to the master. Thus:

$$c_r = n. \tag{17}$$

The master has to perform the *Reduce* step for the vectors received from the workers. So, it has to compose K vectors of dimension n. Hence:

$$c_{Reduce\_M} = (K - 1)n. \tag{18}$$

The execution of Step 4 requires $2n$ operations (we assume the constant value of λ/m to be computed in advance). The execution of Step 5 requires $3n - 1$ arithmetic operations and one comparison operation. It follows the equation:

$$c_a = 5n. \tag{19}$$

Let us designate the time spent by the worker to perform one arithmetic operation as τ_{op}, and designate the time spent for transferring a single float number across the network excluding latency as τ_{tr}. In that way, we get the following values for the cost parameters of the BSF model [21] in the case of the Cimmino algorithm:

$$t_s = \tau_{tr} n; \tag{20}$$

$$t_w = \tau_{op} \left(m(n^2 + 2) + (m - K)n \right); \tag{21}$$

$$t_r = \tau_{tr} n; \tag{22}$$

$$t_p = \tau_{op} \left((K - 1)n + 5n \right). \tag{23}$$

Equation (20), obtained on the basis of (14), gives an estimation of the time t_s spent by the master to transfer a message to one worker excluding latency. Equation (21) is obtained using equations (15) and (16). According to the BSF model cost metric, t_w denotes the total time spent by the workers for computations on local data. Therefore, the expression on the right-hand side of equation (16) when substituted into equation (21) is multiplied by the number of workers K. Equation (22), obtained on the basis of (17), gives an estimation of the time t_r spent by the master to transfer a message to one worker excluding latency. Equation (23) obtained using equations (18) and (19) calculates the time t_p spent by the master on the following actions: performing its part of the *Reduce* step, calculating the next approximation and checking of stopping criterion. In accordance with this metric, the time for solving the problem by a system consisting of one master and one worker ($K = 1$) can be estimated as follows:

$$T_1 = 2(L + \tau_{tr} n) + \tau_{op} \left(m(n^2 + 2) + (m - 1)n + 5n \right). \tag{24}$$

The time of solving the problem by a system composed of one master and K workers can be estimated by the following equation:

$$T_K = 2(L + \tau_{tr} n)K + \tau_{op} \left(\frac{m(n^2 + n + 2)}{K} + Kn + 3n \right). \tag{25}$$

On the basis of equations (24) and (25) we can write the equation for speedup a in the form of a function of K:

$$a(K) = \frac{T_1}{T_K} = \frac{2(L + \tau_{tr} n) + \tau_{op} \left(m(n^2 + n + 2) + 4n \right)}{2(L + \tau_{tr} n)K + \tau_{op} \left(\frac{m(n^2+n+2)}{K} + Kn + 3n \right)}. \tag{26}$$

To determine the upper bound for the scalability of the Cimmino algorithm in accordance with the procedure described in [21], let us deduce the derivative $a'(K)$ and solve the equation

$$a'(K) = 0. \tag{27}$$

Using simple algebraic transformations, from equation (26), we can deduce the following equation for the derivative of speedup:

$$a'(K) = \big(2(L+\tau_{tr}n) + \tau_{op}\cdot\big(m(n^2+n+2)+4n\big)\big)\cdot \\ \cdot\frac{\tau_{op}\cdot\left(\frac{m(n^2+n+2)}{K^2}-n\right)-2(L+\tau_{tr}n)}{\left(2(L+\tau_{tr}n)K+\tau_{op}\cdot\left(\frac{m(n^2+n+2)}{K}+Kn+3n\right)\right)^2}. \tag{28}$$

Let us solve the equation

$$\big(2(L+\tau_{tr}n) + \tau_{op}\cdot\big(m(n^2+n+2)+4n\big)\big)\cdot \\ \cdot\frac{\tau_{op}\cdot\left(\frac{m(n^2+n+2)}{K^2}-n\right)-2(L+\tau_{tr}n)}{\left(2(L+\tau_{tr}n)K+\tau_{op}\cdot\left(\frac{m(n^2+n+2)}{K}+Kn+3n\right)\right)^2} = 0. \tag{29}$$

Dividing both sides of equation (29) by the positive quantity

$$\big(2(L+\tau_{tr}n) + \tau_{op}\big(m(n^2+n+2)+4n\big)\big)$$

and multiplying by the positive quantity

$$\left(2(L+\tau_{tr}n)K+\tau_{op}\left(\frac{m(n^2+n+2)}{K}+Kn+3n\right)\right)^2,$$

we obtain the equation

$$\tau_{op}\left(\frac{m(n^2+n+2)}{K^2}-n\right)-2(L+\tau_{tr}n)=0,$$

which implies

$$K=\sqrt{\frac{m(n^2+n+2)\tau_{op}}{2(L+\tau_{tr}n)+\tau_{op}n}}.$$

Thus, equation (27) for $K>0$ has the only root $K_0=\sqrt{\frac{m(n^2+n+2)\tau_{op}}{2(L+\tau_{tr}n)+\tau_{op}n}}$. We are interested in the case when $K_0>1$. This is the case when the condition $m(n^2+n+2)\tau_{op} > 2(L+\tau_{tr}n)+\tau_{op}n$ is satisfied. In accordance with (28), we have

$$a'(1)=\frac{\tau_{op}\big(m(n^2+n+2)-n\big)-2(L+\tau_{tr}n)}{2(L+\tau_{tr}n)+\tau_{op}\left(m(n^2+n+2)+4n\right)}>0. \tag{30}$$

It follows that the maximum of speedup is obtained at the point K_0. Thus, in accordance with the BSF model, the upper bound K_{max} of the scalability of the Cimmino algorithm is determined by the following equation:

$$K_{max}=\sqrt{\frac{m(n^2+n+2)\tau_{op}}{2(L+\tau_{tr}n)+\tau_{op}n}}. \tag{31}$$

Let us simplify equation (31). For $n, m \to \infty$, we have

$$m(n^2+n+2)\tau_{op}=O(mn^2) \tag{32}$$

and

$$2(L+\tau_{tr}n)+\tau_{op}n=O(n). \tag{33}$$

Substituting the right-hand sides of equations (32) and (33) into (31), we obtain

$$K_{max} = \sqrt{\frac{O(mn^2)}{O(n)}},$$

which is equivalent to

$$K_{max} = \sqrt{O(mn)}.$$

If we assume that $m > n$ (the number of inequalities is greater than the number of variables), then, in this case, we have $K_{max} = \sqrt{O(n^2)}$, which is equivalent to

$$K_{max} = O(n). \tag{34}$$

In that way, the upper bound of the scalability of the Cimmino algorithm on lists increases in proportion to the dimension of the problem n. In conclusion of this section, let us write the equation for estimating the parallel efficiency e as a function of K. Considering equation (26), we have

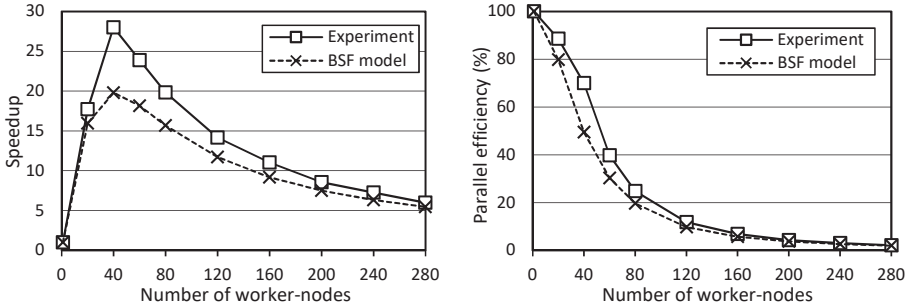
$$e(K) = \frac{a(K)}{K} = \frac{2(L + \tau_{tr}n) + \tau_{op}\left(m(n^2 + n + 2) + 4n\right)}{2(L + \tau_{tr}n)K^2 + \tau_{op}\left(m(n^2 + n + 2) + K^2n + 3Kn\right)}. \tag{35}$$

4. Numerical Experiments

In order to verify the analytical results, we implemented the Cimmino algorithm in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library. The source code of this program is freely available on Github, at `https://github.com/leonid-sokolinsky/BSF-Cimmino`. The system of inequalities was taken from the model scalable linear-programming problem *Model-n* given in [24]. In this system, the number of inequalities is $m = 2n + 2$, where n is the dimension of the space. We investigated the speedup and parallel efficiency of the Cimmino algorithm on the supercomputer "Tornado SUSU" [25]. The calculations were performed for the dimensions 1 500, 5 000, 10 000 and 16 000. At the same time, we plotted the curves of speedup and parallel efficiency for these dimensions using equations (26) and (35). For this, the following values in seconds were determined experimentally: $L = 1.5 \cdot 10^{-5}$, $\tau_{op} = 2.9 \cdot 10^{-8}$ and $\tau_{tr} = 1.9 \cdot 10^{-7}$. The results are presented in Fig. 1–4. In all cases, the analytical estimations were very close to experimental ones. Moreover, the performed experiments show that the upper bound of the BSF-program scalability increases proportionally to the problem dimension. It was analytically predicted using the equation (34).

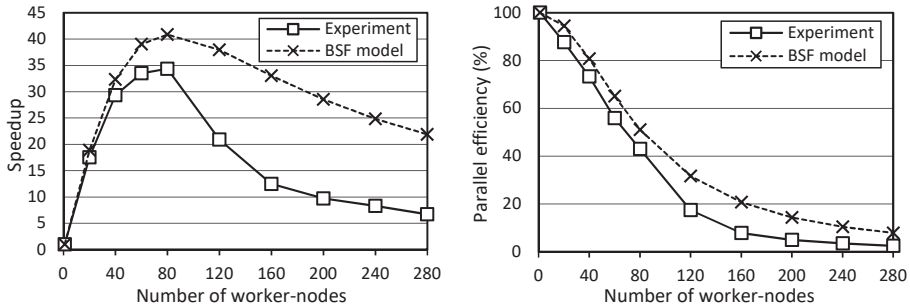
Conclusion

In this paper, the scalability and parallel efficiency of the iterative Cimmino algorithm used to solve large-scale linear inequality systems on multiprocessor systems with distributed memory were investigated. To do this, we used the BSF (Bulk Synchronous Farm) parallel computation model based on the "master-slave" paradigm. The BSF-implementation of the Cimmino algorithm in the form of operations on lists using higher-order functions *Map* and *Reduce* is described. A scalability upper bound of the BSF-implementation of the Cimmino algorithm is obtained. This estimation tells us the following. If space dimension n is greater than or equal to the number m of inequalities, then the upper bound of the scalability of the Cimmino algorithm on lists increases in proportion to the dimension of the problem n. So,



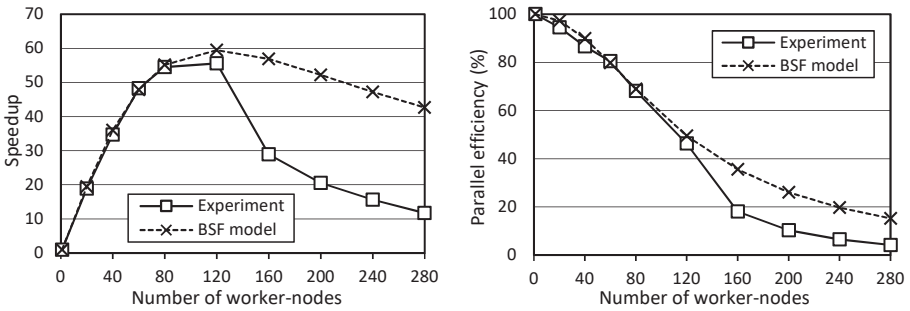
(a) Speedup (b) Parallel efficiency

Figure 1. Experiments for $n = 1\,500$ and $m = 3\,002$



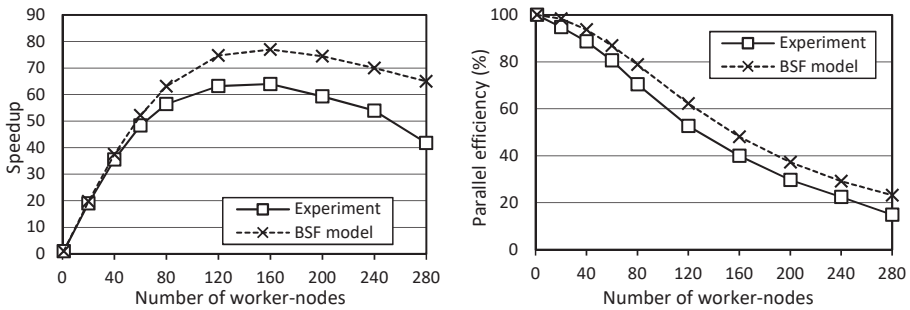
(a) Speedup (b) Parallel efficiency

Figure 2. Experiments for $n = 5\,000$ and $m = 10\,002$



(a) Speedup (b) Parallel efficiency

Figure 3. Experiments for $n = 10\,000$ and $m = 20\,002$



(a) Speedup (b) Parallel efficiency

Figure 4. Experiments for $n = 16\,000$ and $m = 32\,002$

we may conclude that the Cimmino algorithm on lists is scalable well. Also, the equations for estimating the speedup and parallel efficiency of the Cimmino algorithm on lists are obtained. The implementation of the Cimmino algorithm in C++ language using the BSF algorithmic skeleton and the MPI parallel programming library was performed. This implementation is freely available on Github, at `https://github.com/leonid-sokolinsky/BSF-Cimmino`. On a cluster computing system, the large-scale experiments were conducted to obtain the actual speedup and parallel efficiency curves for systems having number of variables 1 500, 5 000, 10 000, 16 000 and the number of inequalities 3 002, 10 002, 20 002, 32 002, respectively. The results of the experiments showed that the BSF model predicts the upper bound of the scalability of the Cimmino algorithm on lists with high accuracy. As future research directions, we intend to do the following:

1) apply the Cimmino algorithm to implement the Qwest phase of the NSLP algorithm [2], designed to solve large-scale non-stationary linear programming problems;

2) carry out computational experiments to solve large-scale linear programming problems on a cluster computer system under the conditions of dynamically changing the input data.

Acknowledgments

The study was partially supported by the RFBR according to research project No. 17-07-00352-a, by the Government of the Russian Federation according to Act 211 (contract No. 02.A03.21.0011) and by the Ministry of Science and Higher Education of the Russian Federation (government order 2.7905.2017/8.9).

References

1. Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Society for Industrial and Applied Mathematics (2009)

2. Sokolinskaya, I., Sokolinsky, L.B.: On the Solution of Linear Programming Problems in the Age of Big Data. Parallel Computational Technologies. PCT 2017. Communications in Computer and Information Science 753, 86–100 (2017), DOI: 10.1007/978-3-319-67035-5_7

3. Censor, Y., Elfving, T., Herman, G.T., Nikazad, T.: On Diagonally Relaxed Orthogonal Projection Methods. SIAM Journal on Scientific Computing 30, 473–504 (2008), DOI: 10.1137/050639399

4. Zhu, J., Li, X.: The Block Diagonally-Relaxed Orthogonal Projection Algorithm for Compressed Sensing Based Tomography. In: 2011 Symposium on Photonics and Optoelectronics (SOPO). pp. 1–4. IEEE (2011)

5. Censor, Y.: Mathematical optimization for the inverse problem of intensity-modulated radiation therapy. In: Palta, J.R. and Mackie, T.R. (eds.) Intensity-Modulated Radiation Therapy: The State of the Art. pp. 25–49. Medical Physics Publishing, Madison, WI (2003)

6. Agmon, S.: The relaxation method for linear inequalities. The Canadian Journal of Mathematics 6, 382–392 (1954), DOI: 10.4153/CJM-1954-037-2

7. Motzkin, T.S., Schoenberg, I.J.: The relaxation method for linear inequalities. The Canadian Journal of Mathematics 6, 393–404 (1954), DOI: 10.4153/CJM-1954-038-x

8. Cimmino, G.: Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. La Ric. Sci. XVI, Ser. II, Anno IX, 1. 326–333 (1938)

9. Benzi, M.: Gianfranco Cimmino's Contributions to Numerical Mathematics. In: Atti del SemiSeminario di Analisi Matematica, Dipartimento di Matematica dell'Universitá di Bologna (Ciclo di Conferenze in Ricordo di Gianfranco Cimmino, 2004). pp. 87–109. Tecnoprint, Bologna, Italy (2005)

10. Censor, Y., Zenios, S.A.: Parallel Optimization: Theory, Algorithms, and Applications. Oxford University Press, New York (1997)

11. Censor, Y., Elfving, T.: New methods for linear inequalities. Linear Algebra Appl. 42, 199–211 (1982), DOI: 10.1016/0024-3795(82)90149-5

12. Censor, Y.: Sequential and parallel projection algorithms for feasibility and optimization. In: Censor, Y. and Ding, M. (eds.) Proc. SPIE 4553, Visualization and Optimization Techniques, (25 September 2001). pp. 1–9. International Society for Optics and Photonics (2001)

13. Kelley, C.T.: Iterative Methods for Linear and Nonlinear Equations. Society for Industrial and Applied Mathematics, Philadelphia (1995)

14. Bilardi, G., Pietracaprina, A.: Models of Computation, Theoretical. In: Encyclopedia of Parallel Computing. pp. 1150–1158. Springer US, Boston, MA (2011)

15. JaJa, J.F.: PRAM (Parallel Random Access Machines). In: Encyclopedia of Parallel Computing. pp. 1608–1615. Springer US, Boston, MA (2011)

16. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM 33, 103–111 (1990), DOI: 10.1145/79173.79181

17. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: towards a realistic model of parallel computation. In: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming – PPOPP'93. pp. 1–12. ACM Press, New York, New York, USA (1993)

18. Forsell, M., Leppanen, V.: An extended PRAM-NUMA model of computation for TCF programming. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012. pp. 786–793. IEEE Computer Society, Washington, DC, USA (2012)

19. Gerbessiotis, A. V.: Extending the BSP model for multi-core and out-of-core computing: MBSP. Parallel Computing 41, 90–102 (2015), DOI: 10.1016/j.parco.2014.12.002

20. Lu, F., Song, J., Pang, Y.: HLognGP: A parallel computation model for GPU clusters. Concurrency and Computation Practice and Experience 27, 4880–4896 (2015), DOI: 10.1002/cpe.3475

21. Sokolinsky, L.B.: Analytical Estimation of the Scalability of Iterative Numerical Algorithms on Distributed Memory Multiprocessors. Lobachevskii Journal of Mathematics 39, 571–575 (2018), DOI: 10.1134/S1995080218040121

22. Sokolinskaya, I., Sokolinsky, L.B.: Scalability Evaluation of NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems. Supercomput. RuSCDays 2017. Communications in Computer and Information Science 793, 40–53 (2017), DOI: 10.1007/978-3-319-71255-0_4

23. Cole, M.I.: Parallel programming with list homomorphisms. Parallel Processing Letters 05, 191–203 (1995), DOI: 10.1142/S0129626495000175

24. Sokolinskaya, I., Sokolinsky, L.: Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. Supercomput. RuSCDays 2016. Communications in Computer and Information Science 687, 212–223 (2016), DOI: 10.1007/978-3-319-55669-7_17

25. Kostenetskiy, P.S., Safonov, A.Y.: SUSU Supercomputer Resources. In: Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. vol. 1576. pp. 561–573 (2016)