International Journal of Innovation in Science and Mathematics
Volume 11, Issue 2, ISSN (Online): 2347–9051

A Parallel Implementation of SOR Method

Xiaorong Zhu[1*] and Yumei Huang[2]
[1]College of Information Science and Technology, Taishan University,Taian, China.
[2]College of Information Science and Technology, Taishan University, Taian, China.
*Corresponding author email id: smile_nine@126.com

Date of publication (dd/mm/yyyy): 31/03/2023

Abstract **– Jacobi iteration and SOR iteration are the basic methods for solving linear equations, but the appearance of parallel computers makes people notice that they have significant differences in parallel processing performance. Jacobi iteration has very obvious intrinsic parallel computing characteristics, while SOR intrinsic parallelism is far worse than Jacobi iteration. In this paper, we give the PPSOR iteration and prove that it has the performance of full parallelism, so we find a parallel implementation of SOR.**

Keywords **– Iterative Method, SOR Method, Parallel Implementation, Equations, Parallel Point SOR.**

I. INTRODUCTION

Many problems in natural science and engineering technology can be reduced to solving linear algebraic equations, such as network problems in electricity, ship mathematical lofting problems, curve fitting of experimental data, etc. The coefficient matrices of these equations can be roughly divided into low-order DENSE matrices (order less than 150) and large sparse matrices (matrix order higher and more zero elements) [1].

Iterative method is an important numerical method for solving large sparse matrix equations. The so-called iterative method is to approach the exact solution of linear equations gradually by some iterative process. It has the advantages of less memory units of computer, simple program design and constant original coefficient matrix in the calculation process, but there are some shortcomings in convergence and convergence speed. Iteration methods include Jacobi iteration Method and Successive Over Relaxation Method (SOR method for short). Among them, Jacobi iteration has very obvious intrinsic parallel computing characteristics [2.3.4]. SOR method has simple calculation formula and easy program design. It consumes less computer memory, but needs to select the best relaxation factor and its inherent parallelism is far less than Jacobi iteration. This paper tries to find a suitable parallel implementation of SOR.

II. BASIC METHODS

A. *Jacobi Iteration Method*

With equations $\sum_{j=1}^{n} a_{ij} x_j = b_i (i = 1, 2, \cdots n)$, and its matrix form to

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

Among them A is nonsingular matrix and $a_{ii} \neq 0 (i = 1, 2, \cdots n)$, and split A into $A = D - L - U$, then

$$D = \begin{pmatrix} a_{11} & & 0 \\ & \ddots & \\ 0 & & a_{nn} \end{pmatrix}, \quad L = -\begin{pmatrix} 0 & & \\ a_{21} & & \\ a_{31} & \ddots & \\ \vdots & & \\ a_{n1} & a_{n2} \cdots & 0 \end{pmatrix}, \quad U = -\begin{pmatrix} 0 & a_{12} \cdots & a_{1n} \\ & \ddots & a_{2n} \\ & & \vdots \\ & & 0 \end{pmatrix}.$$

The i equation in equation (1) is removed by a_{ij} and then transferred to obtain the equivalent system of equations.

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j), i = 1, 2, \cdots n. \tag{2}$$

Record briefly $x = B_0 x + f$ and $B_0 = I - D^{-1}A = D^{-1}(L+U)$, $f = D^{-1}b$. By applying the iterative method to equation set (2), the Jacobi iterative formula of solution (1) is obtained as follows:

$$\begin{cases} x^0 = (x_1^0, \cdots x_n^0)^T \\ x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^k) \end{cases} \tag{3}$$

where, $x^0 = (x_1^0, \cdots x_n^0)^T$ is the initial vector and $x^k = (x_1^k, \cdots x_n^k)^T$ is the vector of the k iteration. If x^k has been calculated, the next iteration vector $x^{k+1} = (x_1^{k+1}, \cdots x_n^{k+1})^T$ $k = 0, 1, 2, \cdots; i = 1, 2, \cdots n$ can be calculated from equation (3). The matrix form of iteration formula (3) is as follows, and B_0 is the Jacobi iteration matrix.

$$\begin{cases} x^0, \ starting \ variable \\ x_i^{k+1} = B_0 x^{k+1} + f \end{cases} \tag{4}$$

The advantage of Jacobi iterative method is that the formula is simple, each iteration only needs to calculate a matrix and vector product. In the use of computer implementation, need to be stored in the two groups work unit x^k and x^{k+1}.

B. *Overrelaxed Iterative Method (SOR Method)*

Consider system (1), and the decomposition of A is as above. Let's say we have the k iteration vector and the component $x_j^{k+1}(j = 1, 2 \cdots i-1)$ of the $k+1$ iteration vector x^{k+1}, and we want to compute the component x_i^{k+1}. First we define the auxiliary quantity.

$$\tilde{x}_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k), \ k = 0, 1, 2 \cdots; i = 1, 2, \cdots, n. \tag{5}$$

Then take x_i^{k+1} as the weighted average of x_i^k and $\tilde{x}_i^{k+1}$, that is

$$x_i^{k+1} = (1-\omega)x_i^k + \omega\tilde{x}_i^{k+1} = x_i^k + \omega(\tilde{x}_i^{k+1} - x_i^k) \tag{6}$$

Substituting (5) into (6), the successive overrelaxation iteration formula for solving the system of equations $Ax = b$ can be obtained as follows:

$$\begin{cases} x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k) \\ x_i^k = (x_1^k, x_2^k \cdots x_n^k)^T (k = 0, 1, 2 \cdots; i = 1, 2, \cdots n) \end{cases} \tag{7}$$

where ω is called the relaxation factor. The above equation can also be written as follows:

$$\begin{cases} x_i^{k+1} = x_i^k + \Delta x_i \, (k = 0,1,2\cdots; i = 1,2,\cdots n) \\ \Delta x_i = \dfrac{\omega}{a_{ii}} (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k) \end{cases}$$

In the following, we give the matrix form of SOR method. Iteration formula (7) can also be written as $a_{ii} x_i^{k+1} = (1-\omega) a_{ii} x_i^k + \omega (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k)$ $(i = 1,2,\cdots n)$. From the decomposition of equation $A = D - L - U$,

We get $Dx^{k+1} = \omega(b + Lx^{k+1} + Ux^k) + (1-\omega)Dx^k$, that is $(D - \omega L)x^{k+1} = ((1-\omega)D + \omega U)x^k + \omega b$ Obviously, for any value of ω, $D - \omega L$ is not singular, so we have $x^{k+1} = (D - \omega L)^{-1}((1-\omega)D + \omega U)x^k + \omega(D - \omega L)^{-1} b.$ That is, the SOR iteration formula for solving equation (1) is $x^{k+1} = L_\omega x^k + f,$ where $L_\omega = (D - \omega L)^{-1}((1-\omega)D + \omega U), f = \omega(D - \omega L)^{-1} b$ which is called the SOR method's iteration matrix.

In SOR iteration method, every iteration, the main computation is to compute a matrix and vector multiplication. When $\omega < 1,$ (7) is called low relaxation method, and when $\omega > 1,$ (7) is called super relaxation method. Choosing the appropriate relaxation factor is a way to achieve the acceleration of SOR method, but the selection of the best relaxation factor is generally obtained by numerical practice. The appearance of parallel computer makes people start to seek the method of accelerating the iteration from another Angle [5, 6]. Jacobi iteration has very obvious intrinsic parallel characteristics because its components are independent of each other. The calculation of each component in SOR method is related one by one, and its intrinsic parallelism is far worse than Jacobi iteration. However, since SOR method is mostly used for solving large sparse matrix equations, it is worth considering to find a parallel implementation of SOR by using the special distribution of zero or non-zero elements of coefficient matrix.

III. Full Parallelism Scheme

Assuming that the coefficient matrix is dense, then similar to the Jacobi method, the operations involved in SOR iteration can be organized as parallel computation procedures with full degree of parallelism. Here, full parallelism means that the degree of parallelism is equal to the order of the system of equations to be solved [2].

The SOR iteration for solving the linear system $AX = g (A = [a_{ij}]_{n\times n}, g = [g_1, ..., g_n]^T)$ of order n is

$$\begin{cases} \hat{x}_i^{(k+1)} = \sum_{j=1}^{i-1} -a_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} -a_{ij} x_j^{(k)} + g_i / a_{ii} \\ x_i^{(k+1)} = x_i^{(k)} + \omega(\hat{x}_i^{(k+1)} - x_i^{(k)}) \end{cases} \quad (8)$$
$$k = 0,1,...; i = 1,2,...$$

Its matrix is represented as $(D - \omega L)x^{(k+1)} = [(1-\omega)D + \omega U]x^{(k)} + \omega g,$ and $A = D - L - U, L, U$ is a strictly lower and upper triangular matrix. It is usually considered to solve the above triangular equations in parallel, and the basic idea is to combine two successive iteration steps of SOR.

Let $\bar{U} = \omega(D^{-1}U - I), \bar{L} = \omega D^{-1} L, f = \omega D^{-1} g,$ where I is the identity matrix, then (1) can be rewritten as a rectangular form: $X^{(k+1)} = X^{(k)} + UX^{(k)} + LX^{(k+1)} + f,$ so $X^{(k+2)} = X^{(k+1)} + UX^{(k+1)} + LX^{(k+2)} + f$ in k+1, k+2 two successive iterations, there are two lower and upper triangular matrix vector products $\bar{L}X^{(k+1)}$ and $\bar{U}X^{(k+1)}$,

they can combine n n-dimensional vector number multiplication operation $x_i^{(k+1)}B_i$ (i = 1,... ,n), in which, $B=\bar{L}+\bar{U}$ where B_i are the i column of B.

So let's say that we have,

$$Y=(I+\bar{U})X^{(k)} \tag{9}$$

Design the following cyclic iterative process: Add $n+1$ vector $f, x_1^{(k+1)}B_1, x_2^{(k+1)}B_2, ..., x_n^{(n)}B_n$ to Y one by one, and this process gradually produces $y_i = x_i^{(k+1)}$ (i = 1, 2... ,n) and $Y=(I+\bar{U})x^{(k+1)}$, and the latter just prepares the initial value for the next cycle. The complete formula of (9) is

$$\begin{cases} Y^{(0,n+1)}=(I+\bar{U})X^{(0)} \\ Y^{(k,1)}=Y^{(k-1,n+1)}+f \\ Y^{(k,i)}=Y^{(k,i-1)}+Y_{i-1}^{(k,i-1)}B_{i-1} \end{cases} \quad i=2,3,...,n,n+1,\ k=1,2,... \tag{10}$$

Iteration (10) is all composed of multiplication and addition of n-dimensional vectors, and obviously its parallelism degree is always n. We call (10) the full parallelism iteration of SOR, denoted as PPSOR (Parallel Point SOR). Before starting (10), the formation of matrix B and vector f can also be formed by n-dimensional vector operations:

$$\begin{cases} Y=[-\omega/a_{11}, -\omega/a_{22}, ..., -\omega/a_{nn},]^T \\ B_i=Y*A_i, i=1,2,...,n \\ f=-Y*g \end{cases} \tag{11}$$

where A_i is the i column of coefficient matrix A, and the operation * is the product of the corresponding components. According to (10) and (11), the parallelism degree of the proposed parallelization scheme is identical to the order n of the system of equations, except that the initial value $Y^{(0,n+1)}$ is calculated by the product operation of triangular matrix and vector. If implemented on vector computer, it can make full use of the special "link" function of vector operation. If $H=[f,B]$ is set on $R^{n\times(n+1)}$ and H_i is the i column $(i=0,1,2,...,n)$, $y_0^{(l)}=1(l=0,1,2,...)$, the initial value $Y^{(0)}=(I+\bar{U})X^{(0)}$, then PPSOR (10) can be expressed in a more concise form mathematically: $Y^{(l+1)}=Y^{(l)}+y_{l \bmod(n+1)}^{(l)}H_{l \bmod(n+1)}, l=0,1,...$. Step k of PPSOR (10) produces step k iteration $X^{(k)}$ of (8), that is, $y_i^{(k,i)}=x_i^{(k)}, i=1,2,...,n, k=1,2,...$, which means diagonal elements of a matrix $\left[Y^{(k,1)}, Y^{(k,2)}, ..., Y^{(k,n)}\right]$ of order n form the n components of $X^{(k)}$.

Since step k of PPSOR produces step k iteration $X^{(k)}$ of (8), the parallelization design does not reduce the convergence speed of SOR. However, compared with SOR (8), the parallel scheme introduces redundant operation on the total amount of four operations, which can obtain the operation amount $T_1=2k_0n(n+1)$ of (1), the operation amount $T_2=2n^2(k_0+1)+k_0n$ of PPSOR (3), and the redundant operation amount $T_2-T_1=(2-\frac{k_0}{n})n.$ If there are p processors (let's assume that p can divide n), the theoretical speedup ratio

S and efficiency E are respectively $S=\frac{T_1}{(n/p)(T_2/n)}=\alpha p, E=\frac{S}{p}=\alpha,$ in which $\alpha=\frac{T_1}{T_2}=\frac{1+\frac{1}{n}}{1+\frac{1}{k_0}+\frac{1}{2n}}\approx\frac{k_0}{k_0+1}=\beta(k_0),$

$\beta(k_0)$ is strictly increasing with respect to k_0, so the parallel speedup of PPSOR is quite satisfactory, and the larger the number of iteration steps k_0, the more obvious the speedup and the higher the efficiency[7].

As can be seen from $\alpha=\frac{T_1}{T_2}=1-\frac{T_2-T_1}{T_2}$, the reason why the speedup ratio is less than the number of processors p is that the parallelization design introduces redundant computation amount T_2-T_1. Theoretically, the redundant computation amount $T_2-T_1=(2-\frac{k_0}{n})n$ can be negative, that is, the number of iteration steps k_0 can be greater than twice the order n of the system of equations, in which case the speedup ratio can be greater than p. From the above analysis, it can be seen that PPSOR still has good parallelization computing efficiency even for small k_0.

IV. Conclusion

In this paper, two iterative algorithms for solving large linear equations are introduced and their shortcomings are analyzed. In view of SOR iterative methods, which are usually considered to be unfavorable to parallel computing, the PPSOR iterative formula is given and proved to have the performance of full parallelism, and the expression of the formula is concise and clear. Thus, a parallel implementation of SOR is obtained to some extent.

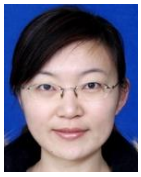
Acknowledgment

This research was supported by the Shandong Provincial Natural Science Foundation, China (ZR2020MF035).

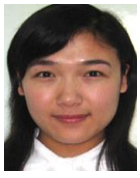
References

[1] Li Qingyang, Wang Nengchao, Yi Dayi. Numerical analysis. Wuhan: Huazhong University of science and technology press, 2012.
[2] Li Xiaomei, Jiang Zengrong. Parallel Algorithm. Changsha: Hunan Science and Technology Press, 1992.
[3] Wen Ruiping, Duan Hui. A class of preconditioned parallel multi-splitting SOR iterative methods for h-matrix system of linear equations [J]. Applied Mathematics, 2020, 33(04): 814-825.
[4] WU Ruihuan. A New pretreatment method of SOR [J]. Journal of Taiyuan Teachers College (Natural Science Edition), 2019, 18(02): 9-11+15.
[5] Grama, Ananth, Anshul Gupta and George Karypis. Introduction to parallel computing. 2d ed. Harlow, England: Addison-Wesley, 2013.
[6] James.M.Ortega. Introduction to parallel and vector solution of linear systems. New York: Plenum Press, 1988.
[7] Michael J. Quinn. Programming in C with MPI and Open MP. 2014.

Author's Profile



First Author
ZHU Xiaorong, (1979-), Female, Master, Associate Professor, Taishan University, Research Interest: Mathematics Education. (Tai 'an 271000, Shandong, China).



Second Author
Yumei Huang, is a lecturer at Taishan University. She obtained her master's degree from Shandong University of Science and Technology in July, 2008. Her research interests are in the areas of applied mathematics and mathematics education in recent years.