An. Şt. Univ. Ovidius Constanţa Vol. 11(2), 2003, 127–136

OPTIMIZATION ALGORITHM FOR FINDING SOLUTIONS IN LINEAR PROGRAMMING PROBLEMS

Ioan Popoviciu

Abstract

When speaking about linear programming problems of big dimensions with rare matrix of the system, solved through simplex method, it is necessary, at each iteration, to calculate the inverse of the base matrix, which leads to the loss of the rarity character of the matrix. The paper proposes the replacement of the calculus of the inverse of the base matrix with solving through iterative parallel methods a linear system with rare matrix of the system.

General Presentation

Linear programs for big real systems are characterized by rare matrices, having a low percentage of non-zero elements. The rare character appears at each base matrix, but disappears at the inverse of this matrix. In its classical form, the simplex method uses a square matrix, the inverse of the base matrix, whose value is putting up-to-date at each iteration. The number of non-zero elements of the inverse matrix increases rapidly and depends on the number of iterations. Because of this, in the place of the calculus of the rare matrix, one can solve the linear systems with a rare matrix of the system through iterative parallel methods.

Let's take the linear programming problem in the standard form:

$$Ax = b,\ x \geqslant 0 \tag{1}$$

$$\max\left(f\left(x\right) = c^{T}x\right), \tag{2}$$

where A is a matrix with m lines and n columns, $x \in R^n$, $b \in R^m, c \in R^n$.

At each iteration, one takes a base, meaning a square matrix of order m, which can be inverted, extracted from the matrix A, denoted with A',where

$I \subset N, |I| = m$. We associate a basic solution to the base I defined by: $x_I^B = \left(A^I\right)^{-1} b,\ x_{\overline{I}}^B = 0$, where $\overline{I}$ is the complement of I in N.

The bases which are being successively generated through simplex method are of the type $x_I^B \geq 0$, meaning the basic solutions considered are all admissible (they fulfill the conditions (1) and (2)).

An iteration consists of a change of the base I into an adjacent base I'; this is a base obtained through the changing of the index $r \in I$ with the index $s \in I$, $I' = I - r + s$.

To determine r and s, one has to calculate:

$$u = f^I \left(A^I\right)^{-1},\ \ d^{\overline{I}} = f^{\overline{I}} - uA^{\overline{I}}, \tag{3}$$

where $u, f^I, d^{\overline{I}}$ are row vectors. This allows that $s \in \overline{I}$ is selected by the condition $d^s > 0$. Then:

$$x_I^B = \left(A^I\right)^{-1} b,\ \ T^s = \left(A^I\right)^{-1} a^s, \tag{4}$$

where a^s is the column number s of the matrix A, and x_I^B, b, T^s, a^s are column vectors. One obtains $r \in I$ through condition:

$$\frac{x_r}{T_r^s} = \min\left\{\frac{x_i}{T_i^s} \mid i \in I, T_i^s > 0\right\}. \tag{5}$$

When the values r and s are determined, it follows the updating of the inverse of the base matrix , meaning that $\left(A^{I'}\right)^{-1}$ is determined. This is obtained from the relation:

$$\left(A^{I'}\right)^{-1} = E_r(\eta)\left(A^I\right)^{-1} \tag{6}$$

where $E_r(\eta)$ is the matrix obtained from the unit matrix of order n, by replacing the column e^r with the vector expressed by:
$\eta = \left(-\frac{c_1}{c_r}, \ldots, -\frac{c_{r-1}}{c_r}, \frac{1}{c_r}, -\frac{c_{r+1}}{c_r}, \ldots, -\frac{c_n}{c_r}\right)$, where $c = \left(A^I\right)^{-1} a^s$.

In this way, the mathematical equations are represented by the relations (4-6), and the inverse of the base matrix appears in the relations (3) and (4). The relations (3), (4) can be replaced by:

$$uA^I = f^I \tag{3'}$$

$$A^I x_I^B = b,\ A^I T^s = a^s. \tag{4'}$$

In the first equation, the matrix is the transpose of the base matrix; in the last two equations, even the base matrix A appears and consequently these two systems benefit from the rare character of matrix A, an efficient solution being possible through iterative parallel methods.

The parallel algorithm of the conjugated gradient.

In order to solve linear systems of large dimensions of the type ($3'$) and ($4'$) we are going to present a parallel implementation of the algorithm of the conjugated gradient, a method where, in the first place, one has to do a multiplication between a rare matrix and a parallel vector.

Let's take the product $y = Ax$, where A is a rare matrix $n \times n$, and x and y are vectors of n dimension. In order to accomplish a parallel execution of the product $y = Ax$, one has to perform a partitioning of the matrix A into a matrix distributed over many processors. In this view, a subset of the components of the vector x and consequently a subset of the rows of the matrix A are being allocated to a processor so that the components of vectors x and y can be divided into three groups:

- **internal** are those components which belong (and consequently are calculated) to the processor and do not take part into the communication between the processors. We say in consequence that y_j is an **internal component** if it is calculated by the processor to whom it belongs and if the index j of the column corresponding to the element a_{ij} different from zero in the line i corresponds to a component x_j which also belongs to the same processor;

- **border set** are those components which belong (and by consequence are calculated) to the processor, but they require a communication with other processors in order to calculate them. Thus, we may say that y_j is a **border set component** if it is calculated by the processor which it belongs to and if at least one column index j associated to the non-void elements a_{ij} from line i, corresponds to a component x_j which does not belong to the processor;

- **external** are those components which do not belong to (and by consequence are calculated) by the processor, but which correspond to column indexes associated to non-zero elements from the rows belonging to the processor.

In conformity with this organisation, there corresponds to each processor a vector whose components are ordered as follows:

- the first components are numbered from 0 to $N_i - 1$, where N_i is the number of internal components ;

- the next components are border set components and occupy the positions from N_i to $N_i + N_f - 1$, where N_f is the number of border set components;

- the last components are external components and occupy the positions comprised between $N_i + N_f$ and $N_i + N_f + N_e - 1$, where N_e is the number of external components.

Within this vector associated with a processor, the external components are being ordered so that those which are used by the processor occupy successive positions.

For example let's take $A(6,6)$ and suppose that x_0, x_1, x_2 and by consequence the rows $0, 1, 2$ of matrix A are allocated to the processor 0; x_3 and x_4 and by consequence the rows 3 and 4 are allocated to the processor 2; x_5 and by consequence rows 5 are allocated to the processor 1. The matrix A has the non-zero elements marked by a * in the following description.

$$A = \begin{array}{ll} & \\ & 0 \\ \mathit{proc}.0 & 1 \\ & 2 \\ \mathit{proc}.2 & 3 \\ & 4 \\ \mathit{proc}.1 & 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{cccccc} * & * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{array} \right) \end{array}$$

For processor 0 which has the rows 0, 1, 2 attached to the matrix A and respectively the components x_0, x_1, x_2, we have:

$N_i = 1$: a sole internal component y_0 because in calculating y_0 it appears only those that belong to the x_0, x_1, x_2 processor 0.

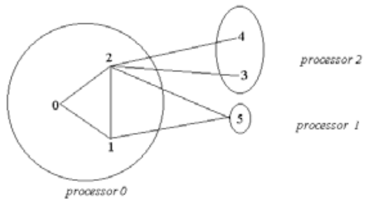
$N_f = 2$: two border set components y_1 and y_2 in whose calculus the elements belonging to other processors also appear:

- in the calculus of y_1, x_5 also appears; this belongs to the processor 1.

- in the calculus of y_2, x_5, x_3, x_4 there also appears; x_5 belongs to the processor 1 while x_3, x_4 belong to the processor 2.

$N_e = 3$: three external components because in the calculus of y_0, y_1, y_2 there appear three components x_5, x_3, x_4 which belong to other processors.

The communication graph corresponding to the processor 0 is defined in the following picture:



To the rows 0, 1, 2, the following vectors correspond, vectors in which the indices of the columns corresponding to the external components are grouped and sorted into processors:

Line		the indices of columns with non-zero elements			
0	$\longrightarrow$	0	1	2	
1	$\longrightarrow$	1	0	2	5

2 $\longrightarrow$ 2 0 1 5 3 4

Each processor has to acknowledge on which of the processors the external components are calculated: in the above example, processor 1 calculates the component y_5 and processor 2 calculates the components y_3 and y_4. At the same time, each processor has to acknowledge which of its internal components are being used by other processors.

Let's remind the schematic structure of the algorithm CG:

```
$x =$ initial value
$r = b - Ax$ ...rest
$p = r$ ...initial direction
repeat
    $v = A * p$
        ...multiplication matrix-vector
    $a = (r^T * r)/(p^T * v)$
        ...product "dot"
    $x = x + a * p$
        ...update solution vector
        ...operation "saxpy"
    $new\_r = new\_r - a * v$
        ...update rest vector
        ...operation "saxpy"
    $g = (new\_r^T * new\_r)/(r^T * r)$
        ...product "dot"
    $p = new\_r + g * p$
        ...update new direction
        ...operation "saxpy"
    $r = new\_r$
until ($new\_r^T * new\_r$ smaller)
```

It is noticed that the following operations are necessary in the algorithm CG:

1. A product rare matrix-vector;
2. Three vector updatings (operations "SAXPY");
3. Two scalar products (operations "DOT");
4. Two scalar dividings;
5. A scalar comparison for the testing of the convergence.

For the parallel implementation of the algorithm CG, the following distinct parts appear:

a) Distribution of the date on processors

The date are being distributed on processors on lines so that each processor has a consecutive number of lines from the rare matrix assignated:

typedefstructtag_dsp_matrix_t

```
{
    int N;                 /* dimension matrix $N \times N$ */
    int row_i, row_f;      /* rank of beginning and ending line which belongs to the processor*/
    int nnz;               /* number of non-void elements from the local matrix */
    double* val;           /* elements of the matrix */
    int* row_ptr;          /* beginning of a matrix */
    int* col_ind;          /* column index*/
} dsp_matrix_t;
```

Each processor will store the rank of the lines belonging to it, the elements of the matrix and two pointers row_ptr and col_ind used in the storing of the compressed on lines of a matrix.

b) In/out operation

In/out operations comprise the reading of the matrix and its stiring in a compressed lines format.

c) Operations on vectors

The operations on vectors are of two types:

- operations "saxpy" for updating of the vectors, which do not require communication between processors;
- operations "dot" (scalar product) which do not require communication between processors with the help of function MPI_Allreduce.

d) Multiplication matrix-vector

Each processor uses at the calculus the rows of the matrix which belong to it, but needs elements of the vector x which belong to other processors. This is why a processor receives these elements from the other processors and sends at the same time its part to all the other processors. In this way, we can write schematically the following sequence:

```
$new\_element\_x = my\_element\_x$
for $i = 0, num\_proces$
    - send $my\_element\_x$ to the processor $my\_proc + i$
    - calculate locally with $new\_element\_x$
    - receive $new\_element\_x$ from the processor $my\_proc - i$
repeat.
```

The optimisation of the communication

A given processor does not need the complete part of x which belongs to other processors, but only the elements corresponding to the columns which contain non-zero elements. At the same time it sends to the other processors only the non-zero elements of x. This is the reason why the structure presented above comprises the field *col_ind* which indicates the rank of the column that contains a non-zero element. In this way, we can schematically write the following sequence:

- each processor creates a mask which indicates the rank of the columns of non-zero elements from A;

- communication between processors:

```
$new\_element\_x = my\_element\_x$
    for $i = 0, num\_proces$
        - if communication necessary between $my\_proc$ and $my\_proc + i$
            - transmit $my\_element\_x$ to the processor $my\_proc + i$
        endif
        - calculate locally with $new\_element\_x$
        - receive $new\_element\_x$ from the processor $my\_proc - i$
    repeat.
```

The algorithm implemented for a matrix of the dimension $N \times N = 960 \times 960$, with 8402 non-zero elements has given the following results:

Number of processors	Number of iterations	Calculus duration	Duration of commu nication between processors	Time for the memory allocation	Time for opera tions with vectors	Total duration
1	205	3.074	0.027	0.002	0.281	3.384
2	205	2.090	0.341	0.002	0.136	2.568
4	204	1.530	0.500	0.002	0.070	2.110

* time is expressed in minutes.

The analysis of the performance of the algorithm CG.

The analysis of the performance of the algorithm CG is done from the point of view of the time necessary for the execution of the algorithm. In this model the initiation times for the matrix A and of the other vectors used is neglectable. At the same time, the time necessary for the verification of the convergence is disregarded and it is presupposed that the initializations necessary for an iteration have been done.

A) Analysis of the sequential algorithm

Notations:

m=vectors dimension

N=total number of non-zero elements of matrix A

k=number of iterations for which the algorithm is executed

T_{comp1s}=total calculus time for the vectors updating (3 operations SAXPY)

T_{comp2s}=total calculus time for the product A_p and for the scalar product (r, r)

T_{comp3s}=total calculus time for the scalar products(A, A_p) and (p, A_p)

T_{comp4s}=total calculus time for the scalars α and β

T_{seq}=total calculus time for the sequential algorithm.

Then $T_{\text{seq}} = T_{\text{comp1s}} + T_{\text{comp2s}} + T_{\text{comp3s}} + T_{\text{comp4s}}$.

Within the algorithm there are three operations SAXPY, each vector being of dimension m. If we suppose that T_{comp} is the total calculus time for the multiplication of two real numbers with double precision and for the adding of the results, then T_{comp2s} is the total calculus time for the product of rare matrix-vector and for the two scalar products. The product matrix-vector implies N elements and the scalar product implies m elements. Then $T_{comp2s} = (N + m) * T_{comp}$.

T_{comp3s} is the calculus time of two scalar products and can be written as $T_{comp3s} = 2 * m * k * T_{comp}$.

The calculus for the scalars α and β implies two operations of division and a subtraction of real numbers. Let's take $T_{comp\alpha}$ the calculus time for all these operations. Then $T_{comp4s} = 2 * k * T_{comp\alpha}$.

The total calculus time for the sequential algorithm CG is:

$$T_{\text{seq}} = (6 * m + N) * T_{\text{comp}} + 2 * k * T_{\text{comp}\alpha}$$

B) Analysis of the parallel algorithm

Within the parallel algorithm each processor executes k iterations of the algorithm in parallel. We define:

b = dimension of the block from matrix A and from vectors x, r, p belonging to each processor; p = number of processors;

T_{comp1p} = total calculus time for the vectors updating on each processor;

T_{comp2p} = total calculus and communication time for the A_p and (r, r);

T_{comp3p} = total calculus time for the calculus of the scalar products and of the global communication; T_{comp4p} = total calculus time for the scalars α and β

Here T_{comp1p} is the total time for the calculus of $3b$ vectors updating. If the matrix A is very rare (the density is smaller than 5 percentages) the communication time exceeds the calculus time. This way T_{comp2p} is taken equal with T_{comm}, the communication time of a block of dimension b to all the p processors. T_{comp3p} implies the global calculus and communication time, noted with t_{glb}. Then:

$$T_{par} = T_{comp1p} + T_{comp2p} + T_{comp3p} + T_{comp4p}, \text{ where:}$$

$T_{comp1p} = 3 * b * k * T_{comp}$; $T_{comp2p} = T_{comm}$;

$T_{comp3p} = 2 * b * k * T_{comp} + T_{glb}; T_{comp4p} = 2 * k * T_{comp\alpha}$.

Therefore, to estimate T_{seq} and T_{par}, it is necessary to estimate the values of $T_{comp}, T_{comp\alpha}, T_{comp}$ and T_{glb}.

REFERENCES

1. Axelsson O., *Solution of liniar systems of equations: iterative methods,* In: Barker [1] 1-51.

2. Bank R, Chan T., *An analysis of the composite step biconjugate gradient method,* 1992.

3. Golub G, Van Loan., *Matrix computations,* Notrh Oxford Academic, Oxford, 1983.

4. Ostrovski, A.M., *On the linear iteration procedures for symmetric matrices*, Rend. Math. Appl., **14**(1964), 140-163.

Department of Mathematics-Informatics
"Mircea cel Batrân" Naval Academy
Fulgerului 1, 900218, Constantza, Romania