Scalable Parallel Algorithm for Solving Non-stationary Systems of Linear Inequalities

L. B. Sokolinsky[1*] and I. M. Sokolinskaya[1**]

(Submitted by E. E. Tyrtyshnikov)

[1]*South Ural State University (National Research University), Chelyabinsk, 454080 Russia*

Abstract—In this paper, a scalable iterative projection-type algorithm for solving non-stationary systems of linear inequalities is considered. A non-stationary system is understood as a large-scale system of inequalities in which coefficients and constant terms can change during the calculation process. The proposed parallel algorithm uses the concept of pseudo-projection which generalizes the notion of orthogonal projection. The parallel pseudo-projection algorithm is implemented using the parallel BSF-skeleton. An analytical estimation of the algorithm scalability boundary is obtained on the base of the BSF cost metric. The large-scale computational experiments were performed on a cluster computing system. The obtained results confirm the efficiency of the proposed approach.

1. INTRODUCTION

The problem of finding a feasible solution to a linear inequality system also known as a feasibility problem is often encountered in the practice of mathematical modeling. As examples, we can mention linear programming [1, 2], image reconstruction from projections [3], tomography image reconstruction [4] and intensity modulated radiation therapy [5]. In many cases, the linear inequality systems arising in such context involve up to tens of millions of inequalities and up to hundreds of millions of variables [2]. Moreover, in mathematical economic models, systems of linear inequalities are often non-stationary. It means that the coefficients of the system and constant terms are changed during the process of solving the problem, and the period of changing the source data can be within hundredths of a second.

At present time, there are a lot of methods for solving systems of linear inequalities. Among these methods, we can distinguish a class of "self-correcting" iterative methods that can be parallelized efficiently. Pioneering works here are papers [6, 7], which propose the Agmon−Motzkin−Schoenberg relaxation method for solving systems of linear inequalities. This method uses the orthogonal projection onto a hyperplane in Euclidean space. Censor and Elfving in [3, 8] proposed a modification of the Cimmino method [9−11] for solving systems of linear inequalities in Euclidean space $\mathbb{R}^n$. The similar method of pseudo-projections based on Fejer approximations was proposed by the authors in [12]. In the article [2], the pseudo-projection method was used to solve the problem of finding a feasible solution to a non-stationary linear inequality system. The convergence theorem for this method was proven by the authors for the case when changing the feasible set is a translation. We have constructed a parallel implementation of the pseudo-projection method and executed large-scale computational experiments on a cluster computing system by varying the displacement rate of the polytope M bounding the feasible

*E-mail: `leonid.sokolinsky@susu.ru`
**E-mail: `irina.sokolinskaya@susu.ru`

region. Performed evaluation showed that the parallel pseudo-projection algorithm converges only with very low rate of displacement of the polytope M.

The aims of this article are the following: analyzing the low efficiency of the parallel pseudo-projection algorithm for the non-stationary case, modifying the algorithm to solve this issue, evaluating the modified algorithm and conducting the large-scale computational experiments on a cluster computing system to examine the efficiency of proposed solution.

The paper is organized as follows. In Section 2, we provide a formal definition of a non-stationary system of linear inequalities and describe a modified pseudo-projection algorithm ModAP calculating a feasible solution for such systems under condition of source data dynamic changes. Section 3 describes the ModAPL algorithm, which is a representation of the ModAP algorithm in the form of operations on lists using the higher-order functions *Map* and *Reduce*, and presents a parallel implementation of the ModAPL algorithm. Section 4 is devoted to an analytical evaluation of the ModAPL parallel algorithm scalability by using the cost metric of the BSF parallel computation model. Section 5 provides an information about the implementation of the ModAPL parallel algorithm, as well as describes the results of large-scale computational experiments on a cluster computing system that confirm the efficiency of the proposed approach. Section 6 summarizes the obtained results and concludes that the scalability of the algorithm depends on the number of dynamically changing parameters of the source linear inequality system.

2. NON-STATIONARY PROBLEM AND PSEUDO-PROJECTION ALGORITHM

Let the following feasible system of linear inequalities be given in $\mathbb{R}^n$:

$$A^{(t)}x \leq b^{(t)}, \tag{1}$$

where the matrix $A^{(t)}$ has m rows. The non-stationarity of the system (1) is understood in the sense that the entries of the matrix $A^{(t)}$ and the elements of column $b^{(t)}$ depend on time $t \in \mathbb{R}_{\geq 0}$. Let $M^{(t)}$ be a polytope bounding the feasible region of the system (1) at instant of time t. Such a polytope is always a closed convex set. We will also assume that the polytope $M^{(t)}$ is a bounded set. Let us define the distance from the point $x \in \mathbb{R}^n$ to the polytope $M^{(t)}$ as follows:

$$d\left(x, M^{(t)}\right) = \inf_{y \in M^{(t)}} ||x - y||,$$

where $||\cdot||$ signifies the Euclidean norm. Let us denote the i-th row of the matrix $A^{(t)}$ as $a_i^{(t)}$. We assume from now on that $a_i^{(t)}$ is not equal to the zero vector for all $i = 1, \ldots, m$. Let $P_i^{(t)}$ be a half-space representing a set of feasible points for the i-th inequality of the system (1):

$$P_i^{(t)} = \left\{x | x \in \mathbb{R}^n, \left\langle a_i^{(t)}, x \right\rangle \leq b_i^{(t)}\right\}.$$

Then, $M^{(t)} = \bigcap_{i=1}^{m} P_i^{(t)}$. Each equation $\left\langle a_i^{(t)}, x \right\rangle = b_i^{(t)}$ defines the corresponding hyperplane $H_i^{(t)}$:

$$H_i^{(t)} = \left\{x \in \mathbb{R}^n | \left\langle a_i^{(t)}, x \right\rangle = b_i^{(t)}\right\}.$$

We define the *reflection vector* $\rho_{H_i^{(t)}}(x)$ of the point x with respect to the hyperplane H_i as follows:

$$\rho_{H_i^{(t)}}(x) = \frac{\left\langle a_i^{(t)}, x \right\rangle - b_i^{(t)}}{\left|\left|a_i^{(t)}\right|\right|^2} a_i^{(t)}.$$

Then, the *orthogonal projection* $\pi_{H_i^{(t)}}(x)$ of the point x onto hyperplane $H_i^{(t)}$ is calculated by the equation $\pi_{H_i^{(t)}}(x) = x - \rho_{H_i^{(t)}}(x)$.

Define the vector $\rho^{+}_{H_i^{(t)}}(x)$ as a *positive slice* of the reflection vector

$$\rho^{+}_{H_i^{(t)}}(x) = \frac{\max\left\{\left\langle a_i^{(t)}, x\right\rangle - b_i^{(t)}, 0\right\}}{\left\|a_i^{(t)}\right\|^2} a_i^{(t)}. \tag{2}$$

The vector $\rho^{+}_{H_i^{(t)}}(x)$ will be a non-zero vector if and only if the point x does not satisfy the i-th inequality of the system (1). Let us designate

$$\varphi^{(t)}(x) = \frac{1}{h}\sum_{i=1}^{m} \rho^{+}_{H_i^{(t)}}(x), \tag{3}$$

where h is the number of non-zero terms in the sum $\sum_{i=1}^{m} \rho^{+}_{H_i^{(t)}}(x)$.

Define the half-space P_i $(i = 1, \ldots, m)$ as follows: $P_i^{(t)} = \left\{x \in \mathbb{R}^n | \left\langle a_i^{(t)}, x\right\rangle \le b_i^{(t)}\right\}$. We will say that the point $\tilde{x}$ belongs to the half-space $P_i^{(t)}$ with precision $\varepsilon > 0$ and denote it as $\tilde{x} \underset{\varepsilon}{\in} P_i^{(t)}$, if the following condition holds

$$\forall i \in \{1, \ldots, m\}\left(\tilde{x} \in P_i^{(t)} \vee \frac{\left|\left\langle a_i^{(t)}, \tilde{x}\right\rangle - b_i^{(t)}\right|}{\left\|a_i^{(t)}\right\|} < \varepsilon\right). \tag{4}$$

This means that, for any $i = 1, \ldots, m$, either the point $\tilde{x}$ belongs to the half-space $P_i^{(t)}$, or the distance between the point $\tilde{x}$ and this half-space is less than ε. We assume that the point $\tilde{x}$ belongs to the polytope $M^{(t)}$ with precision ε, and we will denote it as $\tilde{x} \underset{\varepsilon}{\in} M^{(t)}$, if the following condition holds:

$$\forall i \in \{1, \ldots, m\}\left(\tilde{x} \underset{\varepsilon}{\in} P_i^{(t)}\right).$$

In other words, for any $i = 1, \ldots, m$, either the point $\tilde{x}$ belongs to the polytope $M^{(t)}$, or the distance between the point $\tilde{x}$ and the polytope $M^{(t)}$ is less than ε.

The pseudo-projection algorithm for the non-stationary case is shown in Fig. 1. Here ε is a small positive value that is a parameter of the algorithm. This algorithm calculates a point $\tilde{x}$ that is an approximate solution of the non-stationary system of linear inequalities (1) in the sense that the point $\tilde{x}$ belongs to the polytope $M^{(t)}$ with precision ε (Step 4). If the polytope M does not change over time, the Algorithm 1 finishes in a finite number of steps for any $\varepsilon > 0$. This follows from the fact that the mapping $\alpha^{(t)} = x_k - \varphi^{(t)}(x_k)$ used in the Step 3, when t is fixed, is an single-valued continuous $M^{(t)}$-fejerian mapping [13], and consequently the iterative process implemented in the Algorithm 1 converges to a point belonging to the polytope $M^{(t)}$ [14]. Another proof of the convergence of the Algorithm 1 can be found in [3]. However, both proofs are valid only for stationary problems. The non-stationary

---

Algorithm 1.

1:   $k := 0$
2:   $x_0 := \mathbf{0}$
3:   $x_{k+1} := x_k - \varphi^{(t)}(x_k)$
4:   If $x_{k+1} \underset{\varepsilon}{\in} M^{(t)}$ go to 7
5:   $k := k + 1$
6:   go to 3
7:   $\tilde{x} := x_{k+1}$
8:   Stop
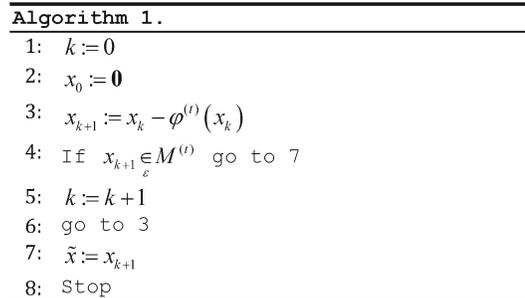
Fig. 1. Pseudo-projection algorithm for non-stationary systems of linear inequalities.
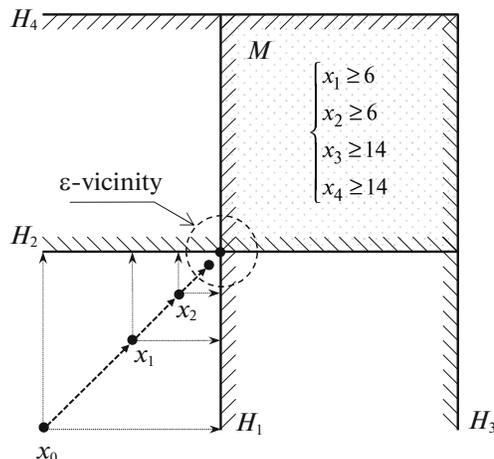
Fig. 2. The case, when the rate of convergence slows down.

case was considered by us in [2] under the assumption that changing the initial data is a translation of the polytope $M^{(t)}$. For this case, we have proved the theorem of a sufficient condition of converging the iterative process implemented by the Algorithm 1.

We have performed a parallel implementation of the Algorithm 1 using the BSF-skeleton [15] and have executed large-scale computational experiments on the "Tornado SUSU" [16] computing cluster. We simulated the non-stationarity of the original inequality system by displacing the polytope $M^{(t)}$ along a straight line at a given rate. The obtained results show that Algorithm 1 demonstrates good scalability of up to 200 processor nodes when solving a non-stationary system involving 108002 inequalities and 54000 variables. However, in some cases, the maximum rate of the polytope displacement, at which the algorithm converged, did not exceed two units per second. It is insufficient for solving practical problems. Such result can be explained by the simple example shown in Fig. 2. A peculiarity of the Fejer mapping $\alpha^{(t)} = x_k - \varphi^{(t)}(x_k)$ used in Step 3 is that each new approximation is closer to the polytope than the previous one. But, in the case shown in Fig. 2, the Fejer process never reaches the polygon M. The most we can achieve in a finite number of iterations is to get a point inside of the polygon vertex vicinity. Assume, that at time t_1, the Algorithm 1 starts to perform the iterative process (Step 3) giving an approximate solution $\tilde{x}$ with distance to the polytope $M^{(t_1)}$ less than ε (we used $\varepsilon = 10^{-7}$ in the experiments). Let the calculation be stopped at time t_2. During the elapsed time, the polytope is displaced to the position $M^{(t_2)}$, the distance from which to the point $\tilde{x}$ is more than ε. Therefore, we have to start the calculation again. In such a way, we will never get a solution with given precision.

In order to repair the problem, we replaced the mapping $\varphi^{(t)}$ used in Step 3 of Algorithm 1 with the following mapping $\psi^{(t)}$:

$$\psi^{(t)}(x) = \lambda \frac{\varphi^{(t)}(x)}{||\varphi^{(t)}(x)||}, \tag{5}$$

where λ is a positive constant being a parameter of the algorithm. A peculiarity of the mapping $\psi^{(t)}$ is that its result is always a vector of fixed length λ. The modified algorithm, named ModAP, is presented in Fig. 3.

3. PARALLEL VERSION OF MODAP ALGORITHM

To construct a parallel version of the algorithm ModAP, we use the BSF (Bulk Synchronous Farm) model of parallel computations [17]. In accordance to the technique proposed in [18], we represent the algorithm in the form of operations on lists using the higher-order functions Map and $Reduce$ defined in the Bird–Meertens formalism [19]. Let us define the list $\mathbb{A} = [(a_1, b_1), \ldots, (a_m, b_m)]$, where a_i is the i-th row of the matrix A, and b_i is the i-th element of the column b $(i = 1, \ldots, m)$. Here, we omit the upper index of time (t) assuming the moment of time to be fixed.

```
Algorithm 2.
1:  k := 0
2:  x_0 := 0
3:  x_{k+1} := x_k − ψ^{(t)}(x_k)
4:  If  x_{k+1} ∈ M_ε^{(t)}  go to 7
5:  k := k + 1
6:  go to 3
7:  x̃ := x_{k+1}
8:  Stop
```

Fig. 3. Modified pseudo-projection algorithm (ModAP).

```
Algorithm 3.
1:  input A
2:  x := 0
3:  B := Map(F_x, A)
4:  y := Reduce(+, B)
5:  x := x − λ y/||y||
6:  update A
7:  if  x ∈ M_A,ε  goto 9
8:  goto 3
9:  output x
10: stop
```

Fig. 4. ModAP algorithm on lists.

Let us define the parameterized function $F_x : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n \times \{0,1\}$ as follows:

$$F_x(a_i, b_i) = \rho_{H_i}^+(x). \tag{6}$$

This function maps the pair (a_i, b_i) to the vector $y_i = \rho_{H_i}^+(x)$, which is the positive slice of the reflection vector of the point x relative to the hyperplane $H_i = \{x \in \mathbb{R}^n | \langle a_i, x \rangle = b_i\}$. The higher-order function $Map(F_x, \mathbb{A})$ applies the function F_x to each element of list $\mathbb{A}$ converting it to the following list

$$\mathbb{B} = [y_1, \ldots, y_m] = [F_x(a_1, b_1), \ldots, F_x(a_m, b_m)],$$

where $y_i \in \mathbb{R}^n$ $(i = 1, \ldots, m)$. Let the symbol $+$ denote vector addition operation. Then, the higher-order function $Reduce(+, \mathbb{B})$ calculates the vector y, which is the sum of the vectors of the list $\mathbb{B}$: $y = y_1 + \ldots + y_m$. Obviously, according to the equation (3), we have $\varphi(x) = y/h$. Taking (5) into account, we obtain from this

$$\psi(x) = \lambda \frac{y}{||y||}.$$

In such a way, we obtain the sequential version of the ModAP algorithm on lists presented in Fig. 4. In Step 1, the input of the list $\mathbb{A}$ containing initial values of the matrix A and the column b of the inequality system (1) are executed. In Step 2, the zero vector is assigned to the x as an initial approximation. In Step 3, the Map function calculates the list $\mathbb{B}$ of positive slices of the reflection vectors of the point x relative to the hyperplanes bounding the polytope $M_\mathbb{A}$, which is the feasible region of the inequality system (1). In Step 4, the vector y is calculated as the sum of all positive slices of the reflection vectors. Step 5 calculates the next approximation x. In Step 6, the original data of the inequality system (1) is updated. Step 7 checks if the obtained approximation belongs to the polytope $M_\mathbb{A}$ with precision ε. If so, then we go to Step 9, where x is output as the approximate solution, after which the iterative process is stopped. Otherwise, we go from Step 8 to Step 3, and the iterative process continues.

The parallel version of the modified pseudo-projection algorithm on lists (ModAPL) is presented in Fig. 5. We used the master-slave paradigm [20]. It is assumed that the computing system includes one master and K slaves ($K \geq 1$). In Step 1, the master and all slaves load the source data presented as the

```
Algorithm 4.
                        Master                    │            j-th Slave (j=1,…,K)
 1:  input 𝔸                                      │  1:  input 𝔸
 2:  x := 0                                        │  2:
 3:  SendToAllWorkers(x)                           │  3:  RecvFromMaster(x)
 4:                                                │  4:  𝔹[j] := Map(F_x, 𝔸[j])
 5:                                                │  5:  y^(j) := Reduce(+, 𝔹[j])
 6:  RecvFromAllWorkers(j, y^(j))                  │  6:  SendToMaster(y^(j))
 7:  y := Reduce(+, [y^(1),…,y^(K)])               │  7:
 8:  x := x − λ (y/‖y‖)                            │  8:
 9:  update 𝔸                                      │  9:
10:  if x ∈_ε M_𝔸 then exit := true else exit := false │ 10:
11:  SendToAllWorkers(exit)                        │ 11:  RecvFromMaster(exit)
12:  if exit = true goto 15                        │ 12:  if exit = true goto 16
13:  SendToAllWorkers(𝔸)                           │ 13:  RecvFromMaster(𝔸)
14:  goto 3                                        │ 14:  goto 3
15:  output x                                      │ 15:
16:  stop                                          │ 16:  stop
```

Fig. 5. ModAPL parallel algorithm on lists.

list $\mathbb{A}$. Step 3 starts the iterative process: the master sends the current approximation x to all slaves. In Step 4, the slaves independently calculate their part of the list $\mathbb{B}$. In Step 5, the slaves independently sum up the elements of their sublists of the list $\mathbb{B}$. In Step 6, the calculated vectors $y^{(1)},\ldots,y^{(K)}$ are sent to the master. In Step 7, the master calculates the resulting vector y. Step 8 calculates the next approximation x. In Step 9, the original data of the inequality system (1) is updated by the master and all slaves. Step 10 checks the stopping criterion and assigns the appropriate value to the variable exit. In Step 11, the master sends the value of the variable exit to all slaves. Depending on this value, we either go to the next iteration or terminate the iterative process (Steps 12−16).

4. ANALYTICAL STUDY OF PARALLEL ALGORITHM

To obtain an analytical estimation of the scalability of the parallel version of the ModAPL algorithm (Fig. 5) we use the cost metric of the BSF model [18]. It includes the following parameters:

K : number of slave nodes;
t_s : time spent by the master node to send current approximation to one slave node (excluding latency);
t_{Map} : time spent by a single slave node to process the entire list $\mathbb{A}$;
t_p : time spent by the master node to process results received from the slave nodes and to check the stopping criterion;
t_r : time spent by the master node to receive the result from one slave node (excluding latency);
t_a : time spent by a node (master or slave) to process an addition of two vectors;
l : length of the list $\mathbb{A}$ (the same as the length of the list $\mathbb{B}$);
L : latency (time of transferring one-byte message node-to-node).

For the ModAPL algorithm we have

$$l = m. \tag{7}$$

Within one iteration, we introduce the following notation:

c_s : number of float values sending from the master node to one slave node;
c_{Map} : number of arithmetic operations to execute the higher-order function Map for the entire list $\mathbb{A}$;
c_a : number of arithmetic operations to add two vectors in n-dimensional space;
c_r : number of float values sending from one slave node to the master node;

c_p : number of arithmetic operations performed by the master in Steps 8 and 10 of Algorithm 4;

c_u : number of float values sending from the master node to one slave node in Step 13 of Algorithm 4.

Calculate the specified values. At the beginning of each iteration in Step 3, the master sends the vector x containing n float values to every slave. Therefore $c_s = n$. In the Map step (Step 3 of the Algorithm 3), the function F_x defined by the equation (6) is applied to all elements of the list $\mathbb{A}$ having the length m. In view of (2), this implies $c_{Map} = (5n + 1)m$. To add two vectors of dimension n, one must execute n arithmetic operations. Therefore, $c_a = n$. In Step 6 of Algorithm 4, jth slave node sends the vector $y^{(j)}$ of dimension n to the master node. Hence it follows that $c_r = n$. Assume that the square root is calculated by using the first four terms of the Taylor series. This is 9 arithmetic operations. Then, the execution of Step 8 of Algorithm 4 requires $5n + 8$ arithmetic operations. According to the equation (4), the execution of Step 11 requires $(6n + 11)m$ arithmetic operations. Hence it follows that

$$c_p = (6n + 11)m + 5n + 8.$$

If only one value changes in the source data during one iteration, then one real number must be sent in Step 13. In this case, we have $c_u = 1$. If all values changes in the source data during one iteration, then $(n + 1)m$ real number must be sent in Step 13. In this case, we have $c_u = (n + 1)m$. Let τ_{op} denote the time that a processor node spends to execute one arithmetic operation (or one comparison operations), and τ_{tr} denote the time that a processor node spends to send one float value to another processor node excluding latency. Then for $c_u = 1$ we get the following values for the cost parameters of the ModAPL parallel algorithm:

$$t_s = (c_s + c_u)\tau_{tr} = (n + 1)\tau_{tr}; \tag{8}$$

$$t_{Map} = c_{Map}\tau_{op} = (5n + 1)m\tau_{op}; \tag{9}$$

$$t_r = c_r\tau_{tr} = n\tau_{tr}; \tag{10}$$

$$t_a = c_a\tau_{op} = n\tau_{op}; \tag{11}$$

$$t_p = c_p\tau_{op} = ((6n + 11)m + 5n + 8)\tau_{op}. \tag{12}$$

Using the scalability boundary equation from [18], and taking into account (7), we obtain from (8)–(12) the following estimation:

$$K_{MAX} = \sqrt{\frac{t_{Map} + lt_a}{2L + t_s + t_r + t_a}} = \sqrt{\frac{(5n + 1)m\tau_{op} + nm\tau_{op}}{2L + (n + 1)\tau_{tr} + n\tau_{tr} + n\tau_{op}}}. \tag{13}$$

Assuming $m = kn$ for some $k \in \mathbb{N}$ and $n \gg 1$, it is easy to get the following estimation from (13):

$$K_{MAX} = O\left(\sqrt{n}\right), \tag{14}$$

where K_{MAX} is the number of processor nodes, for which the maximal speedup is achieved.

When $c_u = (n + 1)m$ we have $t_s = (c_s + c_u)\tau_{tr} = (n + (n + 1)m)\tau_{tr}$. We obtain from this the following estimation:

$$K_{MAX} = \sqrt{\frac{t_{Map} + lt_a}{2L + t_s + t_r + t_a}} = \sqrt{\frac{(5n + 1)m\tau_{op} + nm\tau_{op}}{2L + (n + (n + 1)m)\tau_{tr} + n\tau_{tr} + n\tau_{op}}}. \tag{15}$$

Assuming $m = kn$ for some $k \in \mathbb{N}$ and $n \gg 1$, it is easy to get the following estimation from (15):

$$K_{MAX} = O\left(1\right). \tag{16}$$

Thus, we can conclude that the ModAPL parallel algorithm will demonstrate a limited scalability when a small part of the source data is dynamically changed. In this case, according to the equation (14), the scalability bound increases proportionally to the square root of n, where n is the space dimension. If a large part of the source data is dynamically changed, then, according to the equation (16), the ModAPL parallel algorithm becomes inefficient due to the lack of speedup.

$$\begin{cases} x_0 & & & & \leq & 200 \\ & x_1 & & & \leq & 200 \\ & & \ddots & & \cdots & \cdots \\ & & & x_{n-1} & \leq & 200 \\ x_0 & + \; x_1 & \cdots & + \; x_{n-1} & \leq & 200(n-1)+100 \\ x_0 & + \; x_1 & \cdots & + \; x_{n-1} & \geq & 100 \\ x_0 & & & & \geq & 0 \\ & x_1 & & & \geq & 0 \\ & & \ddots & & \cdots & \cdots \\ & & & x_{n-1} & \geq & 0 \end{cases}$$

Fig. 6. Test problem.

5. COMPUTATIONAL EXPERIMENTS

To verify the efficiency of the ModAPL parallel algorithm, we implemented this algorithm in C++ language using a parallel BSF-skeleton [15] based on MPI parallel programming library. All source codes are freely available at `https://github.com/leonid-sokolinsky/NSLP-Quest`. As a test problem, we used a scalable inequality system of dimension n from [21]. This system is presented in Fig. 6. The number of inequalities in this system is $m = 2n + 2$. The non-stationarity of the problem was simulated by a translation of the polytope $M^{(t)}$. The computational experiments were conducted on the "Tornado SUSU" computing cluster [16], whose specifications are shown in Table 1. The results of the experiments are presented in Fig. 7. The diagrams show the dependence of the running time of the ModAPL parallel algorithm on the displacement rate of the polytope $M^{(t)}$. The values of coordinate-wise displacement indicate how many units per second will be added to each coordinate of the polytope M before next iteration. In the first experiment, we used the system in Fig. 6 with dimension $n = 32000$ and number of inequalities $m = 64002$. Calculations were conducted on the hardware configurations with 50, 75, and 100 processor nodes. The results of this experiment are presented in Fig. 7a. In the diagram, K denotes the number of processor nodes in the hardware configuration. For $K = 50$, the highest rate of the coordinate-wise shift at which the iterative process "catch up with the polytope" was approximately 10 units per second. Adding up the displacement vectors for all coordinates, we get the total rate vector with a length equal to $\sqrt{32000 \cdot 10^2} \approx 1789$ units per second. For $K = 75$, the highest total displacement rate at which the algorithm converged increases to 2683 units per second. And for $K = 100$, the convergence of the algorithm was observed at the total displacement rate of up to 3220 units per second. For $K > 100$, the parallel efficiency for the problem of dimension $n = 32000$ began to fall off.

In the second experiment, we used the system in Fig. 6 with dimension $n = 54000$ and number of inequalities $m = 108002$. Calculations were conducted on the hardware configurations with 75, 100, and 150 processor nodes. The results of this experiment are presented in Fig. 7b. In this case, the maximum

Table 1. Specifications of "Tornado SUSU" computing cluster

Number of processor nodes	480
Processor	Intel Xeon X5680 (6 cores 3.33 GHz)
Processors per node	2
Memory per node	24 GB DDR3
Interconnect	InfiniBand QDR (40 Gbit/s)
Operating system	Linux CentOS
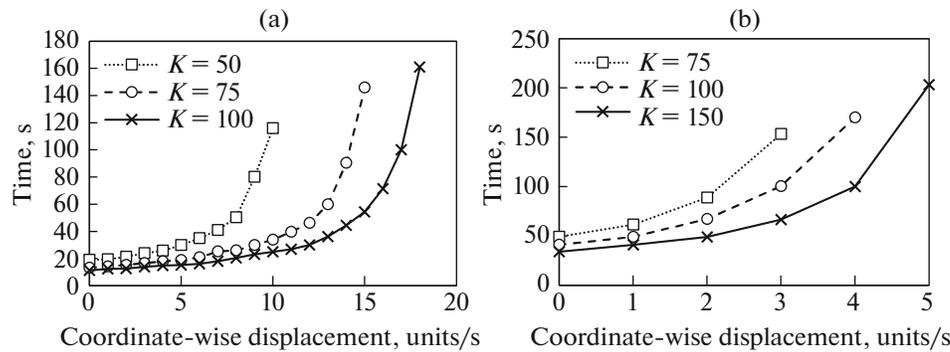
Fig. 7. Experiment results (K is the number of processor nodes): (a) $n = 32000$ and $m = 64002$; (b) $n = 54000$ and $m = 108002$.

displacement rates of the polytope at which the algorithm converged were 697, 930, and 1162 units per second, respectively. For $K > 150$, the parallel efficiency for the problem of dimension $n = 54000$ also began to fall off.

Thus, the conducted experiments show that the proposed modified algorithm of pseudo-projections allows the effective parallelization on cluster computing systems and is able to find feasible points for non-stationary systems of linear inequalities that dynamically changed in a certain way with high rate.

6. CONCLUSION

In this article, we presented the modified parallel iterative pseudo-projection algorithm that can find feasible points for non-stationary systems of linear inequalities of large dimensions on cluster computing systems. This algorithm was presented as the ModAPL algorithm on lists using the higher-order functions *Map* and *Reduce*. For the ModAPL algorithm, we obtained an estimation of the scalability bound of its parallel version using the cost metric of the BSF parallel computing model [18]. If a small part of the source data is dynamically changed during one iteration then the scalability bound is equal to $O\left(\sqrt{n}\right)$, where n is the problem dimension. If a large part of the source data is dynamically changed during one iteration then the scalability bound is equal to $O\left(1\right)$, which means there is no acceleration at all. We implemented ModAPL parallel algorithm in C++ language using a parallel BSF-skeleton [15]. All source codes are freely available at `https://github.com/leonid-sokolinsky/NSLP-Quest`. By using this implementation, we conducted the large-scale computational experiments on a computing cluster. We simulated the problem non-stationarity by transferring the polytope bounding the feasible region by a certain number of units during each iteration. The conducted experiments show that the ModAPL parallel algorithm is able to find feasible points for non-stationary systems of linear inequalities with 54 000 variables and 108 002 inequalities on a cluster computing system with 200 processor nodes.

FUNDING

REFERENCES

1. R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem* (Soc. Ind. Appl. Math., Philadelphia, 2009). https://doi.org/10.1137/1.9780898719000
2. I. Sokolinskaya and L. B. Sokolinsky, "On the solution of linear programming problems in the age of big data," in *Parallel Computational Technologies. PCT 2017,* Commun. Comput. Inform. Sci. **753**, 86−100 (2017). https://doi.org/10.1007/978-3-319-67035-5_7
3. Y. Censor, T. Elfving, G. T. Herman, and T. Nikazad, "New methods for linear inequalities," SIAM J. Sci. Comput. **30**, 473−504 (2008). https://doi.org/10.1137/050639399

4. J. Zhu and X. Li, "The block diagonally-relaxed orthogonal projection algorithm for compressed sensing based tomography," in *Proceedings of the 2011 Symposium on Photonics and Optoelectronics (SOPO)* (2011), pp. 1–4. https://doi.org/10.1109/SOPO.2011.5780660

5. Y. Censor, "Mathematical optimization for the inverse problem of intensity-modulated radiation therapy," in *Intensity-Modulated Radiation Therapy: The State of the Art,* Ed. by J. R. Palta and T. R. Mackie (Med. Phys. Publ., Madison, WI, 2003), pp. 25–49. https://doi.org/10.1118/1.1628279

6. S. Agmon, "The relaxation method for linear inequalities," Canad. J. Math. **6**, 382–392 (1954). https://doi.org/10.4153/CJM-1954-037-2

7. T. S. Motzkin and I. J. Schoenberg, "The relaxation method for linear inequalities," Canad. J. Math. **6**, 393–404 (1954). https://doi.org/10.4153/CJM-1954-038-x

8. Y. Censor and T. Elfving, "New methods for linear inequalities," Linear Algebra Appl. **42**, 199–211 (1982). https://doi.org/10.1016/0024-3795(82)90149-5

9. G. Cimmino, "Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari," Ric. Sci., XVI, Ser. II, Anno IX, No. 1, 326–333 (1938).

10. A. Galantai, *Projectors and Projection Methods,* Vol. 6 of *Advances in Mathematics* (Springer US, Boston, MA, 2004). https://doi.org/10.1007/978-1-4419-9180-5

11. I. M. Sokolinskaya and L. B. Sokolinsky, "Scalability evaluation of cimmino algorithm for solving linear inequality systems on multiprocessors with distributed memory," Supercomput. Front. Innov. **5** (2), 11–22 (2018). https://doi.org/10.14529/jsfi180202

12. I. Sokolinskaya, "Parallel method of pseudoprojection for linear inequalities," in *Parallel Computational Technologies. PCT 2018,* Commun. Comput. Inform. Sci. **910**, 216–231 (2018). https://doi.org/10.1007/978-3-319-99673-8_16

13. V. V. Vasin and I. I. Eremin, *Operators and Iterative Processes of Fejer Type. Theory and Applications* (Walter de Gruyter, Berlin, New York, 2009). https://doi.org/10.1515/9783110218190

14. I. I. Eremin, "Methods of Fejer's approximations in convex programming," Math. Notes Acad. Sci. USSR **3**, 139–149 (1968). https://doi.org/10.1007/BF01094336

15. L. B. Sokolinsky, BSF-skeleton. https://github.com/leonid-sokolinsky/BSF-skeleton. Accessed 2020.

16. P. Kostenetskiy and P. Semenikhina, "SUSU supercomputer resources for industry and fundamental science," in *Proceedings of the 2018 Global Smart Industry Conference (GloSIC)* (IEEE, 2018), pp. 1–7. https://doi.org/10.1109/GloSIC.2018.8570068

17. L. B. Sokolinsky, "Analytical estimation of the scalability of iterative numerical algorithms on distributed memory multiprocessors," Lobachevskii J. Math. **39**, 571–575 (2018). https://doi.org/10.1134/S1995080218040121

18. N. A. Ezhova and L. B. Sokolinsky, "Scalability evaluation of iterative algorithms Used for supercomputer simulation of physical processes," arXiv:1811.04276 (2018). https://doi.org/10.1109/GloSIC.2018.8570131

19. R. S. Bird, "Lectures on constructive functional programming," in *Constructive Methods in Computing Science,* Ed. by M. Broy, Vol. 55 of *NATO ASI Series F: Computer and Systems Sciences* (Springer, Berlin, Heidlberg, 1988), pp. 151–216.

20. S. Sahni and G. Vairaktarakis, "The master-slave paradigm in parallel computer and industrial settings," J. Global Optimiz. **9**, 357–377 (1996). https://doi.org/10.1007/BF00121679

21. I. Sokolinskaya and L. Sokolinsky, "Revised pursuit algorithm for solving non-stationary linear programming problems on modern computing clusters with manycore accelerators," in *Supercomputing. RuSCDays 2016,* Commun. Comput. Inform. Sci. **687**, 212–223 (2016). https://doi.org/10.1007/978-3-319-55669-7_17