ISSN 0965-5425, Computational Mathematics and Mathematical Physics, 2009, Vol. 49, No. 8, pp. 1303–1317. © Pleiades Publishing, Ltd., 2009.
Original Russian Text © V.A. Garanzha, A.I. Golikov, Yu.G. Evtushenko, M.Kh. Nguen, 2009, published in Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 2009, Vol. 49, No. 8, pp. 1369–1384.

Parallel Implementation of Newton's Method for Solving Large-Scale Linear Programs

V. A. Garanzha, A. I. Golikov, Yu. G. Evtushenko, and M. Kh. Nguen

Computing Center, Russian Academy of Sciences, ul. Vavilova 40, Moscow, 119333 Russia
e-mail: garan@ccas.ru

Received February 24, 2009

Abstract—Parallel versions of a method based on reducing a linear program (LP) to an unconstrained maximization of a concave differentiable piecewise quadratic function are proposed. The maximization problem is solved using the generalized Newton method. The parallel method is implemented in C using the MPI library for interprocessor data exchange. Computations were performed on the parallel cluster MVC-6000IM. Large-scale LPs with several millions of variables and several hundreds of thousands of constraints were solved. Results of uniprocessor and multiprocessor computations are presented.

DOI: 10.1134/S096554250908003X

Key words: linear programming, generalized Newton's method, unconstrained optimization, parallel computations.

1. INTRODUCTION

Many applied problems can be reduced to linear problems (LPs), for which numerical methods and software were developed long ago. It may seem that there is not difficult to find an optimal solution in such a problem. However, the dimension of modern problems (millions of variables and hundreds of thousands of constraints) is sometimes too large to be solved by conventional methods. Therefore, new approaches are needed for solving such problems on powerful computers.

Large-scale LPs usually have more than one solution. Such techniques as the simplex method, interior point method, or quadratic penalty function method (e.g., see [1, 2]) make it possible to obtain different solutions in the case of nonuniqueness. For example, the simplex method yields a solution belonging to a vertex of a polyhedron. The interior point method converges to a solution satisfying the strict complementary slackness condition. The method of exterior quadratic penalty function enables one to find the exact normal solution.

In [3, 4], a new method for solving LPs is proposed that is close to the quadratic penalty function method of the form described in [5, 6] and to the modified Lagrangian function method. When applied to the dual LP, this method yields the exact projection of the given point on the set of solutions to the primal LP as a result of the one-time unconstrained maximization of an auxiliary piecewise quadratic function with a finite penalty coefficient. The use of the generalized Newton method for the maximization of the auxiliary function enables one to solve LPs with a very large number of nonnegative variables (several tens of millions) and a moderate number of constraints (several thousand) on computers based on Pentium-IV processors. This paper is devoted to the parallelization of this method so as to make it possible to solve LPs with a larger number of constraints (up to several hundreds of thousands).

In Section 1, we briefly outline the theoretical foundations of the method designed for solving LPs that enables one to find the projection of the given point on the set of solutions to the primal LP. Using duality theory, the auxiliary maximization problem for a concave piecewise quadratic function is obtained in which the number of variables is equal to the number of constraints in the primal LP. Formulas for determining the threshold value of the parameter of this function are given; for this threshold, the one-time maximization of the auxiliary function makes it possible to find the projection of a point on the solution set of the primal LP using simple formulas. The repeated maximization of the same function in which the solution of the primal problem is substituted allows one to solve the dual LP.

In Section 2, we recommend to use the generalized Newton method to maximize the auxiliary concave piecewise quadratic function, which converges in a finite number of steps as shown in [6, 7]. The results of solving test LPs on a uniprocessor computer using this method are presented in Section 3. The comparison of the proposed method (which was implemented in MATLAB) with available commercial and research packages showed that it is competitive with the simplex and the interior point methods.

In Section 4, we propose several variants for parallelizing the generalized Newton method as applied to solving LPs.

In Section 5, we present some numerical results obtained on a parallel cluster. These results show that the proposed approach to solving LPs using the generalized Newton method can be efficiently parallelized and used to solve LPs with several million variables and up to two hundred thousand constraints. For example, for LPs with one million variables and 10000 constraints, one of the parallelizing schemes for 144 processors of the cluster MVC-6000IM accelerated the computations approximately by a factor of 50, and the computation time was 28 s. A LP with two million variables and 200000 constraints was solved in about 40 min. on 80 processors.

2. REDUCING LP TO UNCONSTRAINED OPTIMIZATION

Let the LP in normal form

$$f^* = \min_{x \in X} c^{\mathrm{T}} x, \quad X = \{x \in R^n : Ax = b, x \geq 0_n\} \tag{P}$$

be given. The dual problem is

$$f^* = \max_{u \in U} b^{\mathrm{T}} u, \quad U = \{u \in R^m : A^{\mathrm{T}} u \leq c\}. \tag{D}$$

Here, $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$ are given, x is the vector of primal variables, u is the vector of dual variables, and 0_i is the i-dimensional zero vector.

Assume that the solution set X^* of primal problem (P) is not empty; therefore, the solution set U^* of dual problem (D) is not empty either. We write the Kuhn–Tucker necessary and sufficient conditions for problems (P) and (D) in the form

$$Ax^* - b = 0_m, \quad x^* \geq 0_n, \quad D(x^*)v^* = 0_n, \tag{1}$$

$$v^* = c - A^{\mathrm{T}} u^* \geq 0_n. \tag{2}$$

Here, we introduced the nonnegative vector of slack variables $v = c - A^{\mathrm{T}}u \geq 0_n$ in the constraints of dual problem (D). By $D(z)$, we denote the diagonal matrix in which the ith diagonal entry is the ith component of the vector z.

Consider the problem of finding the projection of the given point $\hat{x}$ on the solution set X^* of primal problem (P):

$$\frac{1}{2}\|\hat{x}^* - \hat{x}\|^2 = \min_{x \in X^*} \frac{1}{2}\|x - \hat{x}\|^2, \quad X^* = \{x \in \mathbb{R}^n : Ax = b, c^{\mathrm{T}} x = f^*, x \geq 0_n\}. \tag{3}$$

Here and in what follows, we use the Euclidean norm of vectors, and f^* is the optimal value of the objective function of the original LP.

For this problem we define the Lagrangian function

$$L(x, p, \beta) = \frac{1}{2}\|x - \hat{x}\|^2 + p^{\mathrm{T}}(b - Ax) + \beta(c^{\mathrm{T}} x - f^*).$$

Here, $p \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^1$ are the Lagrange multipliers for problem (3). The dual problem for (3) is

$$\max_{p \in R^m} \max_{\beta \in R^1} \min_{x \in R^n_+} L(x, p, \beta). \tag{4}$$

Let us write out the Kuhn–Tucker conditions for problem (3):

$$x - \hat{x} - A^{\mathrm{T}} p + \beta c \geq 0_n, \quad D(x)(x - \hat{x} - A^{\mathrm{T}} p + \beta c) = 0_n, \quad x \geq 0_n, \tag{5}$$

$$Ax = b, \quad c^{\mathrm{T}} x = f^*. \tag{6}$$

It is easy to verify that conditions (5) are equivalent to the equation

$$x = (\hat{x} + A^{\mathrm{T}} p - \beta c)_+. \tag{7}$$

This formula yields a solution of the inner minimization problem in (4).

Substituting (7) in the Lagrangian function $L(x, p, \beta)$, we obtain the dual function for problem (4):

$$\hat{L}(p, \beta) = b^{\mathrm{T}} p - \frac{1}{2}\left\|(\hat{x} + A^{\mathrm{T}} p - \beta c)_+\right\|^2 - \beta f^* + \frac{1}{2}\|\hat{x}\|^2.$$

The function $\hat{L}(p, \beta)$ is concave, piecewise quadratic, and continuously differentiable in its variables p and β.

Dual problem (4) reduces to the solution of the outer maximization problem

$$\max_{p \in \mathbb{R}^m} \max_{\beta \in \mathbb{R}^1} \hat{L}(p, \beta). \tag{8}$$

Solving problem (8), we find the optimizers p and β. Substituting them in (7), we find a solution $\hat{x}^*$ of problem (3), that is, the projection of $\hat{x}$ on the solution set of primal problem (P). The necessary and sufficient conditions for (8) are

$$\hat{L}_p(p, \beta) = b - A(\hat{x} + A^{\mathrm{T}}p - \beta c)_+ = b - Ax = 0_m,$$

$$\hat{L}_\beta(p, \beta) = c^{\mathrm{T}}(\hat{x} + A^{\mathrm{T}}p - \beta c)_+ - f^* = c^{\mathrm{T}}x - f^* = 0,$$

where x is determined by (7). These conditions are fulfilled if and only if $x \in X^*$ and $x = \hat{x}^*$.

Unfortunately, unconstrained optimization problem (8) includes the quantity f^*—the optimal value of the objective function in the original LP—which is not known a priori. However, this problem can be simplified by eliminating this difficulty. For that purpose, we propose to solve the simplified unconstrained maximization problem

$$I = \max_{p \in \mathbb{R}^m} S(p, \beta), \tag{9}$$

where the scalar β is fixed and the function $S(p, \beta)$ is defined by

$$S(p, \beta) = b^{\mathrm{T}}p - \frac{1}{2}\|(\hat{x} + A^{\mathrm{T}}p - \beta c)_+\|^2. \tag{10}$$

Consider problem (9) and its relationship with primal problem (P). Note that, in distinction from (3), the solution of its dual problem (8) is not unique. Naturally, it is desirable to find among all the solutions of problem (8) the minimal value β_* of the Lagrange multiplier β. Then, it follows from Theorem 1 below that we may fix a $\beta \geq \beta_*$ in dual problem (8) and maximize the dual function $\hat{L}(p, \beta)$ only in p; that is, we can solve problem (9). In this case, the pair $[p, \beta]$ is a solution of problem (8), and the triple $[\hat{x}^*, p, \beta]$ is a saddle point of problem (3), in which the projection $\hat{x}^*$—a solution of (P)—is determined by formula (7).

To find the minimal β, we use the Kuhn–Tucker conditions for problem (3), which are necessary and sufficient optimality conditions for this problem. Without loss of generality, assume that the first l components of $\hat{x}^*$ are strictly greater than zero. Then, we can represent the vectors $\hat{x}^*$, $\hat{x}$, and c and the matrix A in the form

$$\hat{x}^{*\mathrm{T}} = [\hat{x}_l^{*\mathrm{T}}, \hat{x}_d^{*\mathrm{T}}], \quad \hat{x}^{\mathrm{T}} = [\hat{x}_l^{\mathrm{T}}, \hat{x}_d^{\mathrm{T}}], \quad c^{\mathrm{T}} = [c_l^{\mathrm{T}}, c_d^{\mathrm{T}}], \quad A = [A_l | A_d], \tag{11}$$

where $\hat{x}_l^* > 0_l$, $\hat{x}_d^* = 0_d$, and $d = n - l$.

According to decomposition (11), the optimal vector of slack variables v^* appearing in the Kuhn–Tucker conditions (1), (2) for problems (P) and (D) can be represented in the form $v^{*\mathrm{T}} = [v_l^{*\mathrm{T}}, v_d^{*\mathrm{T}}]$. Then, by the complementary slackness condition, we have $x^{*\mathrm{T}}v^* = 0$, $x^* \geq 0_n$, $v^* \geq 0_n$, and expression (2) is written as

$$v_l^* = c_l - A_l^{\mathrm{T}}u^* = 0_l, \tag{12}$$

$$v_d^* = c_d - A_d^{\mathrm{T}}u^* \geq 0_d. \tag{13}$$

Using the notation introduced in (11), the necessary and sufficient optimality conditions (5), (6) for problem (3) can be written as

$$\hat{x}_l^* = \hat{x}_l + A_l^{\mathrm{T}}p - \beta c_l > 0_l, \tag{14}$$

$$\hat{x}_d^* = 0_d, \quad \hat{x}_d + A_d^{\mathrm{T}}p - \beta c_d \leq 0_d, \tag{15}$$

$$A_l\hat{x}_l^* = b, \quad c_l^{\mathrm{T}}\hat{x}_l^* = f^*. \tag{16}$$

Among the solutions of system (14)–(16), we find the Lagrange multipliers $[p, \beta]$ such that β is minimal; that is, we have the LP

$$\beta_* = \inf_{\beta \in R^1} \inf_{p \in R^m} \{\beta : A_l^{\mathrm{T}} p - \beta c_l = \hat{x}_l^* - \hat{x}_l, A_d^{\mathrm{T}} p - \beta c_d \le -\hat{x}_d\}. \tag{17}$$

The constraints in this problem are consistent; however, the objective function can be unbounded below. In this case, we assume that $\beta_* = \gamma$, where γ is a certain scalar.

Theorem 1 (see [3, 4]) states that, if the system of equations in (17) has a unique solution with respect to p, then β_* * can be written in the form

$$\beta_* = \begin{cases} \max\limits_{i \in \sigma} \dfrac{(\hat{x}_d + A_d^{\mathrm{T}}(A_l A_l^{\mathrm{T}})^{-1} A_l(\hat{x}_l^* - \hat{x}_l))^i}{(v_d^*)^i}, & \sigma \ne \emptyset \\ \gamma > -\infty, \quad \sigma = \emptyset. \end{cases} \tag{18}$$

Here, $\sigma = \{l + 1 \le i \le n : (v_d^*)^i > 0\}$ is an index set and γ is an arbitrary number.

Theorem 1. *Let the solution set* X^* *of problem* (P) *be not empty. Then, for any* $\beta \ge \beta_*$, *where* β_* *is defined by* (17), *the pair* $[p(\beta), \beta]$, *where* $p(\beta)$ *is a solution to unconstrained maximization problem* (9) (*or, which is the same, a solution to system* $A(\hat{x} + A^{\mathrm{T}}p - \beta c)_+ = b$), *determines the projection* $\hat{x}^*$ *of the given point* $\hat{x}$ *on the solution set* X^* *of primal problem* (P) *by the formula*

$$\hat{x}^* = (\hat{x} + A^{\mathrm{T}} p(\beta) - \beta c)_+. \tag{19}$$

If, in addition, the rank of the matrix A_l *corresponding to the nonzero components of* $\hat{x}^*$ *is* m, *then* β_* *is found by formula* (18) *and an exact solution of dual problem* (D) *is found by solving unconstrained maximization problem* (9) *by the formula*

$$u^* = \frac{1}{\beta}(p(\beta) - (A_l A_l^{\mathrm{T}})^{-1} A_l(\hat{x}^* - \hat{x}_l)).$$

Theorem 1 allows us to replace problem (8), which involves an a priori unknown number f^*, with problem (9), in which this number is replaced by the half-interval $[\beta_*, +\infty)$. The latter problem is considerably simpler from the computational point of view. Note that β_* found by solving LP (17) or by formula (18) can be negative.

The next theorem asserts that, if a point $x^* \in X^*$ is known, a solution of dual problem (D) can be obtained by solving (one time) unconstrained maximization problem (9).

Theorem 2. *Let the solution set* X^* *of problem* (P) *be not empty. Then, for any* $\beta > 0$ *and* $\hat{x} = x^* \in X^*$, *an exact solution of dual problem* (D) *is found by the formula* $u^* = p(\beta)/\beta$, *where* $p(\beta)$ *is a solution of unconstrained maximization problem* (9).

To illustrate the application of this theorem, consider the exterior quadratic penalty method applied to dual problem (D); that is, we consider the problem

$$\max_{p \in \mathbb{R}^m} \left\{ b^{\mathrm{T}} p - \frac{1}{2} \|(A^{\mathrm{T}} p - \beta c)_+\|^2 \right\}. \tag{20}$$

It turns out that an exact solution u^* of dual problem (D) can be obtained without letting the penalty coefficient β tend to $+\infty$ in (20). If $\beta \ge \beta_*$ in (20), then, due to Theorem 1, we find the normal solution $\tilde{x}^*$ of primal problem (P) by formula (19) in which $\hat{x} = 0_n$. According to Theorem 2, the unconstrained maximization problem must then be solved or an arbitrary $\beta > 0$:

$$\max_{p \in \mathbb{R}^m} \left\{ b^{\mathrm{T}} p - \frac{1}{2} \|(\tilde{x}^* + \beta(A^{\mathrm{T}} p - \beta c))_+\|^2 \right\}. \tag{21}$$

Using its solution $p(\beta)$, we obtain the solution $p(\beta)/\beta = u^* \in U^*$ of dual problem (D). Note that the complexity of problem (21) does not exceed that of problem (20). Therefore, solving only two unconstrained maximization problems, one can obtain the exact projection on the solution set of the primal problem and a solution of the dual problem if one takes $\beta \ge \beta_*$ in (20) and an arbitrary positive β in (21).

To simultaneously solve the primal and dual LPs, we propose to use the iterative process

$$x_{s+1} = (x_s + A^{\mathrm{T}} p_{s+1} - \beta c)_+, \tag{22}$$

where the arbitrary parameter $\beta > 0$ is fixed and the vector p_{s+1} is determined by solving the unconstrained maximization problem

$$p_{s+1} \in \underset{p \in \mathbb{R}^m}{\arg\max}\left\{b^{\mathrm{T}}p - \frac{1}{2}\|(x_s + A^{\mathrm{T}}p - \beta c)_+\|^2\right\}. \tag{23}$$

Theorem 3. *Let the solution set X^* of primal problem* (P) *be not empty. Then, for any $\beta > 0$ and any initial point x_0, iterative process* (22), (23) *converges to $x^* \in X^*$ in a finite number of steps ω. The formula $u^* = p_{\omega+1}/\beta$ gives an exact solution of dual problem* (D).

This iterative process is finite and gives an exact solution of primal problem (P) and an exact solution of dual problem (D). Note that this method does not require that the threshold value of the penalty coefficient be known. However, if the value of this coefficient used in the computations is less than the threshold value, the method yields a solution to the primal problem in a finite number of steps rather than the projection of the initial point on the solution set of the primal LP. Note that $x_\omega = x^* \in X^*$ is the projection of $x_{\omega-1}$ on the solution set X^* of problem (P).

2. GENERALIZED NEWTON'S METHOD FOR UNCONSTRAINED MAXIMIZATION OF A PIECEWISE QUADRATIC FUNCTION

Unconstrained maximization problem (23) can be solved using any method, for example, the conjugate gradient method. However, Mangasarian showed that the generalized Newton method is especially efficient for the unconstrained optimization of piecewise quadratic functions (see [7]). A brief description of this method follows.

The function $S(p, \beta, \hat{x})$ to be maximized (see (10)) in problem (9) or in (23) is concave, piecewise quadratic, and differentiable. The conventional Hessian for this function does not exist. Indeed, the gradient

$$\frac{\partial}{\partial p}S(p, \beta, \hat{x}) = b - A(\hat{x} + A^{\mathrm{T}}p - \beta c)_+$$

of $S(p, \beta, \hat{x})$ is not differentiable. However, we can define a generalized Hessian for this function by

$$\frac{\partial^2}{\partial p^2}S(p, \beta, \hat{x}) = -AD^{\#}(z)A^{\mathrm{T}},$$

where $D^{\#}(z)$ is an $n \times n$ diagonal matrix with the ith diagonal entry z_i equal to 1 if $(\hat{x} + A^{\mathrm{T}}p - \beta c)_i > 0$ and to 0 if $(\hat{x} + A^{\mathrm{T}}p - \beta c)_i \le 0$ for $i = 1, 2, \ldots, n$. The generalized Hessian thus defined is an $m \times m$ symmetric negative semidefinite matrix. Since it can be singular, we use the modified Newton direction

$$-\left(\frac{\partial^2}{\partial p^2}S(p, \beta, \hat{x}) - \delta I_m\right)^{-1}\frac{\partial}{\partial p}S(p, \beta, \hat{x}),$$

where δ is a small positive number (in the computations, we usually used $\delta = 10^{-4}$) and I_m is the identity matrix of order m.

In this case, the modified Newton method is written as

$$p_{k+1} = p_k - \left(\frac{\partial^2}{\partial p^2}S(p_k, \beta, \hat{x}) - \delta I_m\right)^{-1}\frac{\partial}{\partial p}S(p_k, \beta, \hat{x}).$$

We used the termination rule

$$\|p_{k+1} - p_k\| \le \mathrm{tol}.$$

Mangasarian studied the convergence of the generalized Newton method for the unconstrained optimization of such a concave piecewise quadratic function with a stepsize chosen using Armijo's rule. A proof of the finite global convergence of the generalized Newton method for the unconstrained optimization of a piecewise quadratic function can be found in [7].

3. COMPUTATIONAL EXPERIMENTS ON A UNIPROCESSOR COMPUTER

We solved randomly generated LPs with a large number of nonnegative variables (up to several millions) and a much smaller number of equality constraints ($n \gg m$).

We specified m and n (the number of rows and columns in the matrix A) and the density of filling the matrix with nonzero entries ρ. For example, $\rho = 1$ implies that all the entries of A were randomly gener-

Table 1

$m \times n \times \rho$	Program	T, s	Iter	Δ_1	Δ_2	Δ_3
$500 \times 10^4 \times 1$	EGM (NEWTON)	55.0	12	1.5×10^{-8}	1.8×10^{-12}	1.2×10^{-7}
	BPMPD (Interior point)	37.4	23	2.3×10^{-10}	1.8×10^{-11}	1.1×10^{-10}
	MOSEK (Interior point)	87.2	6	9.7×10^{-8}	3.8×10^{-9}	1.6×10^{-6}
	CPLEX (Interior point)	80.3	11	1.8×10^{-8}	1.1×10^{-7}	0.0
	CPLEX (Simplex)	61.8	8308	8.6×10^{-4}	1.9×10^{-10}	7.2×10^{-3}
$3000 \times 10^4 \times 0.01$	EGM (NEWTON)	155.4	11	6.1×10^{-10}	3.4×10^{-13}	3.6×10^{-8}
	BPMPD (Interior point)	223.5	14	4.6×10^{-9}	2.9×10^{-10}	3.9×10^{-9}
	MOSEK (Interior point)	42.6	4	3.1×10^{-8}	1.2×10^{-8}	3.7×10^{-8}
	CPLEX (Interior point)	69.9	5	1.1×10^{-6}	1.3×10^{-7}	0.0
	CPLEX (Simplex)	1764.9	6904	3.0×10^{-3}	8.1×10^{-9}	9.3×10^{-2}
$1000 \times (5 \times 10^6) \times 0.01$	EGM (NEWTON)	1007.5	10	3.9×10^{-8}	1.4×10^{-13}	6.1×10^{-7}
$1000 \times 10^5 \times 1$	EGM (NEWTON)	2660.8	8	2.1×10^{-7}	1.4×10^{-12}	7.1×10^{-7}

ated, and, for $\rho = 0.01$, only 1% of entries were generated and the other entries were set to zero. The entries of A were randomly chosen in the interval $[-50, +50]$. The solution x^* of primal problem (P) and the solution u^* of dual problem (D) were generated as follows. $n - 3m$ components of x^* were set to zero, and the others were randomly chosen in the interval [0, 10]. Half of the components of u^* were set to zero, and the others were randomly chosen in the interval $[-10, 10]$. The solutions x^* and u^* were used to find the coefficients in the objective function c and the right-hand sides b of LP (P). The vectors b and c were determined by the formulas

$$b = Ax^*, \quad c = A^{\mathrm{T}}u^* + \xi;$$

if $x_i^* > 0$, then $\xi_i = 0$; if $x_i^* = 0$, then ξ_i was randomly chosen in the interval

$$0 \le \gamma_i \le \xi_i \le \theta_i.$$

In the results presented below, we assumed that all $\gamma_i = 1$ and $\theta_i = 10$. Note that, for γ_i close to zero, $\xi_i = (c - A^{\mathrm{T}}u^*)_i = (v_d^*)_i$ can also be very small. According to formula (18), the a priori unknown quantity β_* can be very large in this case. Then, the generated LP can be difficult to solve.

The proposed method for solving the primal and dual LPs, which combines the use of iterative process (22), (23) and the generalized Newton method, is implemented in MATLAB (see [3, 4]). A 2.6 GHz Pentium-IV computer with 1 Gb of memory was used for the computations. Numerical experiments with randomly generated LPs showed that the proposed method is very efficient for LPs with a large number of nonnegative variables (up 50 million variables) and a moderate number of equality constraints (up to 5 thousand). The time needed to solve such problems was in the range from several tens of seconds to one and a half hour. The high efficiency of these computations is explained by the fact that the major computational effort in the proposed method goes for solving the auxiliary unconstrained maximization problem using the generalized Newton method. The dimension of this problem is determined by the number of equality constraints, and this number is considerably less than the number of nonnegative variables in the original LP.

Table 1 presents the results of the test computations obtained using the program EGM (see [4]), which implements method (22), (23) in MATLAB, and other commercial and research packages. All the problems were solved on a 2.02 GHz Celeron computer with 1 Gb of memory. The following packages were used: BPMPD v. 2.3 (the interior point method, see [8]), MOSEK v. 2.0 (the interior point method, see [9]), and the popular commercial package CPLEX (v. 6.0.1, the interior point and simplex methods).

Table 1 shows the dimensions m and n of the problems, the density ρ of the nonzero entries in the matrix A, the time T needed to solve the LP in seconds, and the number of iterations Iter (for EGM, the total number of systems of linear equations that were solved in the Newton method when problem (23) was solved). Everywhere, we assumed that $\beta = 1$. The Chebyshev norms of the residual vectors were calculated:

$$\Delta_1 = \|Ax - b\|_\infty, \quad \Delta_2 = \|(A^{\mathrm{T}}u - c)_+\|_\infty, \quad \Delta_3 = |c^{\mathrm{T}}x - b^{\mathrm{T}}u|. \tag{24}$$

The results presented in Table 1 showed that the program EGM for MATLAB is close in terms of efficiency to the well-known packages based on the interior point and simplex methods. The third row of Table 1 shows the results for a LP with five millions of nonnegative variables, 1000 constraints, and 1% density of filling the matrix A by nonzero entries. The computation time of EGM was 16 min. The fourth row shows the results in the case of 1000 constraints, a completely filled matrix A, and $n = 10^5$. The computation time in this case was 44 min. Both problems were solved very accurately (the norms of the residuals did not exceed 7.1×10^{-7}). The other packages failed to solve both problems. Therefore, for large LPs, the program EGM implemented in MATLAB for a uniprocessor computer turned out to be considerably more efficient than the other packages. Note that an implementation of the same algorithm in C was approximately by a factor of eight more efficient in terms of computation time.

In order to increase the number of constraints up to hundreds of thousands, it is reasonable to develop a parallel implementation of the proposed method.

4. PARALLEL ITERATIVE ALGORITHMS

We consider parallel implementations of method (22), (23) with the use of the generalized Newton method in the distributed memory model. In this case, each process (program being executed) has an individual address space and the data exchange between processes is performed using the MPI library. It is assumed that each process if executed on a separate processor (core).

4.1. Computation Formulas

First, we write out the computation formulas used in the parallel implementation of iterative method (22), (23) and the generalized Newton method.

Step 1. Set $\beta > 0$, the accuracies for the outer and inner iterations *tol1* and *tol*, respectively, and the initial approximations x_0 and p_0.

Step 2. Calculate the value of the function

$$S(p_k, \beta, x_s) = b^{\mathrm{T}} p_k - \frac{1}{2}\|(x_s + A^{\mathrm{T}} p_k - \beta c)_+\|^2. \tag{25}$$

Here, k is the index of the inner iteration of the Newton method for solving unconstrained maximization problem (23) and s is the index of the outer iteration.

Step 3. Calculate the gradient of function (25) with respect to p:

$$G_k = \frac{\partial S}{\partial p}(p_k, \beta, x_s) = b - A(x_s + A^{\mathrm{T}} p_k - \beta c)_+. \tag{26}$$

Step 4. Using the generalized Hessian of function (25), form the matrix $H_k \in \mathbb{R}^{m \times m}$:

$$H_k = \delta I + A D_k A^{\mathrm{T}}. \tag{27}$$

Here, the diagonal matrix $D_k \in \mathbb{R}^{n \times n}$ is specified by

$$(D_k)_{ii} = \begin{cases} 1, & (x_s + A^{\mathrm{T}} p_k - \beta c)^i > 0 \\ 0, & (x_s + A^{\mathrm{T}} p_k - \beta c)^i \leqslant 0. \end{cases} \tag{28}$$

Step 5. Find the direction of maximization δp by solving the linear system

$$H_k \delta p = -G_k. \tag{29}$$

using the preconditioned conjugate gradient method. The diagonal part of H_k is used as a preconditioner.

Step 6. Determine p_{k+1} by the formula

$$p_{k+1} = p_k - \tau_k \delta p,$$

where the iteration parameter τ_k is found by solving the one-dimensional maximization problem using the bisection or Armijo's method

$$\tau_k = \max_{\tau} S(p_k - \tau \delta p, \beta, x_s).$$

In practice, in all the computations described below, $\tau_k = 1$ specified the desired solution of the local maximization problem so that the search was not required.

Step 7. If the stopping criterion for the inner iterations on k

$$\|p_{k+1} - p_k\| \leq \mathrm{tol}$$

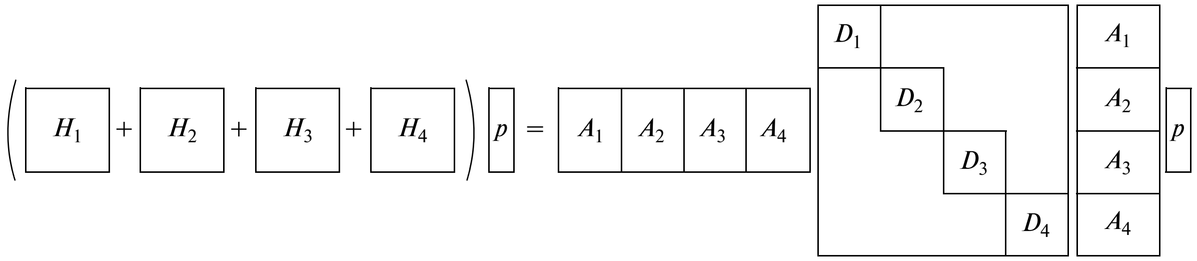


Fig. 1.

is fulfilled, then set $\tilde{p} = p_{k+1}$ and find x_{s+1} by the formula

$$x_{s+1} = (x_s + A^T\tilde{p} - \beta c)_+. \tag{30}$$

Otherwise, set $k = k + 1$ and go to Step 2.

Step 8. If the stopping criterion for the outer iterations on s

$$\|x_{s+1} - x_s\| \le \text{tol}$$

is fulfilled, then find the solution of dual problem (D)

$$u^* = \tilde{p}/\beta.$$

The solution of primal problem (P) is $x^* = x_{s+1}$. Otherwise, set $p_0 = \tilde{p}$, $s = s + 1$, and go to Step 2.

4.2. Main Operations of the Parallel Algorithm

In algorithm 1–8, the main computationally costly operations are the construction of the system of linear equations (29) and its solution. The parallel implementation of the algorithm requires that the following main operations be parallelized:

multiplication of a matrix by a vector: Ax and A^Tp;

calculation of scalar products x^Ty, where $x, y \in \mathbb{R}^n$, and p^Tq, where $p, q \in \mathbb{R}^m$;

construction of generalized Hessian (27);

multiplication of the generalized Hessian by a vector.

Note that all the remaining computations in algorithm 1–8 are local. For example, the conjugate gradient method requires the computation of distributed scalar products and distributed multiplication of a matrix by a vector. The other computations are local and do not require data exchange. For the evaluation of function (25), the distributed multiplication of a matrix by a vector is needed to find the vector

$$z_k = (x_s + A^Tp_k - \beta c)_+$$

and to find the scalar product b^Tp_k. To calculate gradient (26), one more multiplication of a matrix by a vector is needed.

From the viewpoint of the structure of the parallel algorithm, one may assume that an underdeterminate linear system with the matrix A is solved using the least squares method because all the major difficulties of the parallel algorithm appear already at this stage. Various variants of data decomposition for constructing parallel algorithms were discussed, for example, in [10, 11].

4.3. Data Decomposition: Column Scheme

Since the number of rows m in A is much smaller than the number of columns n, a simple column scheme can be used when the number of processors n_p is small. In this scheme, the matrix A is decomposed into block columns A_i of approximately the same size as illustrated in Fig. 1 for the case of four processors.

For the simplicity of the presentation, we assume that n is divisible by n_p. Then, $N_c = n/n_p$. The subvector $x_i \in \mathbb{R}^{N_c}$ and the submatrix $A_i \in \mathbb{R}^{m \times N_c}$ are stored on the processor with the index i. In the description of the data distribution, it is convenient to drop the indexes of the local and the global iteration. In the column scheme, all the vectors of length m, that is, p, b, etc., are duplicated on each processor. As a result,

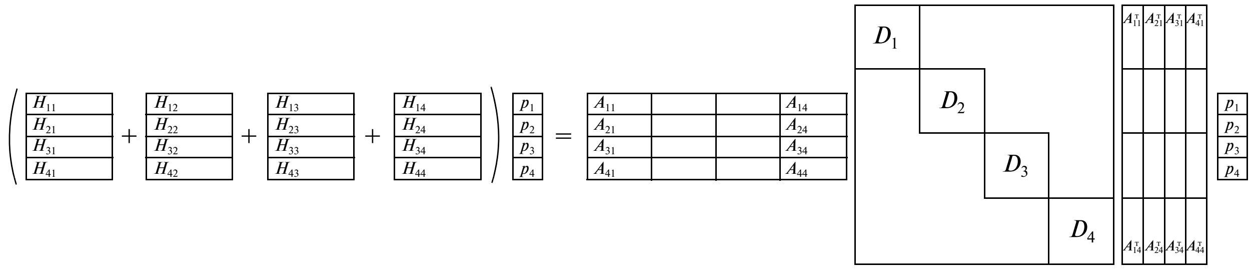

H_{11} H_{21} H_{31} H_{41}
H_{12} H_{22} H_{32} H_{42}
H_{13} H_{23} H_{33} H_{43}
H_{14} H_{24} H_{34} H_{44}
p_1 p_2 p_3 p_4
A_{11} A_{14} A_{21} A_{24} A_{31} A_{34} A_{41} A_{44}
D_1 D_2 D_3 D_4
A_{11}^T A_{21}^T A_{31}^T A_{41}^T
A_{14}^T A_{24}^T A_{34}^T A_{44}^T

Fig. 2.

the multiplication by the transposed matrix, that is, the calculation of the expression A^Tp is local. On the other hand, the multiplication of A by a vector can be written as

$$Ax = \sum_{i=1}^{n_p} A_i x_i.$$

The multiplication of the submatrices by the subvectors x_i is performed independently, and the n_p resulting vectors of length m are added using the function MPI_Allreduce included in the MPI library.

The Hessian H can be written as

$$H = \sum_{i=1}^{n_p} H_i, \quad H_i = A_i D_i A_i^T,$$

where $D_i \in \mathbb{R}^{N_c \times N_c}$ are the diagonal blocks of D. In the column scheme, H is not constructed. The local matrix H_i is computed on the ith processor. To multiply the matrix by a vector q

$$Hq = \sum_{i=1}^{n_p} H_i q,$$

each component H_iq is calculated locally, and the n_p resulting vectors are summed up using the function MPI_Allreduce so that the sum becomes available on all the processors.

As a result, the conjugate gradient method for solving linear system (29) is not parallel, and its computational cost increases proportionally to the number of processors. Denote by γ the ratio of the computational cost of solving the linear system to the other computational costs at a single iteration of the Newton method. Neglecting the time needed to exchange data, we can write an upper bound on the parallel speedup as

$$s(n_p) = n_p \frac{1+\gamma}{1+\gamma n}. \quad (31)$$

Therefore, if $\gamma = 0.05$, then $s(4) = 3.5$ and $s(6) = 4.85$. It is clear that such a simple algorithm is efficient only when the coefficient γ is sufficiently small.

If the actual communication overheads are taken into account, the speedup can be considerably lower.

4.4. Data Decomposition: Cellular Scheme

In order to achieve a better parallelization efficiency, the so-called "cellular" decomposition can be used. In this case, the matrix A is decomposed into rectangular blocks as shown in Fig. 2. This figure illustrates the corresponding decomposition of vectors.

For the simplicity of the presentation, we assume that the total number of processors can be represented as $n_p = n_r \times n_c$; that is, we may arbitrarily assume that the processors are located at the nodes of a lattice of size $n_r \times n_c$, n is divisible by n_c, and m is divisible by n_r. Therefore, $n = n_c \times N_c$ and $m = n_r \times N_r$. The matrix A consists of the submatrices $A_{ij} \in \mathbb{R}^{N_r \times N_c}$. In this scheme, the vector $x \in \mathbb{R}^n$ is decomposed into n_c subvectors x_i and the vector $p \in \mathbb{R}^m$ is decomposed into n_r subvectors p_j; x_i is located simultaneously on n_r processors, and the ith subvector p is duplicated n_c times in the jth row of the lattice of processors.

Consider the main distributed operations for such a decomposition.

(a) The operation $x = A^{\mathrm{T}}p$ is represented in the form

$$x_i = \sum_{j=1}^{n_r} A_{ji}^{\mathrm{T}} p_j, \quad i = 1, 2, \ldots, n_c. \tag{32}$$

At first glance, all the computations are local and, in order to find x_i, the sum of n_r vectors of length N_c must be found using, for example, the function **MPI_Allreduce** included in the MPI library. To find this sum, it is convenient to use the mechanism of splitting the communicators provided by the MPI library. The sums of vectors that reside in different rows of the lattice of processors can be calculated independently. However, if n is very large, the time spent on data exchange is overly large and no speedup is achieved.

A simple solution is to abandon the optimal computational complexity when multiplying the matrix A^{T} by a vector. Assume that, for each j, it is known that the jth processor in the ith row of the lattice of processors contains the entire ith block column of A rather than only the submatrix A_{ij}; that is A is duplicated n_r times. If we assume that the vector p is entirely available on each of the $n_r \times n_c$ processors, the same formula (32) is independently repeated on all the processors belonging to the same column of the lattice. Therefore, the vector x_i is available on all the processors belonging to the ith column of the lattice of processors without extra exchanges. However, for this purpose, the matrix A^{T} is multiplied by the vector $p \times n_r$ times so that the number of multiplications at this stage is close to ρmnn_r.

(b) The operation Ax is almost local because only n_c subvectors of length N_r must be added independently in each of the n_r rows of the lattice of processors.

(c) The computations at the stage of forming the matrix H are completely local and require no data exchange. The number of multiplications at this stage can be estimated as $m^2\rho^2\rho_z n$, where ρ_z is the fraction of nonzero entries in the diagonal matrix D. In this reasoning, we assumed that the nonzero entries are randomly distributed over A, that is, the distribution is uniform on the average.

The assembly result is the sum of n_c matrices, where the jth summand is distributed over the jth column of the lattice of processors, as is seen in Fig. 2.

Let the coefficient γhave the same meaning as in formula (31); that is γ is the ratio of the computational cost of solving the linear system to the other computational costs in a uniprocessor implementation of the algorithm. The coefficient

$$\alpha = \frac{\rho mn}{m^2 n \rho^2 \rho_z} = \frac{1}{m\rho\rho_z} \tag{33}$$

shows an approximate ratio of the cost of multiplying the matrix A by a vector to the cost of forming H in a uniprocessor implementation. Then, we obtain the following very coarse but simple upper bound on the speedup in the case of using the cellular scheme:

$$s_u(n_r \times n_c) = n_r \frac{1 + 2\alpha}{1 + (n_r + 1)\alpha} n_c \frac{1 + \gamma}{1 + \gamma n_c}. \tag{34}$$

In practice, the coefficient α is considerably smaller than that given by formula (33). If we set $\gamma = 0.05$ and $\alpha = 0.1$ in (34), then $s_u(8 \times 8) = 30.3$.

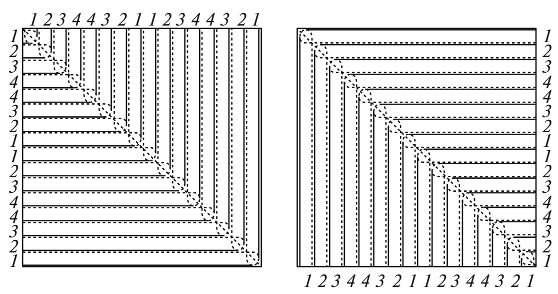
In the analysis of this scheme, we did not consider the question of how to write the whole jth block column of A on every processor in the jth column of the lattice of processors. Indeed, if A is formed locally, that is, if every cell is constructed on one processor, much more time can be needed for such data redistribution than for solving the problem. Since, in this study, we formed the matrices using explicit formulas, this question eluded our attention. Note that it is very difficult to construct optimal data structures in this problem because the matrix A is very sparse and the Hessian can be assumed to be filled almost completely.

4.5. Data Decomposition: Cyclic Mirror Scheme

In the parallel implementation of the cellular scheme of the decomposition of A, we neglected the fact that the generalized Hessian H is symmetric; therefore the cost of its calculation can be almost halved compared with the quadratic matrix of the general form. This cost can be easily reduced in the column scheme; however, an attempt to do the same in the cellular or row scheme results in a misbalance of the processor load both at the stage of constructing H and at the stage of multiplying H by a vector.

To balance the processor load, the well known cyclic mirror decomposition scheme should be used; this scheme is illustrated in Fig. 3 for the 4×4 lattice of processors.

(a)



(b)

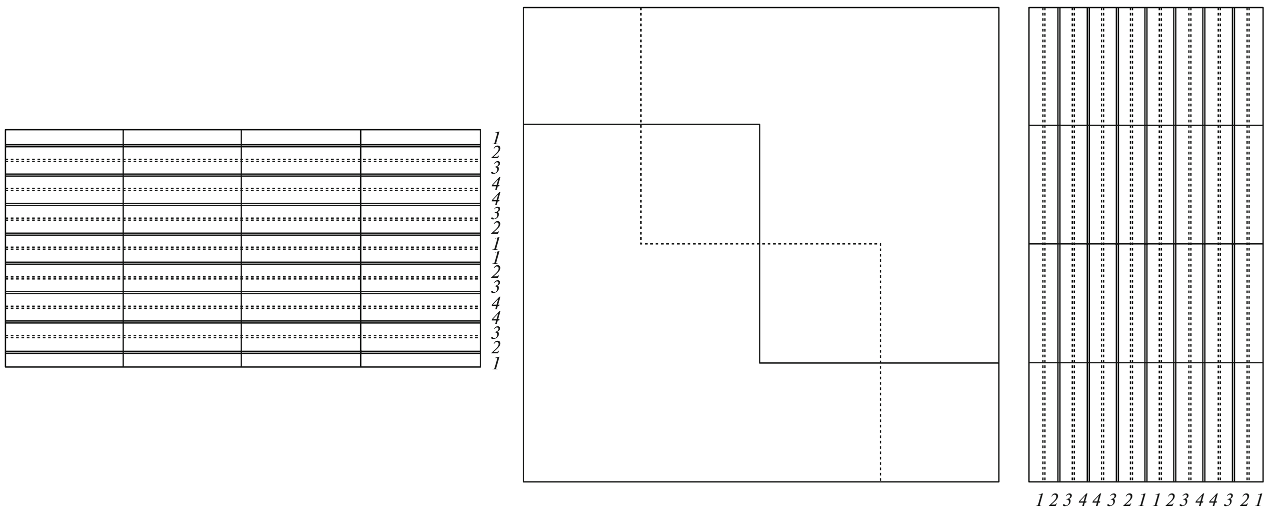


Fig. 3.

Figure 3a shows the variants of the decomposition of the Hessian, and Fig. 3b shows the decomposition of the matrices A, A^T, D, and the vectors.

In this example, we use an analog of the cellular scheme, but the rows of A are decomposed into groups in which every (block) row is assigned to a particular processor; when we pass from a group to the next one, the enumeration of the block rows in the group is changed for the opposite one. To represent H, two storage schemes can be used, which are illustrated in Fig. 3a. For this decomposition, the number of operations per processor when the matrix H is filled and when it is multiplied by a vector is almost constant. This storage scheme was not implemented; rather, it was modeled by a scheme in which H is filled as a general asymmetric matrix. Note that such an approach can lower α almost by a factor of two, and the actual values of the speedup obtained with the cyclic scheme can be lower than those obtained with the cellular scheme even though the computation time in the cyclic scheme can be lower.

4.6. Data Decomposition: Row Scheme

One can consider two very different variants of the implementation of the row scheme of data decomposition, which is illustrated in Fig. 4 for four processors.

The first variant can be considered as a particular case of the cellular scheme in which $n_c = 1$; therefore, a coarse upper bound on the speedup in the case of the row scheme is

$$s_u(n_r \times 1) = n_r \frac{1 + 2\alpha}{1 + (n_r + 1)\alpha}. \tag{35}$$

Therefore, for $\alpha = 0.1$, we obtain $s_u(8 \times 1) = 5.05$. The main drawback of this scheme is that the entire matrix A must be stored on each processor so that the memory required for this algorithm cannot be less than ρmnn_p. The nonoptimality in terms of memory considerably restricts the maximal size of the prob-

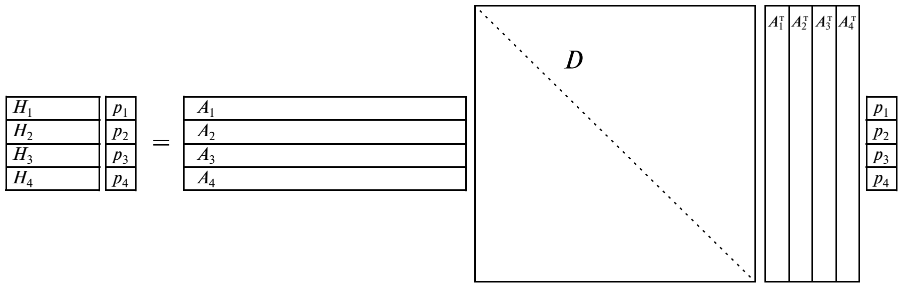


Fig. 4.

lems that can be solved. Hence, we face the problem of designing an algorithm that is optimal in terms of memory when each processor stores only a block row of A.

We implemented the so-called matrix-free algorithm in which the matrix H is not constructed at all; only its main diagonal is found, and the sequence of operations $q = ADA^{\mathrm{T}}p$ or

$$q_i = \sum_{j=1}^{n_p} A_j D A_j^{\mathrm{T}} p_j$$

is used to multiply H by the vector p. Here, the vector p_j is located at the jth processor. In order to avoid the exchange of vectors of length n, the sparsity of the diagonal matrix D is used; this property implies that the number of nonzero components of the vector $DA_j^{\mathrm{T}}p_j$ cannot be greater than in D. Therefore, after packing the sparse vectors, the length of the vectors that each processor transmits to its neighbors can be estimated as $2\rho_z n$, which can be considerably less than n.

Unfortunately, at the stage of constructing D, we need the quantity $A^{\mathrm{T}}p$, so that one cannot avoid one summation of n_p vectors of length n per each iteration of the Newton method. To optimize this operation, one can use the simultaneous execution of computations and data exchanges. One can hardly expect an efficient parallelization from the matrix-free method. Rather, we would like the parallel version of the algorithm not to be much slower than the sequential version. Then, the parallel implementation is much more efficient than the methods that use virtual memory when the required memory exceeds the available RAM.

On the other hand, if A is a very sparse matrix in the sense that H can also be assumed to be sparse, the row scheme can be very efficient although it must be considerably modified. In addition, in the framework of the row scheme, one can efficiently implement the parallel preconditioned conjugate gradient method for solving the linear system; as a preconditioner, the reliable incomplete LU factorization can be used as the preconditioner (see [12]).

5. NUMERICAL RESULTS

For the numerical experiments, we used the generator of random test LPs described in Section 3.

The computations were performed on the cluster MVC-6000IM consisting of two-processor nodes based on 1.6 GHz Intel Itanium 2 processors connected by Myrinet 2000.

Table 2

T, s	s	n_p						
		1	2	4	8	16	32	64
T_{tot}		1439.32	1311.40	568.79	408.94	236.68	188.17	142.65
	s_{tot}	1	1.10	2.53	3.52	6.08	7.65	10.09
T_{lin}		121.88	124.42	122.09	120.60	116.87	109.80	106.17
	s_{lin}	1	0.98	1.00	1.01	1.04	1.11	1.15
T_{rem}		1317.35	1186.69	446.61	288.25	119.73	78.30	36.40
	s_{rem}	1	1.11	2.95	4.57	11.00	16.82	36.19

Table 3

T, s	s	n_p						
		1	2	4	8	16	32	64
T_{tot}		406.66	242.78	121.32	62.02	32.13	18.43	15.49
	s_{tot}	1	1.67	3.35	6.56	12.66	22.07	26.24
T_{lin}		31.47	16.45	8.36	4.4	2.28	1.55	2.09
	s_{lin}	1	1.91	3.77	7.16	13.81	20.24	15.04
T_{rem}		375.13	226.24	112.88	57.53	29.76	16.79	13.32
	s_{rem}	1	1.66	3.32	6.52	12.60	22.35	28.16

Table 4

T, s	s	$n_r \times n_c$					
		1×1	2×2	4×4	8×8	10×10	12×12
T_{tot}		1439.32	502.98	145.13	45.65	36.77	28.21
	s_{tot}	1	2.86	9.92	31.53	39.14	51.03
T_{lin}		121.88	62.17	31.66	15.89	12.94	11.26
	s_{lin}	1	1.96	3.85	7.67	9.42	10.82
T_{rem}		1317.35	440.73	113.43	29.74	23.79	16.9
	s_{rem}	1	2.99	11.61	44.30	55.37	77.94

Table 5

$m \times n \times \rho$	n_p	T_{tot}	Δ_1	Δ_2	Δ_3
$5 \times 10^4 \times 10^6 \times 0.01$	16	400.02	2.1×10^{-6}	2.1×10^{-7}	1.1×10^{-10}
$10^5 \times 10^6 \times 0.01$	20	484.62	8.1×10^{-6}	3.6×10^{-6}	5.6×10^{-11}
$10^5 \times 2 \times 10^6 \times 0.01$	40	823.13	4.5×10^{-6}	4.2×10^{-7}	7.2×10^{-11}
$2 \times 10^5 \times 2 \times 10^6 \times 0.01$	80	2317.42	4.9×10^{-5}	6.6×10^{-6}	5.2×10^{-10}

The results are presented in Tables 2–5. In these tables, T_{tot} denotes the computation time in seconds, T_{lin} is the time spent on solving systems of linear equations (29), T_{rem} is the time spent on the other computations. Furthermore, s_{tot} denotes the parallel speedup in solving LPs, s_{lin} is the parallel speedup in solving linear systems, and s_{rem} is the speedup of the other computations. β was equal to 100, which exceeded β_* in these problems. Everywhere, $\hat{x} = 0_n$; that is, the normal solution was sought in problem (3).

Table 2 presents the results obtained using the column scheme for $m = 10^4$, $n = 10^6$, and $\rho = 0.01$.

Table 3 presents the results obtained using the row scheme for $m = 5000$, $n = 10^6$, and $\rho = 0.01$.

Table 4 presents the results obtained using the cellular scheme for $m = 10^4$, $n = 10^6$, and $\rho = 0.01$.

It is seen from Tables 1–4 that the highest parallel speedup was obtained using the row and cellular data decomposition schemes. The maximum speedup of 51.03 on 144 processors was achieved using the cellular scheme. Note that the low speedup for the column scheme on two processors can be apparently explained by the fact that a part of the computations performed on a single processor was actually executed on two cores due to automatic parallelization.

Table 5 shows the computation time for the matrix-free scheme and the Chebyshev norms of criteria residuals (24) for various problems and various numbers of processors.

The use of the matrix-free scheme enabled us to solve LPs with the greatest number of constraints m compared with the other schemes. For example, a LP with two million variables and two hundred thousand constraints was solved in less than 40 minutes on 80 processors. This scheme uses the cluster's memory most efficiently; it can be recommended for use when m and n are large. For this scheme, the speedup

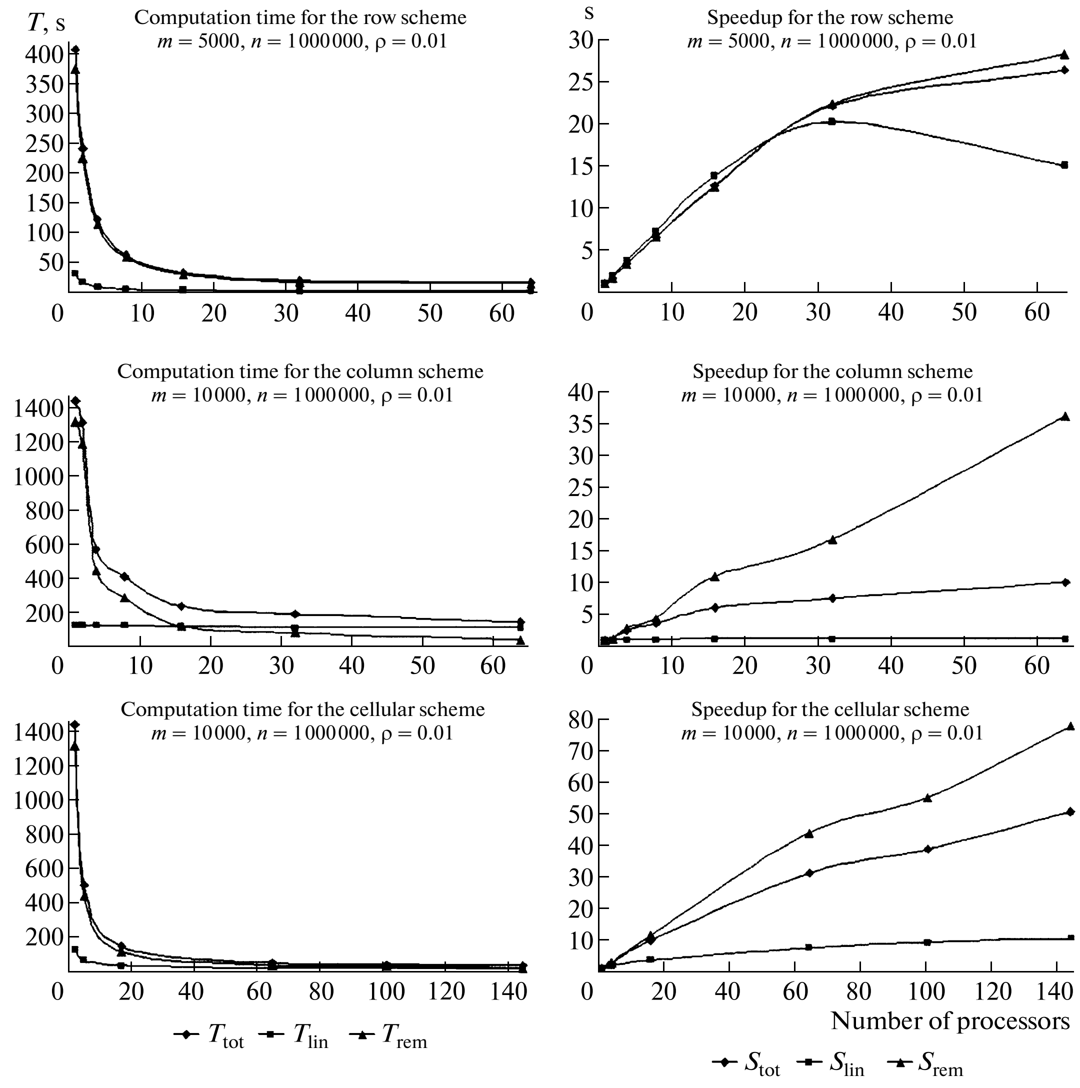


Fig. 5.

for relatively small m can be less than unity. On the other hand, in the most interesting and difficult case of large m, this scheme is very efficient and sometimes gave the speedup greater than unity. In any case, the matrix-free scheme is much more efficient than the methods that use slow disk memory when the random access memory is insufficiently large.

The computation time for typical test problems and the speedup for different data decomposition schemes are shown in Fig. 5.

6. CONCLUSIONS AND THE DIRECTIONS OF FURTHER RESEARCH

In this study, parallel versions of the generalized Newton method for solving linear programs based on various data distribution schemes (column, row, and cellular schemes) were developed. The resulting parallel algorithms were successfully used to solve large-scale LPs (up to several millions of variables and several hundreds of thousands of constraints) for a relatively dense matrix A.

As could be expected, the development of an efficient solver proved to be a very challenging task; for every parallel variant of the algorithm, a reasonable tradeoff between the computational scalability and the scalability in terms of memory had to be found. The highest speedup was obtained for the cellular scheme. However, depending on the dimension of the problem and the sparsity structure of the matrix A, the cellular, row, or column scheme can be optimal.

The cost of constructing the generalized Hessian can be considerably reduced by calculating only a variable correction to the Hessian at each iteration of the Newton method.

The resulting speedup is quite acceptable (about 30 for 64 processors); however, it must be taken into account that the parallel assignment of the input data for the solver is a nontrivial task, and it can take longer time than the parallel solution of the problem.

In the implementation of the parallel algorithm proposed in this paper, shared memory was not used. One can expect that a combination of the algorithms based on a combination of distributed and shared memory using MPI/Open MP will produce more efficient algorithms for modern multicore architectures of computer clusters.

An important factor in the speedup of computations is the optimization of operations on dense and sparse matrices. We used the library Lapack for working with dense matrices. However, the vector and matrix operations on sparse matrices leave additional possibilities for optimization. In particular, one can use special representations of sparse matrices that combine portability and make use of the cache memory characteristics of modern processors (see [13, 14]); such representations can considerably reduce the time needed for vector and matrix multiplications.

ACKNOWLEDGMENTS

This study was supported by the Russian Foundation for Basic Research (project no. 08-01-00619), by the Program of the State Support of Leading Scientific Schools (project no. NSh-5073.2008.1), and by the Program P-2 of the Presidium of the Russian Academy of Sciences.

REFERENCES

1. I. I. Eremin, *Linear Optimization Theory* (Ural'skoe Otdelenie Ross. Akad. Nauk, Yekaterinburg, 1998) [in Russian].
2. F. P. Vasil'ev and A. Yu. Ivanitskii, *Linear Programminge* (Faktorial, Moscow, 2003) [in Russian].
3. A. I. Golikov and Yu. G. Evtushenko, "Solution Method for Large-Scale Linear Programming Problems," Dokl. Akad. Nauk **397** (6), 727–732 (2004) [Dokl. Math. **70**, 615–619 (2004)].
4. A. I. Golikov, Yu. G. Evtushenko, and N. Mollaverdi, "Application of Newton's Method for Solving Large Linear Programming Problems," Zh. Vychisl. Mat. Mat. Fiz. **44**, 1564–1573 (2004) [Comput. Math. Math. Phys. **44**, 1484–1493 (2004)].
5. A. I. Golikov and Yu. G. Evtushenko, "Search for Normal Solutions in Linear Programming Problems," Zh. Vychisl. Mat. Mat. Fiz. **40**, 1766–1786 (2000) [Comput. Math. Math. Phys. **40**, 1694–1714 (2000)].
6. C. Kanzow, H. Qi, and L. Qi, "On the Minimum Norm Solution of Linear Programs," J. Optimizat. Theory Appl. **116**, 333–345 (2003).
7. O. L. Mangasarian, "A Newton Method for Linear Programming," J. Optimizat. Theory Appl. **121**, 1–18 (2004).
8. Cs. Meszaros, "The BPMPD Interior Point Solver for Convex Quadratic Programming Problems," Optimizat. Meth. Software **11**, 431–449 (1999).
9. E. D. Andersen and K. D. Andersen, "The MOSEK Interior Point Optimizer for Linear Programming: An Implementation of Homogeneous Algorithm" in *High Performance Optimization* (Kluwer, New York, 2000), pp. 197–232.
10. G. Karypis, A. Gupta, and V. Kumar, "A Parallel Formulation of Interior Point Algorithms," Proc. Supercomputing, 204–213 (1994).
11. T. F. Coleman, J. Czyzyk, C. Sun, et al., "pPCx: Parallel Software for Linear Programming," in *Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing, PPSC, 1997* (SIAM, 1997).
12. I. E. Kaporin, "High Quality Preconditioning of a General Symmetric Positive Definite Matrix Based on Its UTU+UTR+RTU-Decomposition," Numer. Linear Algebra Appl. **5**, 483–509 (1998).
13. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-Oblivious Algorithms," in *Proc. 40th Ann. Symposium on Foundations of Computer Science, New York, 1999*, pp. 285–297.
14. G. S. Brodal, "Cache-Oblivious Algorithms and Data Structures," in *Proc. 9th Scandinavian Workshop on Algorithm Theory,* Lect. Notes Comput. Sci. **3111**, (Springer, Berlin, 2004), pp. 3–13.