One out-of-core block solving method of large-scale linear equations

Xiaofei Xu* *and Zongqing* Wu

State Key Laboratory of Astronautic Dynamics, Xi'an Satellite Control Center, Xi'an, China

Keywords: Out-of-Core solving, Linear equations, Block gauss elimination.

Abstract. The huge amount of memory is needed for solving the large-scale linear equations. However, many problems can not be calculated due to the limitation of the computer's physical memory. To solve the above problem, One Out-of-Core solving method using the hard disk instead of memory is proposed in this paper. The data are read and written concentratively by blocks, thus the solving time of the problem can be greatly shortened. The experiments indicate that the method can solve the larger-scale linear equations efficiently, the solving time is only 5.7% more than In-Core method.

1 Introduction

Large-scale scientific computing involves the access and processing of massive data. However, due to the limitation of the memory capacity of the computer, only part of the data can be read into the memory at a certain time to participate in the calculation, and then written back to the hard disk after calculation. Since the data is not all placed in the internal memory during the operation, it is called Out-of-Core calculation [1].

The computer system divides all storage systems into four levels from the inside to the outside, namely registers, caches, internal memories and external memories. Their access speeds are successively decreased, and the storage capacity is sequentially increased.

Figure 1 depicts the specific process of the Out-of-Core calculation method. After the data is read from the hard disk and processed by the computer, the data is directly stored in the corresponding file on the hard disk.

Solving large-scale linear equations requires a lot of physical memory. The literature [2, 3] introduces an iterative algorithm for linear equations. By using preconditioned processing, faster convergence can be obtained. But they are all based on the in-core solution method of memory, but for some large-scale calculations, computer memory can not meet the needs of problem solving. Several parallel solving methods for linear equations are introduced in [4-9]. Use computer clusters to expand memory, but to build a parallel environment requires a certain cost. This paper proposes an extra-core solution method that uses inexpensive hard

* Corresponding author: 5801267@qq.com

disk storage instead of memory to effectively solve the problem of insufficient memory. In order to speed up the solution time and make full use of computer memory, a block Gaussian elimination method is proposed, which greatly speeds up the problem solving. Experiments show that the Out-of-Core solution method proposed in this paper can effectively solve large-scale linear equations with good stability.

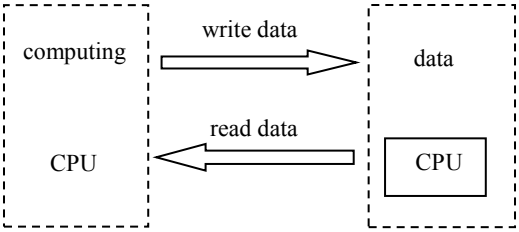
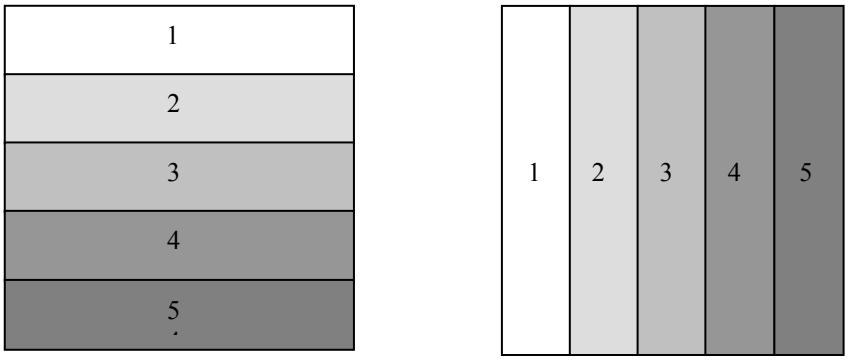


Fig. 1. Out-of-Core calculation method caption.

2 Out-of-Core solution method design

For the linear equations AX=B, the matrix A is first partitioned into the corresponding file on the hard disk. If the entire matrix is written to a single file, it will cause the stored file to be too large, which will affect the data access speed. The block mode of matrix element storage is shown in Figure 2. There are two ways, one is to divide the block by matrix row, and the other is to block by matrix column.



(a) Block by row (b) Block by column

Fig. 2. Blocking mode of matrix storage.

When the matrix element is written to the corresponding file, the direct reading and writing technique of the file is used. The meaning of direct access to the file is: the space and content of the file are divided into several small modules of the same size in advance, and these modules are automatically numbered sequentially. When reading and writing files, you must first assign the read and write position to the first module, and then read and write. Direct access to the file can be read and written to any location in the file. The program opened a direct read file using the OPEN command below:

OPEN (UNIT=FILEID, FILE=FILENAME, ACCESS="DIRECT", FROM="UNFORMATTED", RECL=2, STATUS="REPLACE")

The ACCESS=""DIRECT" in the above command and the RECL field defining the

module size cannot be omitted. When ACCESS="DIRECT", the default value of FROM is "UNFORMATTED". Binary files are used for recording, which saves storage space. The program format for reading and writing files directly is:

READ (FILEID, REC=N, IOSTAT=ERROR) Z

WRITE (FILEID, REC=N, IOSTAT=ERROR) Z

FILEID is the ID number of the file to be written, REC is the position of the read-write file N, Z is the matrix element to be read or written. After calculating a matrix element, it is written to the corresponding position of the file, until the whole The matrix elements are filled.

Based on the Gaussian elimination method[8], according to the characteristics of column elimination, the method of column-by-column is adopted. The Gaussian elimination method is used to solve the matrix equation, which is the process of transforming the matrix equations AX=B into the upper triangular matrix equations.

$$a_{ji} = a_{ji} + a_{ii}\left(\frac{-a_{ji}}{a_{ii}}\right) = 0 \tag{1}$$

The specific elimination process is shown in Figure 3.

The program for Gaussian elimination is as follows:

```
DO I=1,E_NUM !loop over rows, E_NUM the of matrix elements
DO J=I+1,E_NUM !loop over rows,find the max element row
READ() ... !read the data from the file on the hard disk
......
ENDDO
DO J=I,E_NUM !loop over columns, exchange rows
READ()..... !read the data from hard disk
......
WRITE()..... !write the data into the file after exchanging
ENDDO
DO J=I+1,E_NUM! Loop over rows, computethe product factors
READ()..... !read the data from the file on the hard disk
......
ENDDO
DO J+I,E_NUM !loop over columns, elimination
READ()..... !read the data form hard disk
......
WRITE()..... !write the data into the file after elimination
ENDDO
ENDDO
```

One element in each processing matrix needs to be read and written once, which results in a long time spent reading and writing data. Aiming at this problem, a method of block Gaussian solution is proposed. The data is read and written in blocks, which makes full use of computer physical memory, which greatly shortens the data reading and writing time, thus effectively speeding up the problem solving time.

The whole elimination process can be divided into n steps (n is the number of blocks divided by the matrix). In the i-th step, the ith block impedance element is first read into the memory and stored in the array A(J, K), and the column is eliminated. At the same time, the maximum principal row number and product factor of each column are stored in the arrays Max(k) and Each_k(j, k). After the i-th block is finished, the corresponding file is written to the hard disk. Then, the element of the i+1th block is read into the memory and stored in the

array A(j, k), and the elimination is performed according to the elimination process of the i-th block, and the maximum principal element row number and the multiplication factor are already stored in the array Max(k).) and Each_k (j, k), after the elimination processing, the i+1th block is written to the corresponding file of the hard disk. Until the last block is processed, the i-th step ends.

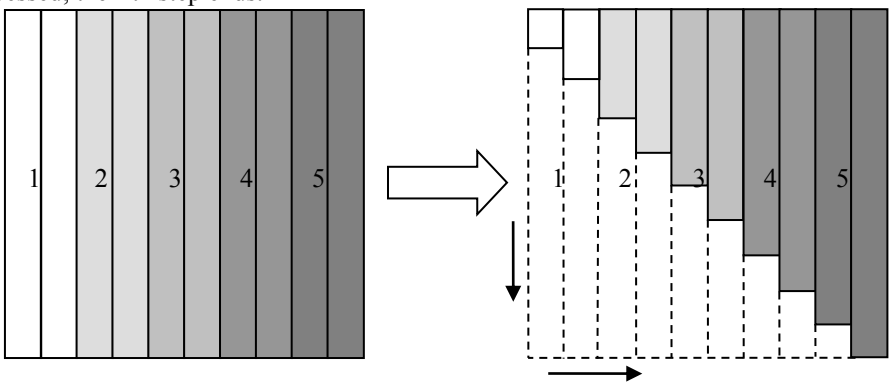


Fig. 3. Convere to an upper triangle.

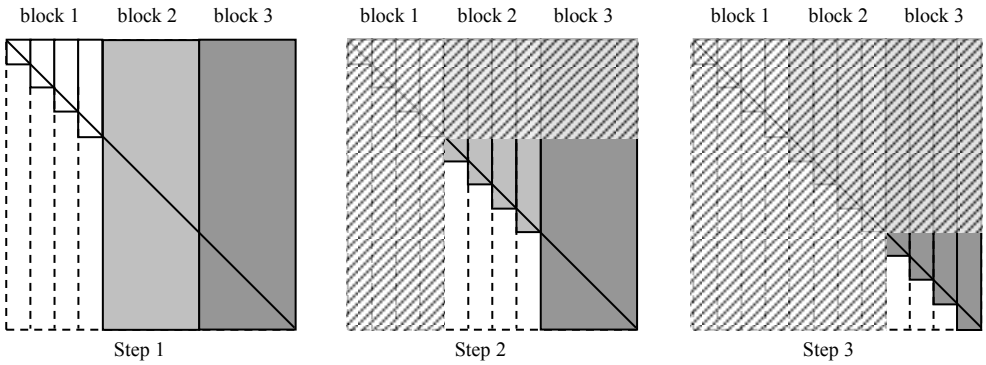


Fig. 4. Block Gaussian elimination process.

Figure 4 is a schematic diagram of a block Gaussian elimination process in which the unshaded area is the element that needs to be processed at each step. As the number of steps increases, the elements that need to be processed at each step are gradually reduced.

In order to reduce data traffic, the matrix is further transformed into a diagonal matrix, avoiding the iterative process. The upper triangular elimination process is similar to the lower triangular elimination process, with the opposite direction. The difference is that after reading the i-th block, the multiplication factor is calculated, and only the right-end matrix B is subjected to the elimination processing. The triangle of the matrix A is already 0, and the upper triangular elimination has no effect on the main diagonal elements.

It is worth noting that in order to reduce the read and write operations, in the first step of the upper triangle elimination, there is no need to read the last block into the memory, because the block has already been read into the memory in the next step of the lower triangle elimination group. The total number of read and write files required is

$$\begin{aligned} read\_num &= n! + n - 1 \\ write\_num &= n! \end{aligned} \tag{2}$$

where n is the number of blocks divided by the matrix.

The size of each block is smaller than the computer memory to ensure that the entire block can be read into the memory each time, but it should not be too small. Too small also causes memory waste. The standard for blocking is close to half of the physical memory of the computer, because there is room to store the maximum principal row number and multiplier required for the elimination.

3 Experimental results and analysis

The experimental environment in this paper is Intel 3.60 GHz CPU, 2G DDR memory, 160Gb hard disk. In order to verify the validity of the proposed method, the author first calculates the equations with a coefficient matrix size of 7000×7000. The theory requires memory 365Mb, which is smaller than computer memory (2Gb). The traditional in-core solution method and the extra-nuclear solution method proposed in this paper are calculated separately, and the solution time is compared. The experimental data pairs are shown in Table 1.

Table 1. Solution Time comparison.

Method	*Matrix block number*	*Time(s)*
Traditional intra-core solution	——	613
Out-of-Core solution method	2	648
Out-of-Core solution method	4	696
Out-of-Core solution method	6	736
Out-of-Core solution method	8	785

As can be seen from Table 1, the more the number of matrix blocks, the longer the solution time, because the number of blocks increases, the number of read and write data increases, and the required read and write time is longer. When the matrix is divided into 2 blocks, the solution time is the shortest, and the solution time in the kernel is only 35s (5.7%). The larger the coefficient matrix, the smaller the ratio of the data read and write time to the entire solution time, the higher the solution efficiency.

In order to verify the stability of the method, the coefficient matrix size is calculated to be 40000×40000 linear equations. The theory requires 11.9Gb of memory, which cannot be calculated on a single computer. Using the method of this paper, the matrix is divided into 12 blocks, which takes about 112 hours to successfully solve the equations. It shows that the proposed method has good stability for solving large-scale linear equations.

4 Summary

In the fields of computer-aided geometric modeling, computer-aided decision making, computer numerical calculation, etc., large-scale linear equations need to be solved. Solving such linear equations requires a lot of physical memory. The extra-nuclear solution method proposed in this paper solves the problem of insufficient memory. Experiments show that the proposed method can effectively solve large-scale linear equations, and the solution efficiency is only 5.7% different from that in the kernel.

References

1. Ramanujam, J., M. Kandemir, A. Choudhary, et al. Compilation techniques for out-of-core parallel computations. Parallel Computing, 1998, 23(324): 597-628.
2. Xiang T. M., Liang C. H. Itaerative solution for dense linear systens arising in computational electromagnetis. Journal of Xidian University, 2003, 30(6): 748-751.
3. Fu C. J. Parallel computing for solving finite element linear systems of equations on workstation cluster. Computer Engineering and Design, 2008, 29(24): 6441-6443.
4. Wu D, H. Parallel accelerating method of solving large-scale linear equations. Mechanical & Electrical Engineering Magazine, ,2008, 25(4): 58-59.
5. Zeng X. W. A new approach to parallel method for system of linear equations. Journal of UEST of China, 2005, 34(3): 413-416.
6. Chang S. L., C. L. Zhang, A new iterative algorithm for linear equation. Journal of Jinan University, 2004, 25(3): 256-259.
7. Fan Y. H., Q. Y. Lv, Y. F. Nie, Improved paralle algorithm for solving block-tridiagonal linear equations. Computer Engineering and Applications, 2009, 45(3): 60-63.
8. Ding Y., Z. H. Feng, T. J. He, Paralle computation of linear equations based on SSE. Traffic and Computer 2004, 22(1): 41-43.
9. Zhang Wei-ru and Pan Wu-ming. Parallel jacobi iterative algorithm based on MPI. Software Guide, 2008, 7(9): 16-17