International Journal of Grid and Distributed Computing
Vol. 10, No. 1 (2017), pp.31-42
http://dx.doi.org/10.14257/ijgdc.2017.10.1.04

Parallel Gauss-Jordan Elimination on Two-Dimensional Constant Bandwidth Storage

Yu Liu[1] and Renhao Xiong[2]

[1]*College of Mechanical and Control Engineering of Guilin University of Technology, Guilin, Guangxi, China*
lewis_5709@163.com
[2]*College of Information Science and Engineering of Guilin University of Technology, Guilin, Guangxi, China*
xiongrenhao@126.com

Abstract

As commonly used in two-dimensional magnetotelluric (MT) inversion, Occam inversion method, though featured by its stable convergence and independence on initial model, is relatively poor in time performance, with Gauss-Jordan elimination as one of the biggest time-consuming parts. For symmetrical banded coefficient matrix on two-dimensional constant bandwidth storage, the application of sequential dispatch strategy to the workload on lines of work triangle in parallel algorithm will lead to load imbalance. Therefore, based on paired-dispatch, a kind of strategy is presented to solve this problem, after which the algorithm's effect on shared memory parallel system is verified and research is carried on to focus on the optimization methods of algorithm performance. Experimental results as compared with serial algorithm show that these methods have effectively improved the performance of algorithm, which contributes to a speedup of 3.72 of parallel Gauss-Jordan algorithm as a whole, and that MT Occam inversion algorithm based on this parallel algorithm also demonstrates good speedup performance.

Keywords*: Gauss-Jordan elimination; two-dimensional constant bandwidth storage; high performance computing; algorithm optimization; shared memory parallel programming*

1. Introduction

Generally, the solution methods of linear algebraic equations can be divided into direct methods and iterative methods [1]. Parallel algorithms based on these two kinds of methods are widely researched, but most of them are iterative methods and based on distributed memory system. The reason, on the one hand, though direct methods based on Gauss elimination are simple, they always have high time complexity and poor time performance. On the other hand, iterative methods have better application adaptability for they have the advantage of balancing accuracy and time to some range. Meanwhile, distributed memory systems are apt at making use of multi-host environment for wider performance improvement space. Visibly, Gauss elimination has limited application scope.

Magnetotelluric Occam inversion adopts especial adaptive algorithms to calculate Lagrange multiplier, whose method is stable and doesn't need more experience, but relatively slow [2,6]. Gauss-Jordan elimination is exactly one of the most time-consuming steps. In Occam inversion algorithm, rigidity matrix commonly uses two-dimensional constant bandwidth storage for its zonal distribution. Two-dimensional constant bandwidth storage saves memory space while it adds difficulties to algorithm design on shared memory parallel systems. But the efficiency of elimination increases and the procedure of

ISSN: 2005-4262 IJGDC

elimination maintains its natural parallelism for the specificity of matrix. After all, the parallelization of Gauss-Jordan elimination can be still regarded as a workable scheme to improve time performance.

It's not difficult to prove Gauss-Jordan elimination's parallelizability theoretically. But only a theoretical argument is not enough for such applications as Occam inversion method. Specific implementation detail increases the difficulty and complexity of parallel coding in the development process of application programs. Fortunately, such programming models as OpenMP (Open Multi-Processing) have greatly improved the development efficiency of parallel programs. Comparison results of the algorithm's OpenMP realization to sequential dispatch strategy show that paired-dispatch strategy has efficiently solved the problem of load imbalance. Meanwhile, among factors that affect program's time performance, influence of the performance of memory access can't be overlooked. The bigger the problem's scale is, the more important it is to optimize the performance of memory access, as well as other optimization such as improvement of the algorithm itself which contributes to better time performance to a certain degree, too. As experimental results indicated, these endeavors considerably speed up the program.

2. Gauss-Jordan Elimination on Symmetric Banded Matrix

Define n order non-singular symmetric matrix A and n order column vector B, then equations can be expressed as AX=B, in which X stands for the unknown vector. Process of Gauss-Jordan elimination is exactly the conversion process of A and B[7, 11]. One time of elimination process selects the ith line as the primary line, then executes elimination from line i+1 up to line n. So the elimination process needs n-1 times totally. Let L^i be the primary line in the ith elimination, A^i be the matrix after the ith elimination, C^i be the n-i order matrix to be eliminated after the ith elimination, it has:

$$C^i_{r,s} = A^i_{r+i,s+i} \qquad \text{(i)}$$

in which $0 \le i \le n$-1, $1 \le r,s \le n$-i and $C^0=A^0=A$. There are also another two theorems for the elimination process of symmetric matrix A.

Theorem 1: In the process of elimination, $C^i(0 \le i \le n\text{-}1)$ keeps its symmetry.

Prove: $C^0=A$ is symmetric, suppose $C^k(0 \le k \le n\text{-}1)$ is symmetric, perform an elimination to C^k. That is, to all $2 \le r,s \le n$-i, let:

$$C^{k\,\prime}_{r,s} = C^k_{r,s} - \frac{C^k_{r,1}}{C^k_{1,1}} C^k_{1,s} \qquad \text{(ii)}$$

and:

$$C^{k\,\prime}_{s,r} = C^k_{s,r} - \frac{C^k_{s,1}}{C^k_{1,1}} C^k_{1,r} \qquad \text{(iii)}$$

while:

$$C^k_{r,s} = C^k_{s,r} \qquad \text{(iv)}$$

That's to say, C^{k+1} is symmetric, and theorem 1 is proved.

Theorem 2: $A^i_{i,i}=1$, $A^i_{i,i+s}=C^{i-1}_{s+1,1}/C^{i-1}_{1,1}$ $(1 \le i \le n\text{-}1,\ 1 \le s \le n\text{-}i)$.

Namely, after the ith elimination, the ith element of L^i is 1, and the subsequent elements are respectively equal to proportionality coefficient in the process of the ith elimination. Theorem 2 is easily proved because of the symmetry of the submatrix to be eliminated according to theorem 1. These two theorems demonstrated above are the foundations of two-dimensional serial and parallel bandwidth storage Gauss-Jordan elimination method.

2.1 Algorithm Description

Gauss-Jordan elimination is generally divided into two steps, namely, the elimination and the back substitution, which can be specifically described as follows:

Step 1: Select the primary line L^i, eliminate the elements of the non-primary lines successively, transform A and B's corresponding elements;

Step2: After finishing the elimination of the non-primary line, let the (i+1)th to nth of L^i and the corresponding elements of B are divided by the ith element respectively, the ith element of the L^i is set to 1;

Step 3: Execute the above operations n-1 times totally to accomplish the elimination process, and at the same time, the matrix of A is transformed to the upper triangular matrix whose diagonal elements are 1;

Step 4: From the n-1 line of the A, eliminate the non-zero elements on the right of the diagonal line, and meanwhile transform the corresponding elements of the B;

Step 5: Perform Step 4 n-1 times, let A transform to the cell matrix.

The steps mentioned above are general ones of Gauss-Jordan elimination method, the actual algorithm can be optimized according to theorem 1 and theorem 2. According to theorem 2, since the proportional coefficient has been calculated on the process of elimination, elements behind the ith element of L^i are set to the corresponding proportional coefficient directly.

According to theorem 1, the submatrix to be eliminated in the process of elimination maintains symmetric, and only the elements on the right of diagonal line need to be eliminated, which reduces the amount of computation greatly. In addition, there are also some other feasible optimization methods. For example, for Step 2, because the A is finally transformed to a unit matrix, the operation of set 1 can be omitted if the elements on the diagonal line are set to 1. At the same time, in Step 3, "the diagonal elements are 1" is not really the 1. In fact, the diagonal elements do not need to be accessed truly, this agreement is feasible. Also, for Step 4, the solution of the equation can be obtained after this step is completed, since it is the transformed B, the matrix transformation in A is not necessary.

In some implementations of Gauss-Jordan elimination method, Step 1 is decomposed into two steps: transform A first, and transform B after the first step is finished. As theorem 2 shows, these decomposed steps are feasible. In fact, the decomposition algorithm reduces the time performance in some extent. The analysis is carried out below.

2.2 Two-dimensional Constant Bandwidth Storage

Suppose the bandwidth is b, the two-dimensional constant bandwidth storage matrix S of band symmetric matrix A is shown in Figure 1. As theorem 1 shows, the Gauss-Jordan elimination on A can be performed on S, the elimination algorithm for A needs only to correspond to S. The relationship between A and S can be expressed as follows:

$$A_{i,j} = S_{i,j-i+1} \qquad \text{(v)}$$

Thereinto $1 \le i,j \le n$, $i \le j \le i+b-1$. Since the operation on S is equivalently on A, for the sake of simplicity, the elimination method on S is described as the original matrix A in the following passage. Because the elements out of the primary line bandwidth are zero, the elimination on these columns can be omitted. Therefore, in an elimination process, the elements to be eliminated gather in a triangle region, as shown in Figure 1, which is called "work triangle". From the above analyses, the two-dimensional constant bandwidth storage method not only saves the storage space, but also enhances the efficiency.

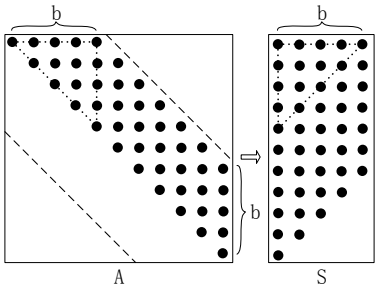

Figure 1. Two-dimensional Constant Bandwidth Storage of Matrix S

3. Parallel Algorithm

In the two steps of Gauss-Jordan elimination, elimination is the most time consuming. Take 50 thousand order, 1 thousand bandwidth matrix A for example, elimination occupies more than 99% of the running time as shown in Table 1 which records the statistical data of these two steps. In elimination process, the current elimination depends on the previous elimination result and the order of the submatrix to be eliminated minished step by step. Since the elimination operation of each element in the lines to be eliminated can be done parallelly, generally, the elimination step can also be paralleled.

Table 1. Running Time of Serial Algorithm(s)

Elimination	Back Substitution	Total	Elimination/ Total
162.2854	0.403	162.688	99.7%

The banded distribution of non-zero elements in A gathers the elements to be eliminated inside the work triangle. Each time's calculation amount decreases as the order of submatrix becomes smaller in elimination of normal matrix. Different to normal matrix, the width of work triangle's right-angle side doesn't diminish until it moves down to the tail. That's to say, in the last *b*-1 times' eliminations, the width of the work triangle reduces successively by 1 from *b*, but remains the same in the previous *n*-*b*+1 times'' elimination. Each time's elimination workload is balanced though total parallel workload is smaller than the dense matrix at the same order, providing favorable conditions for the effect of parallel computing in a general way where *b* is much less than *n*. Even so, sequential dispatch strategy goes against the load balance and causes a waste of computing resource because element amount of each line in work triangle is not the same to each other. The following paired strategy can be adopted to solve this problem.

3.1 Paired Dispatch Strategy

Inside work triangle whose width is *b*, there are *b*-1 lines to be eliminated, and element number in lines to be eliminated decreases progressively. It will cause a waste of computing resources if lines to be eliminated are dispatched sequentially to processors, for the relatively huge difference of calculation amount between each processor leads to processor idleness when it finishes calculating before other processors. Visibly, partition among lines to be eliminated is not Uniform Partition[12, 14]. Workload must be decomposed in lighter granularity so as to achieve uniform partition, to balance workload and to enhance the processor utilization rate. Some kind of paired dispatch strategy can be applied to realize that. Lines to be eliminated are from line 2 to line *b* for work triangle of width *b*. Lines are dispatched to processors in pairs because element number decreases progressively in lines. Let the total amount of processors be *p'*:

$$\begin{cases} v = \lceil (b-1)/2 \rceil \\ c' = \max(1, v/\ p') \\ p = \min(p', v) \end{cases} \quad \text{(vi)}$$

Thereinto, v is the total pair amount of lines to be eliminated. Each processor is tied to c' pairs of lines, while the amount of actually used processors is p. In addition, let pair number actually dispatched to a processor be c, for the ith processor $P_i(0 \le i \le p\text{-}1)$:

$$c = \begin{cases} c'(i \ne p-1) \\ v - i \times c'(i = p-1) \end{cases} \quad \text{(vii)}$$

All lines dispatched to P_i can be expresses as set $L=L_1UL_2$, in which:

$$\begin{cases} L_1 = \{l \mid 2 + i \times c' \le l \le 1 + i \times c' + c\} \\ L_2 = \{l' \mid l' = 2 + b - l, l \in L_1 \wedge l \ne 2 + b - l\} \end{cases} \quad \text{(viii)}$$

For processor P_i, paired strategy implies branch statement, namely, respectively, calculation of variable c and dispatch set L_2 of lines to be eliminated. For the former, only processor P_{p-1} executes different branches; for the later, when l=2+b-l, it has l=(2+b)/2, line l can only be dispatched to P_{p-1} as well. So that processor P_0 to P_{p-2} execute the same branch, and no special regulations are needed for branch optimization for paired strategy.

3.2 Parallel Algorithm

In shared memory systems, the main parallelization means of the algorithm is that the lines which can be eliminated asynchronously in each elimination are dispatched to processors by paired strategy, and race condition brought in by data sharing is removed. An example, to which this strategy applies in multiple threads is shown in Figure 2, in which work triangle's width is 12, lines to be eliminated are from line 2 to 12, and the total thread number is 3. Visibly, except for the last one, the workload of other threads is the same. Experimental results show that, compared to sequential strategy, paired strategy saves 42% of the running time.

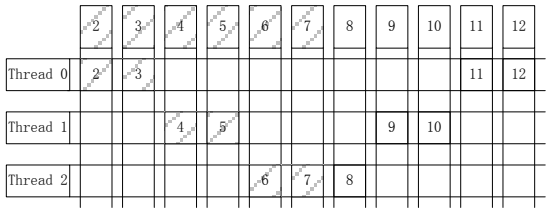

Figure 2. Paired Strategy

According to theorem 2, serial algorithm can be optimized by directly setting the proportionality coefficient on primary lines after the elimination of each line. But the data of primary lines in parallel elimination progress are shared by every processor which causes conflict condition. Generally two kinds of methods can be adopted to relieve conflict condition: 1) Cache primary line. Writing in to primary line won't cause conflict because each line's proportionality coefficient is located at primary line's different position; 2) Write delay. Every processor writes a proportionality coefficient to the shared cache array, which is copied to the primary line after this time's elimination completes.

Paired strategy is mainly applied to elimination step. For back substitution step, one elimination can be sketched as continuous reductions of some element on B, in which conflict condition is write operation to that element. The rescission method is to reduce continuous reductions to continuous additions and one time of reduction to that element, ensures mutually exclusive writing. Time performance improvement of parallel algorithm

is mainly reflected in elimination step, for the back substitution step is less time-consuming.

4. Performance Optimization

Since the relatively high time complexity of Gauss-Jordan elimination method, store and access methods of it in particular, have great influence on the performance of the algorithm when the order of A is relatively large.

4.1 Storage Optimization

The effect caused by storage mode to algorithm time performance specifically manifests as the latency waiting for the completion of memory access. To balance the cache latency and hit rate, modern CPU usually has a multi-level Cache structure to reduce the average memory access time[15, 19]. Before the CPU access memory, multi-level Cache is sequentially inquired until it hits, or memory is accessed when it misses. In a multi-level Cache structure, the capacity and speed are different, LLC (Last Level Cache) is the highest level, usually with the largest capacity but the slowest speed. If the LLC is queried when Cache hits, it means Caches of all levels are queried. If Cache misses, the actual time of memory access is the time of query all levels' Cache and direct memory access. Visibly, the number of LLC misses (LLC-M) and the number of LLC references (LLC-R) are two of the important indicators to measure the average memory performance[20, 21].

The data transfer units between Cache and memory are Cache line whose size are fixed. When the data are missing, a block which includes required data is copied from memory to Cache, so the data on the adjacent address of the memory should be assessed as continuously as possible to increase the hit rate of the Cache system. The higher the hit rate, the better the average access performance is.

Two-dimensional or multi-dimensional arrays are deployed and stored at consecutive address space in memory. For example, in C language, two-dimensional array is stored by row major, and in the Fortran language, two-dimensional array is stored by column major. Different access orders have great influence on the performance. For example, when accessing to a matrix of *n* rows and *n* columns in Fortran, Figure 3 shows respectively the time spent when it is accessed by row major order and column major order. When *n* is equal to 2000, both are very close; but when *n* is equal to 10,000, the time required by row major order is 2 or more time by column major order. The function of Cache system is obvious--the bigger the problem, the more prominent the effect.

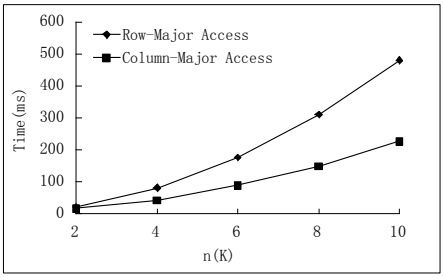

Figure 3. The Time of Row-major and Column-major Access

In the experiment, the two indicators LLC-M and LLC-R reflect the impact, as shown in Table 2. The "-" indicates the failure to statistical data (counter value is less than 6000). When data accessed by row major order and *n* increase from 2000 to 10000, the citations of LLC increases by 135 times. While by column major order, the values of LLC-M and LLC-R are always under 6000. The elimination method in this algorithm description assess

data by row major order while Fortran's default storage mode by column major order. Therefore, for the realization of the algorithm in Fortran, a possible optimization is adopted to transpose S, so that the sequential access of the elements to be eliminated in rows are converted to sequential access in columns.

Table 2. Statistical Data for LLC-R and LLC-M by Row-major and Column-major Access

n	Row-Major (K)		Column-Major(K)	
	LLC-M	LLC-R	LLC-M	LLC-R
2000	340	340	-(<6)	-(<6)
4000	340	8162	-(<6)	-(<6)
6000	680	15305	-(<6)	-(<6)
8000	1020	26188	-(<6)	-(<6)
10000	1020	46254	-(<6)	-(<6)

4.2 Memory Access Merging

In addition to the storage optimization, the reduction of the times of memory access or memory access merging is also an effective optimization tool. For example, for readability and maintainability considerations, elimination steps of the algorithm are usually divided into two separate parts: the elimination on A and the transformation on B. From Theorem 2, it's known that this division mode is feasible.

Generally the elimination refers to the elimination of a line in the augmented matrix, and elements needed to be processed include both elements in coefficient matrix A and the corresponding ones in B. From Theorem 2, elimination on A stores the corresponding scale factor, that is, after erasing all the lines of the work triangle, elements except for the first one in the primary row are equal to the scale factor when the other lines of the work triangle is eliminated. Therefore, the elimination of the augmented matrix can be divided into two steps: firstly, eliminate A, and do not transform the elements in B; secondly, after the elimination of A, transform B separately according to the proportion coefficient saved in A. When transforming B, except for the operation steps corresponding to elimination in A, setting the diagonal elements to 1 and other operations are also included. In fact, the diagonal elements can be agreed to 1 without being divided, but the division of B elements cannot be omitted. Obviously, the algorithm of decomposition steps is easy to be written, but the accessing time of elements on A is increased: in the elimination of augmented matrix, the calculated ratio coefficient r is saved as the main row's elements, and then use it to transform B, while r is normally kept in registers; in the decomposition steps, B's transform needs to take out r from A first.

In decomposition steps, not only taking out the proportional coefficient but also accessing to the primary line of the first element would increase time because both the calculation of ratio coefficient and setting diagonal to 1 are required to access this element. In this regard, memory access merging is feasible, by immediately transforming B when eliminating A. In fact, the effect of memory access merging is far less outstanding than storage optimization due to the structure of the Cache which makes the accessing of A continuous in both cases. In both of the contrast experiments, the recorded data of LLC-M and LLC-R confirm the role of the Cache system. In short, the experimental data show that the above two kinds of optimization have greatly improved the performance of the algorithm.

5. Results and Analysis of the Experiments

Optimization methods and paired strategies of the algorithm are tested as follows, taking a 50 thousand order matrix, whose bandwidth ranges from 100 to 1000, for example. The experimental host is equipped with quad-core processor, the programming language is Fortran, and the parallel routine uses OpenMP compiler directives. The compiler uses PGI CDK (The Portland Group Cluster Development Kit)[22, 24], PGPROF (Portland Group Profiler)[25] is one of the products of PGI CDK, which provides a powerful tool for analyzing and testing program performance.

5.1 Storage Optimization and Memory Access Merging

Figure 4 compares the time performance of no optimization, storage optimization, memory access merging and all optimization (storage optimization and memory access merging). When bandwidth is relatively narrow, the optimization almost doesn't work, with the increase of bandwidth, the effect of optimization become obvious. When the bandwidth is 1000, the optimization methods save 22% of the running time. The influence of storage optimization is larger compared to memory access merging. Storage optimization and memory access merging, compared to no optimization respectively, storage optimization contributes more to the performance of improvement. However, memory access merging has little influence while proceeding memory access merging after storage optimization (that is, all optimization). The reason is that whether memory access merging is performed or not after storage optimization, the access of coefficient matrix A is sequential. Memory access merging does contribute to the time performance by reducing its accessing times; however, the existence of Cache system leads to the insignificance of this optimization.

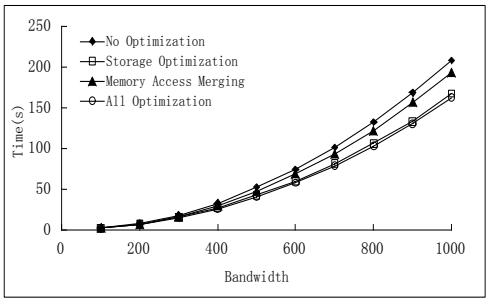

Figure 4. Time Performance before and after the Optimizations

In the realization, the elements of A are plural, which takes up 8 Bytes memory. When bandwidth increases from 100 to 1000, the work triangle increases from 39 KB to 3910 KB. The average accessing performance is closely linked with caching system structure, capacity and replacement strategy, etc. Table 3 calculates two indexes (LLC-M and LLC-R) on the matrix S whose bandwidth ranges from 500 to 1000.

Table 3. LLC-M and LLC-R statistics

Bandwidth	No Optimization(K)		Whole Optimization(K)	
	LLC-M	LLC-R	LLC-M	LLC-R
500	3,061	768,626	0	91,827
600	3,401	1,113,151	0	121,076
700	5,782	1,525,011	0	150,325
800	6,122	2,003,192	0	174,471
900	10,203	2,604,142	0	210,862
1000	28,568	3,286,731	340	247,593

As shown in the table, the two indexes fell dramatically after optimization and miss-query ratio of LLC (LLC-M/LLC-R) decrease significantly. For example, when the bandwidth is 1000, compared non-optimized algorithm with optimized one, the former's LLC-M is 84 times of the latter's and the former's LLC-R is 13 times of the latter's, miss-query ratio of the former is 8.7 ‰ while the latter is 1.4 ‰, which shows that storage optimization and memory access merging enhance the efficiency of caching system greatly. The following serial and parallel algorithms and their contrast are based on these optimizations.

5.2 Load Balancing and Accelerating Performance

Figure 5 shows the comparison among the serial algorithm, the parallel algorithm of sequential strategy and the parallel algorithm of paired strategy in time performance. In the sequential strategy, one elimination will distribute the *b*-1 line of the work triangle whose bandwidth is *b* to *p* processors, every processor's workload is different, the processor with less workload ends first, so one elimination's time *ts* is equal to the running time of the processor with largest workload:

$$ts = \Theta(\sum_{i=1}^{(b-1)/p}(b-i)) \qquad \text{(ix)}$$

In paired strategy, each processor distributes equal workload,and one elimination''s time *tp* is:

$$tp = \Theta(b\frac{b-1}{2p}) \qquad \text{(x)}$$

So that the time performance of paired strategy would be *ts*/*tp* times of sequential strategy when processor number equals to *p*:

$$ts / tp = (2b - \frac{b-1}{p} - 1) / b \qquad \text{(xi)}$$

The value of *ts*/*tp* is 1.75 while *b*=1000, *p*=4. Meanwhile, in the experiment *ts*/*tp*=75.383/43.791=1.72, which reaches good agreement with the theoretical value 1.75. The proportion's limit is 2-1/*p* when $b \to \infty$.

As for acceleration performance, with the increase of bandwidth, the proportion of parallel component increases and the acceleration ratio of parallel algorithm also increases gradually. According to the Amdahl's Law, when bandwidth is 1000, the proportion of parallel component is 99.8%, and the biggest speed ratio is 3.99 for 4 processors; while in the experiment the speed-up ratio is 3.72, it is close to the theoretical limit.

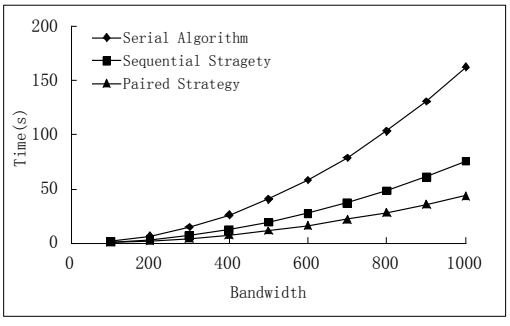

Figure 5. The Comparison of Sequential and Paired Strategy

5.3 The Calculating Example of Occam Inversion

Gauss-Jordan elimination is one of the most time-consuming steps in the MT Occam inversion algorithm. Taking an 889 parameters model as an example, table 4 lists the proportion of running time of relevant steps. Gauss Jordan elimination method accounts for 67% of the total time, which occupies the largest proportion. When applying parallel Gauss Jordan elimination which is based on paired strategy to this algorithm, the acceleration ratio of 4 threads can reach 2.01. According to the Amdahl's Law, when the number of threads is 4 and the parallel component is 67%, the biggest improvable acceleration ratio is about 2.01, which is almost equal to the above value.

Table 4. MT Occam Example of 889 Parameters

Gauss	Gaussd	Cholin	Atamul	Parfld	Parjac	Others
67%	12%	8%	6%	5%	1%	1%

6. Conclusion

Gauss-Jordan elimination based on two-dimensional constant bandwidth storage is paralleled applying paired strategy in this article, and optimization methods of the algorithm based on shared memory systems are discussed. Results and analysis of the experiments show that optimization methods like storage optimization and memory access merging have effectively saved the algorithm's running time, whose effectiveness is highlighted especially for high-order matrix. Paired strategy, whose speed-up ratio is close to the biggest improvable performance according to the Amdahl's Law, further improves the algorithm's time performance as compared to sequential strategy whose running time is as 2-1/*p* times as the former.

Acknowledgements

This research was supported by Natural Science Foundation of China 41264005 and Natural Science Foundation of China 41374079.

References

[1] G.-M. Wu, "Parallel Algorithms and Architectures for Matrix Computations on FPGA", National University of Defense Technology, **(2011)**.
[2] L. Yu, "Parallel MT Occam inversion scheme and its performance analysis", Journal of Wuhan University of Technology, vol. 29, **(2007)**, pp. 136–137.
[3] Y.-L. Meng, L. Yu and J.-Y. Wang, "Parallel computing research of magnetotelluric Occam inversion based on personal computer cluster", Geophysical Prospecting for Petroleum, vol. 45, **(2006)**, pp. 311–315.
[4] Y.-C. Pak, T.-L. Li and Y.-L. Liu, "Improvement of choosing Lagrange Multiplier on MT2D Occam inversion", Journal of Jilin University (Earth Science Edition), vol. 44, **(2014)**, pp. 660–664.
[5] A.-H. Wong, "Occam inversion and its applications of transient electromagnetic depth sounding", Geophysical Prospecting, vol. 42, **(2007)**, pp. 74–76.
[6] H. Chuan, Z. Zhong, Y.-Y. Tan and H.-L. Zhang, "Velocity Model Inversion for Microseismic Monitoring Based on Occam Inversion Algorithm", Acta Scientiarum Naturalium Universitatis Pekinensis, vol. 51, **(2015)**, pp. 43–48.
[7] M. D. Petković and P. S. Stanimirović, "Gauss-Jordan elimination method for computing outer inverses", Applied Mathematics & Computation, vol. 219, **(2013)**, pp. 4667–4679.
[8] J. Ji and X. Chen, "A new method for computing Moore–Penrose inverse through Gauss–Jordan elimination", Applied Mathematics & Computation, vol. 245, **(2014)**, pp. 271–278.
[9] Y. Mei and Z.-M. Li, "Parallel algorithm for solving large-scale dense linear system on CUDA", Computer Engineering & Applicationsl, vol. 47, **(2011)**, pp. 27–30.
[10] L. Yu and X. Yi, "GPU acceleration for the gaussian elimination in magnetotelluric Occam inversion algorithm", Springer International Publishing, **(2015)**, pp. 123–131**.**
[11] G. Sharma, A. Agarwala and B. Bhattacharya, "A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA", Computers & Structures, vol. 128, **(2013)**, pp. 31–37.

[12] G.-L. Chen, "Parallel computing-structure, algorithm and programming", Higher Education Press, Beijing, **(2003)**.
[13] Q.-N. Deng, "An introduction to parallel programming", China Machine Press, **(2012)**.
[14] M.-J. Xue and Y.-J. Liu, "Program of blocks combining with LDLT method for finite element analysis", Computer Science, vol. 41, **(2014)**, pp. 408–409.
[15] J. Kalamatianos, E. J. Mclellan, P. Keltcher, S. Manne, R. E. Klass and J. M. O'connor, "Management of cache size", **(2015)**.
[16] Kanoh, "Cache memory and control method thereof with cache hit rate", **(2015)**.
[17] L. Novakovsky, A. Gendler and O. Stauber, "Multi latency configurable cache", **(2015)**.
[18] L. M. Pinho, E. Quinones, M. Bertogna, A. Marongiu, J. P. Carlos, C. Scordino and M. Ramponi, "P-SOCRATES: A parallel software framework for time-critical many-core systems", Euromicro Conference on Digital System Design, **(2015)**, pp. 214–221.
[19] C. A. Barros, L. F. Q. Silveira, C. A. Valderrama and S. Xavier-De-Souza, "Optimal processor dynamic-energy reduction for parallel workloads on heterogeneous multi-core architectures", Microprocess, Microsyst, vol. 39, **(2015)**, pp. 418–425.
[20] H. Tao, "Research on Key Techniques for Optimizing Last Level Cache Performance", Peking University, **(2013)**.
[21] X.-M. Jia, "Access Behavior Analysis and Optimization of Caches for Chip Multi-Processors", National University of Defense Technology, **(2011)**.
[22] The Portland Group, PGI Release Notes, http://www.pgroup.com/doc/pgirn.pdf, **(2015)**.
[23] The Portland Group, "PGI Visual Fortran Release Notes", http://www.pgroup.com/doc/pvfrn.pdf, **(2015)**.
[24] The Portland Group, "PGI compiler user's guide", http://www.pgroup.com/doc/pgiug.pdf, **(2015)**.
[25] The Portland Group, "PGPROF profiler guide: parallel profiling for scientists and engineers", http://www.pgroup.com/doc/pgprofug.pdf, **(2015)**.

Authors

Yu Liu, He received the M.S. degree in petroleum geophysics from Wuhan College of Geology, Wuhan, China, in 1982, and the Ph.D. degree in geodetection and information technology from China University of Geosciences, Wuhan, China, in 2006. He is currently a Professor with the College of Mechanical and Control Engineering, Guilin University of Technology, Guilin, China. His current research interests include physical geography, image processing, and parallel computing.

Renhao Xiong, He is currently a postgraduate student of Guilin University, Guilin, China, pursuing a Master's degree in computer science and technology. His current research interests include image processing and parallel computing.