Parallel Algorithms for Solving Large Sparse Linear Equations

Jingzhu Li[1,2]
[1.] School of Computer Science,
National University of Defense Technology
Changsha, China
[2]. Fok Ying Tung Graduate School
Hong Kong University of Science and Technology
Hong Kong

Peng Zou
School of Computer Science,
National University of Defense Technology
Changsha, China

Qingbo Wu
School of Computer Science,
National University of Defense Technology
Changsha, China

***Abstract*—To optimize the Block Widemann and Block Lancos algorithm is every important in solving large sparse systems in many engineering computing topic, so the parallel method of these two algorithms is built. This paper designs two different types of data parallel based on the original parallelism level and parallel scalability of two algorithms, and finally achieve a more efficient way for solving the problems. At last, we analysis the computing complexity and time cost, and gave a strategy for choosing the algorithms in different computing environment etc. based on the evaluation.**

Keywords- Large sparse linear equations; Block Lanczos; Block Wiedemann; Parallel Computing;

I. INTRODUCTION

Large sparse equations is an important engineering computing topic, has been widely used in many important areas of our modern life, such as computational fluid dynamics, simulation and design of materials, petroleum seismic data processing, numerical weather prediction, code-breaking, numerical simulation of nuclear explosions and so on. Those are highly related to the solving of the sparse linear algebra equations. Currently, the most effective way for solving large sparse equations is through Block Laczos and Block Widemann algorithm.

The key point of solving large sparse linear system efficiently when dealing with the issue above in this big data era is to design a reasonable parallel algorithm. As we already know that Block Laczos algorithm is a iterative algorithm, so the way make its computing process faster (as we hope to achieve efficient computing) is to find a deeper parallelism such as data parallel, and it is very magnitude because the large scale of data in most large sparse system becomes ignorable. While for the Block Wiedemann we hope to work on dividing matrix into blocks first and then work on a deeper parallelism. We believe in that way, we could achieve optimization for these two significant algorithms.

II. MODELING

A. Block Lanczos algorithm

When using the Block Lanczos algorithm tosolve large sparse linear equations, we set machine word width to N (bit), equations scale to n, then generate a random matrix y of n×N, whenV_0=A×y=B^T×B×y, the Core steps of the algorithm is to calculate the iterative formula:

$$V_{i+1} = AV_iS_i + V_iD_{i+1} + V_{i-1}E_{i+1} + V_{i-2}F_{i+1}$$

For i≥0：

$$D_{i+1} = I_N - W_i(V_i^T A^2 V_i S_i + V_i^T AV_i)$$

$$E_{i+1} = -W_{i-1}V_i^T AV_iS_i$$

$$F_{i+1} = -W_{i-2}(I_N V_{i-1}^T AV_{i-1}W_{i-1})(V_{i-1}^T A^2 V_{i-1}S_{i-1} + V_{i-1}^T AV_{i-1})S_i$$

For $i < 0$:

$$W_i = 0, V_i = 0, S_i = I_N .$$

In the iterative process, V_i is an n × N matrix, D_i, E_i, F_i, S_i and W_i is $N \times N$ matrices, I_N is an identity matrix. Iteration termination occurswhen $V_i^TAV_i$=0 (i≠0). If setting m to be the number of the iteration, namely the minimum number of $V_i^TAV_i$=0, then the value of m close to the rows of sparse matrix /63.24.

Then calculate the formula: $x = \sum_{i=0}^{m-1} V_i W_i V_i^T V_0$

When $V_m^T AV_m = 0$ and $V_m = 0$:

We get Ax=Ay, so the solution of Ax=0 will be the combination of columns of x-y.

For $V^T_mAV_m$=0 and Vm≠0:

Calculate z=(x-y)‖V_m (x-y is in the front columns of z and V_m is next n columns).

BZ: using Gaussian elimination to calculate the solution of BZU=0, finding basis of ZU, then obtaining partial solution of Bx = 0 [2].

It is easy to find the core calculation of this process is AV_i, that is B^T×B×V_i.

The core process of Block Lanczos [5][6] is processing the iteration of $B^T \times B \times y$, its parallel mode is a kind of data parallel modes which divide B into blocks, the blocks of B and B^T is shown as follows:

$$B_{(N\times N)} = \begin{pmatrix} B_{0(k\times N)} \\ B_{1(k\times N)} \\ \cdots \\ B_{(p-1)(k\times N)} \end{pmatrix}$$

$$B^T{}_{(N\times N)} = \begin{pmatrix} B^T{}_{0(k\times N)} \\ B^T{}_{1(k\times N)} \\ \cdots \\ B^T{}_{(p-1)(k\times N)} \end{pmatrix}$$

the vector y: $y_{(N\times n)} = \begin{pmatrix} y_{0(k\times n)} \\ y_{1(k\times n)} \\ \cdots \\ y_{(p-1)(k\times n)} \end{pmatrix}$

This model is a completely relevant data parallel model, its main process is: first, processing $U_{i(k\times N)} = B_{i(k\times N)} \times y_{(N\times n)}$ individually, namely each process runs in parallel. And then obtaining the complete vector U(N×n) in every process through full collection of communication model. Next, processing $V_{i(k\times n)} = B^T{}_{i(k\times N)} \times u_{(N\times n)}$ individually, and then obtaining every vector $V_{(N\times n)}$ through the communication model, at last, set y=v, back to the beginning of this process as the next iteration.

B. Block Wiedemann algorithm

Using the Block Wiedemann algorithm for solving large sparse linear equations includes several steps: generate a sequence of matrix, obtain minimal polynomial and construct equations [1] [3]. So we make a future analysis on each step.

Step 1: Matrix sequence generation

According to the principle of Block Wiedemann, the width for Y may be one or more machine words, so $a^{(i)} = X^T B^i Y$ can be split into a non-correlation calculation:

$$a_j^{(i)} = X^T B^i Y_j, (0 \le j < v)$$

Step 2: Minimal polynomial calculation

After splitting the matrix sequence, a coppersmith algorithm will be used to calculate matrix sequence to gain polynomial $f(x)$ [4]:

$$a(x) f(x) = g(x) + x^L e(x)$$

Based on the algorithm, it is easy to find out the main operations are polynomial multiplication and Gaussian elimination.

Step 3: the structure equations

$$\hat{W} = \sum_{k=0}^{d'} B^k Z \tilde{f}^{(k)}$$

Calculating the formula above, just similar to the first step, the core calculation is to divide the vector $B^K Z$ into several column blocks, then processing the parallel computing: $V_j = BZ_j$ [3].

The main purpose of the first and third step of Block Wiedemann algorithm is processing the iteration computation B × y, its parallel mode is a data parallel model which divides B into blocks. The length of n takes one or more machine words (ωbit), so that n = ω × ν, then each element of the vector Y is a machine word width of ν: $Y_{(N\times n)} = (Y_{0(N\times\square)} Y_{1(N\times\square)} Y_{2(N\times\square)} \cdots Y_{n(N\times\square)})$.

Dividing $a^{(i)} = X^T B^i Y$ into v non-relevant computing to achieve the first layer of parallel computing: $a_j^{(i)} = X^T B^i Y_j, (0 \le j < \square)$, as described in step 1.

The second layer of the parallel is dividing matrix B into several sub-matrices, i.e. row-based division:

$$B_{(N\times N)} = \begin{pmatrix} B_{0(k\times N)} \\ B_{1(k\times N)} \\ \cdots \\ B_{(p-1)(k\times N)} \end{pmatrix}$$

p is the number of the processors in use through the second layer of parallel, k=N/P;

Every iteration computing of BY_j could be split up and run in every processor individually as unrelated: $Y^{(i+1)}_{(j,1)(k\times\omega)} = B_{1(k\times N)} Y^{(i)}_{j(N\times\omega)}$, l=0, 1, 2 … p-1, the number of processors.

Every processor will access the result of others through communication in every iteration process:

$Y^{(i+1)}_{(j,1)(k\times\omega)}$, l=0, 1 … Myid-1, Myid+1 … p-1 (Myid is the current processor number). And the result is obtained as follow:

$$Y^{(i+1)}_{j(N\times\omega)} = \begin{pmatrix} Y^{(i+1)}_{(j,0)(k\times\omega)} \\ Y^{(i+1)}_{(j,1)(k\times\omega)} \\ \cdots \\ Y^{(i+1)}_{(j,p-1)(k\times\omega)} \end{pmatrix}$$

And then the system turns to the next iteration.

In a word, the first and third step of Block Wiedemann can be divided into several non-relevant parallel computing: $B\times Y_{0(N\times\square)}$ $B\times Y_{1(N\times\square)}$ … $B\times Y_{(v-1)(N\times\square)}$, these are the data parallel models which process full-relevant sparse matrix-vector multiplication. And the second step of Block Wiedemann uses Coppersmith algorithm to obtain matrix sequence for polynomial [4] f(x):

$a(x)f(x) = g(x) + x^L e(x)$

It is a polynomial-matrix multiplication, similar to the sparse matrix-vector multiplication, which divides the matrix in a parallel way. Hence, it is a full relevant data parallel model.

And then the process is finished. Next we should analysis the different parallel models for both algorithms individually and evaluate performances.

III. ANALYSIS AND PERFORMANCES

A. Algorithm complexity analysis

Lanczos algorithm complexity is shown in Table 1:

TABLE I. LANCZOS ALGORITHM COMPLEXITY

Items	iteration computing of B^TBY	
	B^TBY: L times	Communicate V: 2L times
Width of vector: n	$2dN^2/n$	$\frac{2LNn}{8(bit/B)} = 4N^2/8$

Wiedemann algorithm complexity (the first and third step only) is shown in Table 2:

TABLE II. WIEDEMANN ALGORITHM COMPLEXITY

Item	Step 1: AI sequence $a^{(i)}=X^TB^iY$		Step 3: made up the solution in minimal polynomial $\hat{w} = \sum_{k=0}^{d'} B^k Z\tilde{f}^{(k)}$	
	BV: L times	Communicate V: Ltimes	BV: L/2 $V_{N\times n}f_{n\times N}$: L/2	Communicate V: L/2
Width of vector: n	$2dN^2/n$	$\frac{LNn}{8(bit/B)} = 2N^2/8$	$(dN^2 + nN^2)/n$	$\frac{LNn/2}{8(bit/B)} = N^2/8$

So we can tell that the Block Lanczos algorithm complexity is 2/3 of which in Block Wiedemann. And when we analysis further more about the Block Wiedemann algorithm, we find that when vector $Y_{(N\times cn)}=(Y_{0(N\times n)}\ Y_{1(N\times n)}\ Y_{2(N\times n)}\ldots Y_{(c-1)(N\times n)})$ is divided into several blocks, the number of iterations will deduce to 1/c, besides each processing $B\times Y_{i(N\times \square)}$ is a non-relevant parallel computing. So it is not hard to claim that Block Wiedemann has more parallel scalability than the other one in the parallel computing field.

B. Computing time analysis

Normally, the iteration methods for sparse liner system need less storage and time. Since the computing time is a key point to evaluate a method, we analysis it for both algorithms [11].

Block Lanczaos computing time:

$$T_{lanzos}=(T_{bv}+T_{btv}+T_{vtav}) \times L \approx 2LT_{bv},$$

T_{vtav} is other matrix multiplication time, L=N/n, the scale of parallel is p.

Block Wiedemann computing time:

$$T_{wiedemann}=(T_{bv}+T_{av}) \times 3L+T_{copp} \approx 3LT_{bv+}T_{copp},$$

T_{av} is other matrix multiplication time, L=N/n, the scale of parallel is p, T_{copp} is the time cost in the second step of Block Wiedemann, namely Coppersmith algorithm ,obtaining polynomial from matrix sequence.

Based on the analysis of Wiedemann, we may induce the width of vector n into cn, and the sparse matrix-vector multiplication in the first and third step will be processed in the same c parallel computing. By then, L will deduce into L=N/(cn), and the time of Block Wiedemann is:

$$T_{wiedemann}=(T_{bv}+T_{av}) \times 3L/c+T_{copp} \approx (3/c)\ LT_{bv}+T_{copp},$$

The scale is cp.

In order to get acceleration in parallel for minimum time of the solution process, p is the scale of Block Lanczos algorithm while cp is of the Block Wiedemann. We can easily get the time difference between these two algorithms, which is:

$$T_d= T_{lanzos} - T_{wiedemann} \approx ((2c-3)/c)\ LT_{bv}-T_{copp}.$$

So when positive, Block Wiedemann will be a better choice for solving large sparse linear systems in parallel way, otherwise we prefer to another one.

For example, in common HPC, we assume that the number of CPU in computing environment is p=256, and the largest scale of parallel full-communication which is accelerating is P_{max}, set $p=P_{max}$, $c=[P/P_{max}]$ and we will make some examples:

If $P_{max}=64$,c=256/64=4: $Td=5/4LT_{bv}-T_{copp}$, because T_{copp} is far smaller than LT_{bv}, it is positive, Block Wiedemann is a better choice.

If $P_{max}=256$, c=256/256=1: $T_d=-LT_{b\square}-T_{copp}$, turn out to be negative this time, so it means Block Lancozs becomes the priority one.

IV. CONCLUSION

This paper focuses on the solution of large-scale sparse matrix, and builts a full-relevant data parallel model for both Block Lanczos and Block Wiedemann algorithm. To analysis and compares these two algorithms, we believe that Wiedemann algorithm is more scalable in parallel computing. But according to different data scale and computing environment, different algorithm choice strategy will be available, especially in solving practical engineering issues.

ACKNOWLEDGMENT

Our work is completed in Fok Ying Tung Graduate School, Hong Kong University of Science and Technology, supported by project "Research and Develop of Community Version of New Network Terminal Operating System'. (National High-tech R&D Program of China, No.2011BAHI4B02)

REFERENCES

[1] Erich Kaltofen, B. David Saunders, On Wiedemann's Method of Solving Sparse Linear Systems. National Science Foundation No.CCR·90-06077 and No.CDA-88-05910.

[2] Wontae Hwang, Dongseung Kim. Load Balanced Block Lanczos Algorithm over GF(2) for Factorization of Large Keys. Department of Electrical Engineering, Korea University, Seoul, Korea (Rep. of).

[3] G.Villard. Further Analysis of Coppersmith's Block Wiedemann Algorithm for the Solution of Sparse Linear Systems.

[4] Villard.G. Computing Minimum Generating Matrix Polynomial. 1997. Preprint IMAG Grenoble, Franc.

[5] Hwang, W. Improved Parallel Block Lanczos Algorithm over GF(2) by Load Balancing, Master Thesis, Korea University, Dec. 2005.

[6] A. El Guennouni, K. Jbilou, H. Sadok. The Block Lanczos Method for Linear Systems with Multiple Right-hand Sides. University du Litteral.

[7] Wise D. S, Franco. J. Costs of quadtree representation of non-dense matrices. Parallel distribute Compute. 9, pp. 282-296(1990).

[8] Lanczos, C.: An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators. Journal of Research of the National Bureau of Standards 45, 4 (Oct. 1950), pp. 255-282.

[9] Flesch, I., Bisseling, R. H.: A New Parallel Approach to the Block Lanczos algorithm for Finding Nullspaces over GF(2). Master's thesis, Department of Mathematics, Utrecht University, Utrecht, the Netherlands, November 2002.

[10] B. Parleft, A New Look at the Lanczos Algorithm for Solving Symmetric System of Linear Equations. Linear Algebra Appl, 29(1980),pp.323-346

[11] Yan Zhong. " Parallel menthod and preconditioning studies for large sparse liner system" PHD thesis, school of computer science,NUDT.