*Joint NCC & IIS Bull.*, Comp. Science, 4 (1996), 17–26
© 1996 NCC Publisher

# On parallelization of new algorithm for solving systems of linear equations*

M. Balandin, O. Chernyshev, E. Shurina, A. Vazhenin

Solving of large-scale systems of linear algebraic equations, ill-conditioned and non-square systems requires special algorithms which should be suitable for effective implementation in modern multi-processor systems. One of such methods was proposed by A.A. Abramov. This approach is based on consequential projecting of solution to a system of ortogonal vectors that are chosen by such mean that projections can be easily found. In this paper we discuss the main features and possible modifications of the Abramov method from the viewpoint of effective parallel implementation. The parallel Abramov algorithm and some results of its testing are also presented.

## 1. Introduction

Solving of large-scale systems of linear algebraic equations (SLAE), ill-conditioned and non-square SLAEs requires special algorithms which should be implemented in modern multi-processor computers. One of such methods is the one proposed by A.A. Abramov in [1, 2]. Thus, we shall name it the Abramov method (AM). This approach is based on consequential projecting of solution to some system of orthogonal vectors which are chosen by such mean that projections can be easily found. Note also, that this approach belongs to a class of conjugative directions methods [3]. Such methods provide an ability to obtain solution with a number of iterations less or equal than the size of SLAE having the square matrix. The AM also allows this ability giving fixed precision of solution.

The AM-algorithm can be defined by choosing coefficients for constructing linear combination from SLAE equations. For example, when we choose these coefficients as components of the right part vector, then an iterative method will be obtained. On the other hand, if we use a set of unit vectors, then we will have a direct algorithm.

In this paper we discuss the main features and possible modifications of the Abramov method from the viewpoint of effective parallel implementation. The parallel AM-algorithm and some results of its testing are also presented.

*Supported in part by the Russian Foundation for Basic Research under Grants 95–01–0135 and 96–01–01557.

## 2. General notes

We consider the system of linear equations

$$Ax = b, \tag{1}$$

where $A$ is $n \times m$ matrix, $b$ and $x$ are $n \times 1$ and $m \times 1$ vectors respectively. We shall denote rows of $A$ as $a_1, a_2, \ldots, a_n$. Evidently, each vector $a_i$ has dimension $1 \times m$.

Let us search solution $x$ in subspace $S$ defined by vectors $a_1$, $a_2, \ldots, a_n$ as their linear shell of $x \in S = L(a_1, \ldots, a_n)$. If $A$ is a square matrix of full rank, then $S$ will equal to $R^n$.

From $n$ equations of initial system (1) we can construct a linear combination in which the $i$-th equation is included with the coefficient of $b_i$. It is similar to multiplication of the initial system (1) to vector $b$ by

$$d^T x = \varphi, \tag{2}$$

where

$$d^T = b^T A, \qquad \varphi = b^T b. \tag{3}$$

The solution of $x$ must satisfy to (2). This allows to find a projection of $x$ to one-dimensional subspace defined by vector $d$. Let $x = x_d + x'_d$, where $x_d$ is projection to be found, and a component of $x'_d$ is ortogonal to $x_d$. It is evidently that $x_d = \gamma d$ and $d^T x'_d = 0$. Then we have

$$d^T x = \varphi = d^T(x_d + x'_d) = d^T x_d = \gamma d^T d.$$

Therefore,

$$\gamma = \frac{\varphi}{\psi}, \qquad \psi = d^T d. \tag{4}$$

Projection has been found. Now we decompose $S$ as $S = L(d) \oplus S'$ and in the next steps we shall take directions for projecting from $S'$.

To reduce $S$ to $S'$ ($\dim S' = \dim S - 1$), we shall orthogonalize each of $a_1, \ldots, a_n$ vectors to the vector $d$ and obtain vectors $\tilde{a}_1, \ldots, \tilde{a}_n$ forming the matrix of $\tilde{A}$. To save the solution of system (1), it is necessary to i.nplement similar transformations of the right part vector $b$ to vector $\widetilde{b}$.

Therefore, for $i = 1, 2, \ldots, n$ we shall execute the following transformations:

$$\tilde{a}_i = a_i - \alpha_i d^T, \tag{5}$$

$$\widetilde{b}_i = b_i - \alpha_i \varphi. \tag{6}$$

Coefficients $\alpha_i$ can be defined from condition $\tilde{a}_i d = 0$ by $\tilde{a}_i d = 0 = (a_i - {}_i d^T)d = a_i d - \alpha_i d^T d$. Therefore,

$$\alpha_i = \frac{a_i d}{\psi}. \tag{7}$$

Implementing the similar calculations for the matrix $\tilde{A}$, right part $\tilde{b}$ and subspace $S'$, we can obtain new direction $d$ and projection of solution to $d$, reduce $S'$ to $S''$, etc. The subspace $S$ has a dimension not greater than $n$. Therefore, we can select only $\leq n$ ortogonal vectors of $d$, and the total number of similar iterations is $\leq n$. Moreover, we can found a number of iterations by $r = \operatorname{rang}(S)$. The sum of projections found is the solution $x$ of (1).

Process must be stopped when $\|d\| < \varepsilon$. Because $\|d\|$ is defined by vector $b$, i.e.,

$$\|d\| = \|b_1 a_1 + b_2 a_2 + \ldots + b_n a_n\| \leq |b_1|\,\|a_1\| + \ldots + |b_n|\,\|a_n\|,$$

condition $\|b\| < \delta$ (recommended in [1]) can be also used.

According to (2)–(7), we can describe the AM-algorithm as follows:

$$\begin{aligned}
&A^0 = A, \quad b^0 = b,\\
&d^k = (A^k)^T b^k, \quad \varphi_k = (b^k)^T b^k, \quad \psi_k = (d^k)^T d^k, \quad x^{k+1} = \frac{\varphi_k}{\psi_k} d^k, \\
&\alpha_i^k = \frac{a_i^k d^k}{\psi_k}, \quad a_i^{k+1} = a_i^k - \alpha_i^k (d^k)^T, \quad b_i^{k+1} = b_i^k - \alpha_i^k \varphi_k, \qquad i = \overline{1,n}.
\end{aligned} \tag{8}$$

Stop when $(b^p)^T b^p < \delta$, $p \leq n$. The result is $x = x^1 + x^2 + \ldots + x^p$.

This method can be decsribed in terms of vectors and matrix operations. We denote as $\bar{\alpha}$ vector composed orthogonalizing coefficients: $\bar{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$. Then (5), (6) equal to:

$$\tilde{A} = A - \bar{\alpha} d^T, \tag{9}$$

$$\tilde{b} = b - \bar{\alpha}\varphi. \tag{10}$$

Taking into account (2), (4) and (7) we can define $\alpha_i = \frac{a_i A^T b}{b^T A A^T b}$. Therefore,

$$\bar{\alpha} = \frac{1}{b^T A A^T b} A A^T b. \tag{11}$$

When substitute (8) and (2) we get:

$$\begin{aligned}
\tilde{A} &= A - \frac{1}{b^T A A^T b} A A^T b b^T A = \Big(I_n - \frac{1}{b^T A A^T b} A A^T b b^T\Big) A,\\
\tilde{b} &= b - \frac{1}{b^T A A^T b} A A^T b b^T b = \Big(I_n - \frac{1}{b^T A A^T b} A A^T b b^T\Big) b.
\end{aligned}$$

Denoting

$$D = I_n - \frac{1}{b^T AA^T b} AA^T bb^T, \tag{12}$$

we can write:

$$\tilde{A} = DA, \tag{13}$$

$$\tilde{b} = Db. \tag{14}$$

Analogously, substituting (2) and (4) in $x_d = \gamma d$, we find

$$x_d = \frac{b^T b}{b^T AA^T b} A^T b. \tag{15}$$

Then in terms of vectors and matrices the AM-algorithm is as follows:

$$\begin{aligned} &A^0 = A, \quad b^0 = b, \\ &x^{k+1} = \frac{(b^k)^T b^k}{(b^k)^T A^k (A^k)^T b^k} (A^k)^T b^k k, \\ &D^k = I_n - \frac{1}{(b^k)^T A^k (A^k)^T b^k} A^k (A^k)^T b^k (b^k)^T, \\ &A^{k+1} = D^k A^k, \quad b^{k+1} = D^k b^k, \end{aligned} \tag{16}$$

Stop when $(b^p)^T b^p < \delta$, $p \leq n$, The result is $x = x^1 + x^2 + \ldots + x^p$.

## 3. Merits and deficiencies

Most evident merit of the AM is universality that allows to solve any SLAEs (with only condition that they have solution). Except this, method does not require initial value for $x$. The main deficiency is necessity to recalculate extended matrix of system at each iteration. This property firstly required additional time and secondly does not allow to operate with matrices stored in sparse formats (or makes such work very difficult). Time-expenses can be compensated by using computing systems, but work with sparse matrices seems very problematic (because it requires transferring of memory fragments and/or special searching algorithms).

We want to pay attention to one property of the AM that can be important when implementing this method. Direct using of formulas of (8) on initial iterations can results in growth of $|\psi|$. For example, for very simple system

$$\begin{aligned} 2x_1 + 10x_2 + x_3 &= 13, \\ 100x_1 + 7x_3 &= 107, \\ 4x_1 + 3x_2 + 9x_3 &= 16, \end{aligned} \tag{17}$$

having solution $x_1 = x_2 = x_3 = 1$ formulas of (8) on first iteration, give $d^1 = (1079, 178, 906)^T$ and $\psi = 117276620$, i.e., $\psi$ has 8-th order although maximal element of matrix – only 2-nd. Since $\psi$ is used as divisor, its big order can result in lost of accuracy.

This disadvantage can be avoided using other coefficients (see Section 4) or scaling (i.e., dividing every row on appropriately chosen number).

Each component of $d^k$ is formed as sum of $a^k_{ij} b^k_i$ by $i$. Number $\psi^k$ is sum of squares of components of $d^k$. So that scaling can be used for rows which satisfy to the following criteria

$$\max_{j=\overline{1,m+1}} (a^k_{ij} b^k_i)^2 > M \qquad (a^k_{im+1} \equiv b^k_i),$$

or

$$\max_{j=\overline{1,m+1}} |a^k_{ij} b^k_i| > \sqrt{M}. \tag{18}$$

Here $M$ is a constant defined by maximal absolute real value in computational system.

Scaling coefficient must be chosen so that it does not result in lost of order for least numbers in row. This value can be chosen as

$$\frac{1}{2}\Big( \max_{j=\overline{1,m+1}} |a^k_{ij}| + \min_{j=\overline{1,m+1}} |a^k_{ij}| \Big) \qquad (a^k_{im+1} \equiv b^k_i).$$

After scaling system (11) will be changed to system

$$\begin{aligned} 2x_1 + 10x_2 + x_3 &= 13, \\ 1.862x_1 + 0.1308x_3 &= 2, \\ 4x_1 + 3x_2 + 9x_3 &= 16. \end{aligned} \tag{19}$$

For (11) $d^1 = (83.7384; 178; 157.26616)$ and $\psi_1 = 63427.3305$.

## 4. Parallel implementation of Abramov's method

Properties of method do not allow to calculate $x^k$'s independently because after each iteration recalculation of matrix is required. So that we can parallelize only calculation of every $x^k$. In the next section we shall consider possibilities of parallelization of other modifications of method.

On each iteration of the AM main computational work is to recalculate extended matrix of system. In this procedure each row is recalculated independently but orthogonalizing vector is common for all rows. It suggests that method can be parallelized by decomposition of matrix $A$ to $Np$ submatrices each containing $N_i$ rows $(N_1+N_2+\ldots+N_{Np}=n)$ and vector $b$ to $Np$ subvectors with dimensions $N_i$. To provide good load-balancing it is suitable to chose $N_i$ as close as possible to $n/Np$. Each subsystem $A_{(i)}x=b_{(i)}$ can be worked out on one processor. Since vector $x$ does not directly participates in computations, it can be stored only at one computational node. On the same node vector $d^k$ composed by components $d^k_{(i)}=(A^k_{(i)})^T b^k_{(i)}$ must be collected and then sent to all other nodes ($d^k$ is need to recalculate all $A_{(i)}$'s and $b_{(i)}$'s). For computations organized by such mean the most suitable topology of processors network is "star". For this topology, the central node is connected to each other, but every peripheral node is connected only to central one.

Assuming that data was distributed as described above, in accordance to formulas of 8 we can write parallel algorithm as follows (MASTER means central node, SLAVE means peripheral one).

MASTER

```
1. Initialization $k=1$; $A^k_{(i)} := A_{(i)}$; $b^k_{(i)} := b_{(i)}$.
2. [Scaling $A^k_{(i)}$and $b^k_{(i)}$].
3. Find $\varphi^k_{(i)} = (b^k_{(i)})^T b^k_{(i)}$.
4. Collect $\varphi^k_{(i)}$ from all SLAVEs and find sum $\varphi^k = \sum_{i=1}^{Np} \varphi^k_{(i)}$.
5. Send $\varphi^k$ to all SLAVEs.
6. If $\varphi^k < \delta$ STOP. Result is $x = \sum_{j=1}^{k-1} x^j$.
7. Find $d^k_{(i)} = \sum_{j=1}^{N_i} b^k_{(i)j} a^k_{(i)j}$.
8. Collect $d^k_{(i)}$ from all SLAVEs and find sum $d^k = \sum_{i=1}^{Np} d^k_{(i)}$.
9. Send $d^k$ to all SLAVEs.
10. Find $\psi^k = (d^k)^T d^k$.
11. Find $x^k = \frac{\varphi^k}{\psi^k} d^k$.
12. For $j = \overline{1, N_i}$ do:
        $\alpha^k_{(i)j} = \frac{1}{\psi^k} a^k_{(i)j} d^k$;  $a^{k+1}_{(i)j} = a^k_{(i)j} - \alpha^k_{(i)j}(d^k)^T$;  $b^{k+1}_{(i)j} = b^k_{(i)j} - \alpha^k_{(i)j}\varphi^k$.
13. $k := k+1$. Go to step 2.
```

SLAVE

```
1. Initialization $k = 1$; $A^k_{(i)} := A_{(i)}$; $b^k_{(i)} := b_{(i)}$.
2. [Scaling $A^k_{(i)}$and $b^k_{(i)}$].
3. Find $\varphi^k_{(i)} = (b^k_{(i)})^T b^k_{(i)}$.
4. Send $\varphi^k_{(i)}$ to MASTER.
5. Recieve $\varphi^k$ from MASTER.
6. If $\varphi^k < \delta$ STOP. Result is $x = \sum_{j=1}^{k-1} x^j$.
7. Find $d^k_{(i)} = \sum_{j=1}^{N_i} b^k_{(i)j} a^k_{(i)j}$.
8. Send $d^k_{(i)}$ to MASTER.
9. Recieve $d^k$ from MASTER.
10. Find $\psi^k = (d^k)^T d^k$.
11. For $j = \overline{1, N_i}$ do:
      $\alpha^k_{(i)j} = \frac{1}{\psi^k} a^k_{(i)j} d^k;\quad a^{k+1}_{(i)j} = a^k_{(i)j} - \alpha^k_{(i)j}(d^k)^T;\quad b^{k+1}_{(i)j} = b^k_{(i)j} - \alpha^k_{(i)j}\varphi^k.$
12. $k := k + 1$. Go to step 2.
```

In this algorithm square brackets [ ] mean optional step.

One can see that all nodes perform similar calculation except central which also collects $\varphi^k$, $d^k$ and calculates projections of solution to $d^k$'s.

It is easy to estimate the volume of communications for presented algorithm. At each iteration MASTER and SLAVE exchange by two real numbers ($\varphi^k_{(i)}$ and $\varphi^k$). Then SLAVE sends $n$ real numbers (vector $d^k_{(i)}$) and receives also $n$ (vector $d^k$). Let $v$ be a size of real number (in bytes), then at one iteration communication volume for each connection is equal to

$$(2 + 2n)v \text{ bytes.}$$

Multiplying this number to $(Np - 1)$ connections, we find that total volume at one iteration is

$$2(Np - 1)(n + 1)v \text{ bytes.} \tag{20}$$

Knowing communication speed for connection and using (14), we can estimate required time.

## 5. Possible modifications

In described variant of the AM current direction for projecting is chosen as linear combination of matrix rows with multiplier equal to coefficients of right part. It is evidently that in general case we can choose these multipliers arbitrary:

$$d^T x = \varphi, \tag{21}$$

where

$$d^T = f^T A, \qquad \varphi = f^T b. \tag{22}$$

Of course, vector $f$ with dimensions $n \times 1$ must provides condition $\|d\| \neq 0$, when $\dim S > 0$.

The most simple case is consequential using of vectors from set $\{e_1, e_2, \ldots, e_n\}$, where $e_i = (0_1, 0_2, \ldots, 0_{i-1}, 1_i, 0_{i+1}, \ldots, 0_n)^T$. Choosing $f = e_j$ we can see that $d^T x = \varphi$ is equal to $j$-th equation of system 1, i.e., $a_j x = b_j$.

Hence, in this case calculation of $d$ and $\varphi$ is not required. Moreover, orthogonalization of $j$-th row to itself leads to zero result, so that at next iterations we can except it from system. By this mean, number of rows to be recalculated at next iteration is reduced by 1. The iterative method becomes direct one and a number of row recalculations equals to

$$n + (n-1) + (n-2) + \ldots + 2 = \frac{1}{2}(n+2)(n-1), \tag{23}$$

instead of $n^2$. But we have to check each equation for condition $\|d\|^2 + \varphi^2 > \varepsilon$, or, the same

$$a_j a_j^T + b_j^2 > \varepsilon. \tag{24}$$

Parallel implementation of such variant with using described above algorithm is available but leads to many superfluous communications (when all rows of $A$ have been zeroised it becomes unnecessary). We can present the following scheme for "pipeline" topology.

Data distribution among nodes is the same as was described in previous section. We start calculation from the left side of pipeline. Leftmost processor takes first row of its submatrix, sends it (with according coefficient of $b$) to the right and starts to ortogonalize other rows. Then it finds projection of solution and adds to $x$ vector (that is initially equal to zero). Each node on the right from the current one receives the direction vector and sends it to the right neighbor then recalculates its own submatrix. Leftmost processor works out 2nd 3rd etc rows of its submatrix analogously. When all rows exhausted, leftmost processor sends to the right collected $x = x^1 + \ldots + x^{N_1}$ and release itself. Second from left processor works by the same mean and so on. When calculations on rightmost node will be finished, its vector $x$ is a result.

## 6. Conclusions

To practically investigate Abramov's method (in variant described in [1]), the program was developed for the Power X'plorer computer of PARSYTEC. This system consists of the Sun-5 computer as a host and 8 processor nodes. Each node contains transputer of T820 (for communication) and the PowerPC-601 processor. As the tests we used a set of matrices: non-square – both not fully defined and over-defined, and square – both badly conditioned and of deficient rank (see [4]).

Results we got show that the AM gives general solution with high precision for $r = \text{rang}(S)$ or less iterations. The hypothesis about growth of $\psi$ was confirmed but we found that it does not influence on solution. Also we found that using of condition $\|d^k\| < \delta$ as criteria of completion is not enough. There is a big possibility of situation when $\varphi_k = \|b^k\| \gg 0$, but $\|d^k\| \simeq 0$. In some tasks it led to overflows and abnormal terminations. Besides, there were cases when application of AM for solving two systems with the same matrix and right parts which were different (but not significantly) gave correct solution for one system and overflow for other. Using additional criteria $\|d^k\| < \varepsilon$ allowed to avoid such situations and, when program was modified, we got right results.

As case of ill-conditioned systems we used the Gilbert matrices with coefficients defined as $a_{ij} = \frac{1}{i+j+1}$ of size in range 10×10 to 50×50. Consequentially 1, 2 and 4 processors had been used. Exact solution was $x = (1, 1, \ldots, 1)^T$. The results achieved are the following. Speed of solution for all numbers of processors was very fast. Therefore, the execution time could not be measured exactly by usual means (we obtained values less than one, one, and two seconds). For matrix 50×50 with $\varepsilon = \delta = 10^{-15}$ we got the solution after 8 iterations with precision $10^{-3}$, and with $\varepsilon = \delta = 10^{-30}$ - after 12 iterations with precision $10^{-5}$. To measure speedup exactly very large systems must be used that will be the next step of our work.

In the nearest future we suppose to investigate variants of AM with arbitrary-chosen coefficients of $f$ and select most suitable scaling procedure. Also we are going to create version of program implementing the AM which uses SPARTH library (see [5]) supporting the parallel computations with an arbitrary precision.

## References

[1] A.A. Abramov, *On one method for solving ill-conditioned systems of linear algebraic equations*, Journal of Computational Mathematics and Mathematical Physics, **30**, No. 4, 1991, 483–491 (in Russian).

[2] A.A. Abramov, *On features of Kreyg's method for solving linear ill-conditioned problems*, Journal of Computational Mathematics and Mathematical Physics, **35**, No. 1, 1995, 144–150 (in Russian).

[3] James M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[4] V.N. Faddeeva, L.J. Kolotilina, *Computational Methods of Linear Algebra. Set of matrices for testing*, Materials on Software, Leningrad, 1982 (in Russian).

[5] A. Vazhenin, *Efficient high-accuracy computations in massively parallel systems*, Proc. "Workshop on Parallel Scientific Computing PARA94-L", 1994, Lyngby, Denmark. Springer-Verlag, 1994, Lecture Notes in Computer Science, **879**, 505–519.