International Journal of Grid and Distributed Computing
Vol. 9, No. 9 (2016), pp.67-82
http://dx.doi.org/10.14257/ijgdc.2016.9.9.07

A MPI + OpenMP + CUDA Hybrid Parallel Scheme for MT Occam Inversion

Yu Liu[1], Renhao Xiong[2] and Renhao Xiong[2]

[1,2,3]*College of Information Science and Engineering of Guilin University of Technology*

Abstract

To improve the performance of the Magnetotelluric Occam inversion, by in-depth analysis of the sequential algorithm, we develop a multi-level hybrid parallel computing scheme for MT Occam inversion based on MPI+OpenMP+CUDA and implement it on a small heterogeneous cluster. We implement the parallel algorithm for solving linear equations with Gauss elimination, jacobian matrix, cross-product matrix calculations and Cholesky decomposition. Through reasonable decomposition, combination and mapping of computing tasks, the scheme reduces the data traffic and realizes the purpose of load balancing. By changing the matrix storage order，the memory access speed is significantly increased. The scheme is tested with multiple synthesis data from 2-D theoretical models and the execution efficiency of sequential code and parallel code on a 4 nodes PC cluster is comparatively analyzed. The test results show that the realization of this hybrid parallel algorithm is feasible and efficient. Compared with the sequential code and pure message passing algorithm, the inversion speed is obviously increased.

***Keywords**: Magnetotelluric, inversion, heterogeneous, parallel computing, MPI+OpenMP+CUDA*

1. Introduction

Occam inversion is one of the most important geophysical inversion methods, which pursues the best fitting between the model and original data, demands that the model is the smoothest at the same time [1]. Using adaptive algorithms to calculate the Lagrange multiplier (μ^{-1}), Occam inversion has the strongpoint of stability of convergence and is independent of starting model. So, this method has been widely used in geophysical exploration [2-5].

But the low computing speed has been the main problem of the Occam inversion method. Hu Zuzhi *et al.*, have done some comparative experiments for various inversion methods, their results show that the Occam method is the most stable, but its inversion time is the longest [6]. The major causes are: 1) the jacobian matrix of the explicit calculation consuming too much CPU time; 2) too dense finite element grid subdivision will lead to huge coefficient matrix, make the amount of calculation increased rapidly; 3) using adaptive algorithm to calculate the Lagrange multiplier will lead to the increase of the iteration, making the overall increase of the calculation. After the algorithm been put forward, many geophysical workers home and abroad have targeted made some improvements to it, obtained certain achievements on its time performance [7-10].

Parallel computing is one of the ways to improve the computing speed of geophysical processing and some research works has been done in the aspect of electromagnetic data parallel processing. Newman and Alumbaugh (1997) used parallel computing to calculate 3-D electromagnetic imaging [11]; Zyserman and Santos (2000) implemented the three-dimensional magnetotelluric finite element parallel computing [12];Chen Jin-chuang and Dai Guang-ming realized a 2.5-D CSAMT forward modeling parallel computing on the PVM environment [13]; Tan Han-dong *et al.*, (2005) realized parallel computing of magnetotelluric 3-D forward modeling[14], and then the magnetotelluric 3-D RRI inversion [20]; Liu Yu (2006) implemented the two-dimensional magnetotelluric Occam inversion based on PVM parallel flatform [15, 16]; In the same year, Chen Lu-jun *et al.*, Realized 3-D electromagnetic numerical simulation based on MPI[21]; Hu xiang-yun and Li yan (2010 and 2012)applied parallel computing to the magnetotelluric data processing and achieved good effect [17-19].

The study achievements mentioned above are all based on message passing model (MPI or PVM), which are usually suitable for large granularity parallel calculation, and not beneficial to the efficient use of multi-core of CPU and many-core of GPU resources. This paper adopts a CPU + GPU multi-level heterogeneous hybrid parallel computing scheme for MT Occam inversion. The top layer implement the parallelism between nodes using message passing, the middle layer further parallelize each MPI task by OpenMP and implement the parallelism between multi-core in the compute nodes with shared memory way, the bottom layer realize the GPU core calculation through CUDA, thereby to use computing resources and decrease the cost of the cluster to the greatest possible.

The remainder of this article is organized as follow. We begin with description of the 3-level hybrid parallel computing model in Section 2. Then in Section 3 the basic principle of magnetotelluric Occam inversion algorithm is given. The Occam inversion hybrid parallel scheme is presented in Section 4 and some major algorithms have also been discussed in this section. Inversion results of 6 theoretical models are provided and discussed in Section 5. Finally, in Section 6, we offer some concluding remarks on this hybrid parallel scheme.

2. Hybrid Parallel Computing Model

MPI was released in May 1994 as a message passing Interface, it is actually a standard specification of a message passing library, with advantages of numerous message passing systems, is currently the international standard of the most popular distributed storage parallel programming, with the characteristics of portability, ease of use, complete asynchronous communication function, and many other advantages. In the MPI programming model, the calculation is made of one or more processes, each process by calling the library function to receive and send message with other processes. The processor can read and write only local memory and the data exchanges between different memories are implemented through the messaging model.

With the development of computer hardware technology, CPU multi-core sharing the same memory block has been widely used. Open Multi-Processing (OpenMP) is a kind of shared memory architecture API which provides a multithreaded capacity. OpenMP is an open specification for shared memory parallelism. The basic idea behind it is data-shared parallel execution. It consists of a set of compiler directives, callable runtime library routines and environment variables that extend FORTRAN, C and C++ programs. Communication in OpenMP is implicit, this makes the OpenMP programming relatively easy to implement. A loop can be parallelized easily by invoking subroutine calls from OpenMP thread libraries and inserting the OpenMP compiler. The unit of workers in OpenMP is threads. Every thread can access variables in shared cache or RAM. When accessing shared data, it costs almost nothing.

GPU general purpose computation has developed rapidly in recent 10 years thanks to the NVIDIA CUDA parallel computing architecture. CUDA (Compute Unified Device Architecture) is the computing engine in NVIDIA graphics processing units or GPUs that are accessible to software developers through industry standard programming languages. It supports a range of computational interfaces including OpenGL and Direct Compute. CUDA’s parallel programming model is designed to over-come the challenge of intensive computing while maintaining a low learning curve for programmers familiar with standard programming languages such as C or Fortran. CUDA enable high levels of fine-grain data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. So the programmer can partition the problem into coarse sub-problems that can be solved independently in parallel, and then into finer pieces that can be solved cooperatively in parallel. Such decomposition preserves language expressivity by allowing threads to cooperate when solving each sub-problem, and at the same time enables transparent scalability since each sub-problem can be scheduled to be solved on any of the available processor cores.

The above three parallel environments can be fused in together to form a multi-layer hybrid structure, the premise is that the system has multiple nodes and each node has multiple CPU cores and at least one GPU. Under this hybrid structure, we can make better use of the advantages of different programming model. MPI is suitable for large task granularity parallel between computing nodes. OpenMP, which has little messaging overhead, is suitable for processing intra-node medium and small granularity tasks in parallel. GPU architecture usually provide a large number of cores(SP), thus can use more threads (lightweight) for parallel processing and is more suitable for fine-grained intensive parallel computing of large data.

The goal of this article is to realize a multi-level, top-down gradual refinement programming model, and the MT Occam inversion in parallel with this model. Figure 1 is the framework of this hybrid model. The top level is cluster environment where computer nodes exchange information through the network and MPI platform. The middle level is multicore computing nodes (PC). The cores share the main memory and the fetch mode is symmetry. The bottom level consist of GPU stream processors(SP), the SP in some form share the local storage ,constant cache and texture cache. Corresponding to the hardware environment, the parallel software environment from to bottom are respectively the MPI, OpenMP and CUDA. In this model, when computing tasks are submitted to the cluster, they are divided into several subtasks at first, and then the subtasks are mapped to a cluster node through the MPI messaging. Within each node, the tasks are assigned to a CPU core with compilation guidance statements, and the corresponding processing threads are established. When using CUDA for GPU programming, each CPU process controls a CUDA device. The CUDA program divides the data to be processed into more residential blocks, and then executes them in parallel. Under the CUDA programming model, the problem is divided into two parts, one part execute on CPU (the host) and the other part on the device side (display chip).

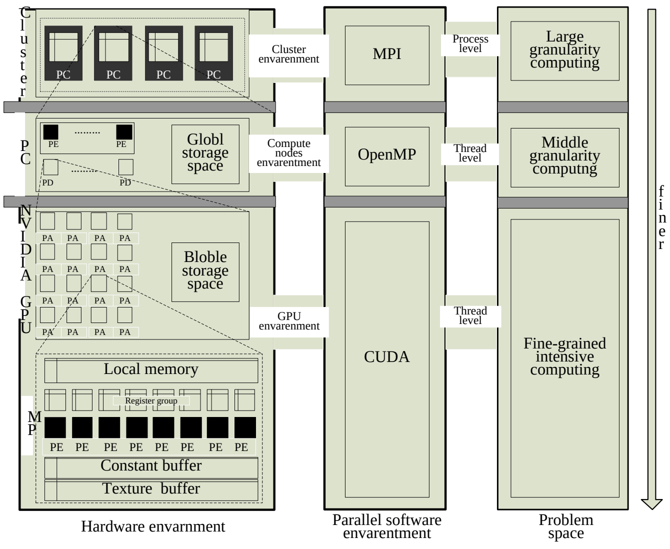

Figure 1. The Basic Architecture of a Heterogeneous Cluster

3. The Basic Theory of MT Occam Inversion

The actual field data in the geophysics are always limited and there inevitably are some errors within the data which will lead to nonunique solutions [22]. In order to obtain the optimal solution, inversion model should be as simple or smooth as possible and the roughness should be as small as possible, on the condition that the misfit is within an expected tolerance, thus to suppress redundant structure. Based on this idea, Constable et al put forward the Occam inversion method in 1987 [23].

MT Occam inversion uses a Lagrange multiplier to balance the model smoothness and misfit. The unconstrained functional is

$$U(m)=\|\partial m\|^2+\mu^{-1}\left(\|Wd-WF(m)\|^2-X_*^2\right)+\|T(m-p)\|^2\ , \tag{2.1}$$

where the μ is a trade-off parameter (Lagrange multiplier), used to balance the model smoothness and misfit. The first term on the right is the roughness and second the misfit weighted by the Lagrange multiplier, ***d*** is the data vector, ***F(m)*** defines the forward mapping, ***R*** is the roughness matrix, $||$***Wd – WF(m)*** $||^2$is the standard 2-d norm, represent the misfit (X^2) of the forward response of the model to the data ***d***, X_*^2 is the desired X^2. $\mathbf{W}$ is the diagonal $n\times n$ matrix

$$\mathbf{W}=diag\{1/\sigma_1,1/\sigma_2,1/\sigma_3...,1/\sigma_n\}, \tag{2.2}$$

serving to standardize the data with the uncertainty in the data. When the data is accurate, to avoid data overflow that will cause the inversion abnormal end, let $X_*^2=0$, $\mathbf{W}=\mathbf{I}$.

The discrete expression of X^2 is:

$$X^2=\sum_{i=1}^{N_d}\frac{\left(d_i-F_i(m)\right)^2}{\sigma_i^2}\quad , \tag{2.3}$$

where d_i is the i^{th} data and σ_i is the uncertainty in the i^{th} datum, $i\in[1,N_d]$, N_d express the data quantity.

the model roughness $R=\|\partial m\|^2$, in a 2-Dinversion $\partial=\begin{bmatrix}\partial_y\\ \partial_z\end{bmatrix}$, where ∂_y and ∂_z respectively express the transverse and longitudinal adjacent model roughness constraint matrix, the discrete expression of R is:

$$R=\sum_{i=1}^{N_{pt}}\left(\sum_{j=1}^{N_m}\partial(j,i)m(j)\right)^2\quad , \tag{2.4}$$

where N_m is the number of parameters, N_{pt} is the number of constraint terms.

Here introducing $\|T(m-p)\|^2$ to constrain the variability of model parameters and fix the conductivity structure, where p is prior information. The diagonal matrix ***T*** serves as a weighting factor, which is used to minimize the horizontal and vertical differences between ***m*** and ***p***.

To linearize the above nonlinear problem in (2.1), refer to formula $F(m_{k+1})=F(m_k)+J_K(m_{k+1}-m_k)$, we can get iteration formula of the inversion:

$$m_{k+1}(\mu)=\left[\mu\left(\partial_k^T\partial_k+TT\right)+(WJ_k)^T(WJ_k)\right]^{-1}\left[(WJ_k)^T\left(W\hat{d}_k\right)+\mu Tp\right]\quad , \tag{2.5}$$

where J_k is the Jacobian matrix, that is, the partial derivative of $F(m_k)$ relative to m_k.

$\hat{d}_k=d-F(m_k)+J_K m_k\ ,k\in[1,ITRMAX]$, where $ITRMAX$ is the limitation of iterations.

Through repeated iterations, and by judging the misfit and roughness, we can get the optimal solution. figure 2 is a basic flow chart of the inversion.

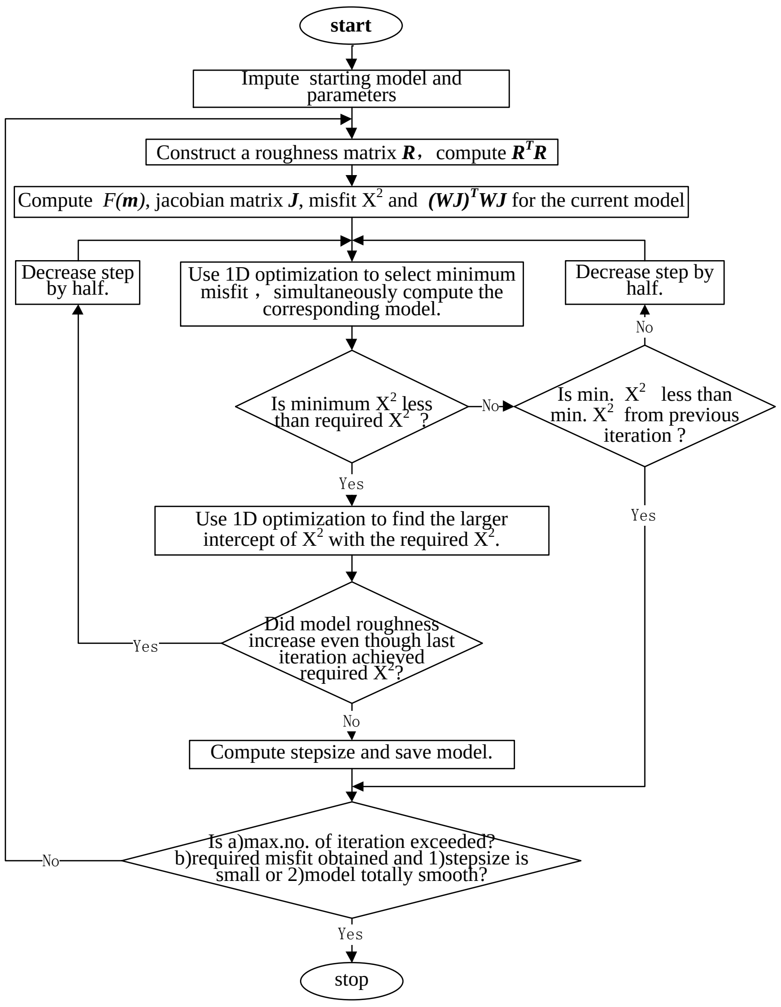
start
Impute starting model and parameters
Construct a roughness matrix R, compute R^TR
Compute $F(m)$, jacobian matrix J, misfit X^2 and $(WJ)^TWJ$ for the current model
Decrease step by half.
Use 1D optimization to select minimum misfit, simultaneously compute the corresponding model.
Decrease step by half.
No
Is minimum X^2 less than required X^2 ?
No
Is min. X^2 less than min. X^2 from previous iteration ?
Yes
Use 1D optimization to find the larger intercept of X^2 with the required X^2.
Yes
Yes
Did model roughness increase even though last iteration achieved required X^2?
No
Compute stepsize and save model.
No
Is a)max.no. of iteration exceeded? b)required misfit obtained and 1)stepsize is small or 2)model totally smooth?
Yes
stop

Figure 2. A Simplified Flow Chart of the Inversion Algorithm

4. Hybrid Parallel Scheme

Through the analysis of the Occam inversion, we know that the algorithm has good gradation. Computations based on grouping frequencies are of large granularity and can be further divided into single frequency computations, which have relatively small granularity. In forward modeling, for each frequency, linear equation group resolving can also be decomposed, thus to make use of the mass core of the GPU. On these grounds, a parallel inversion scheme was designed and the overall process is shown in Figure 3.

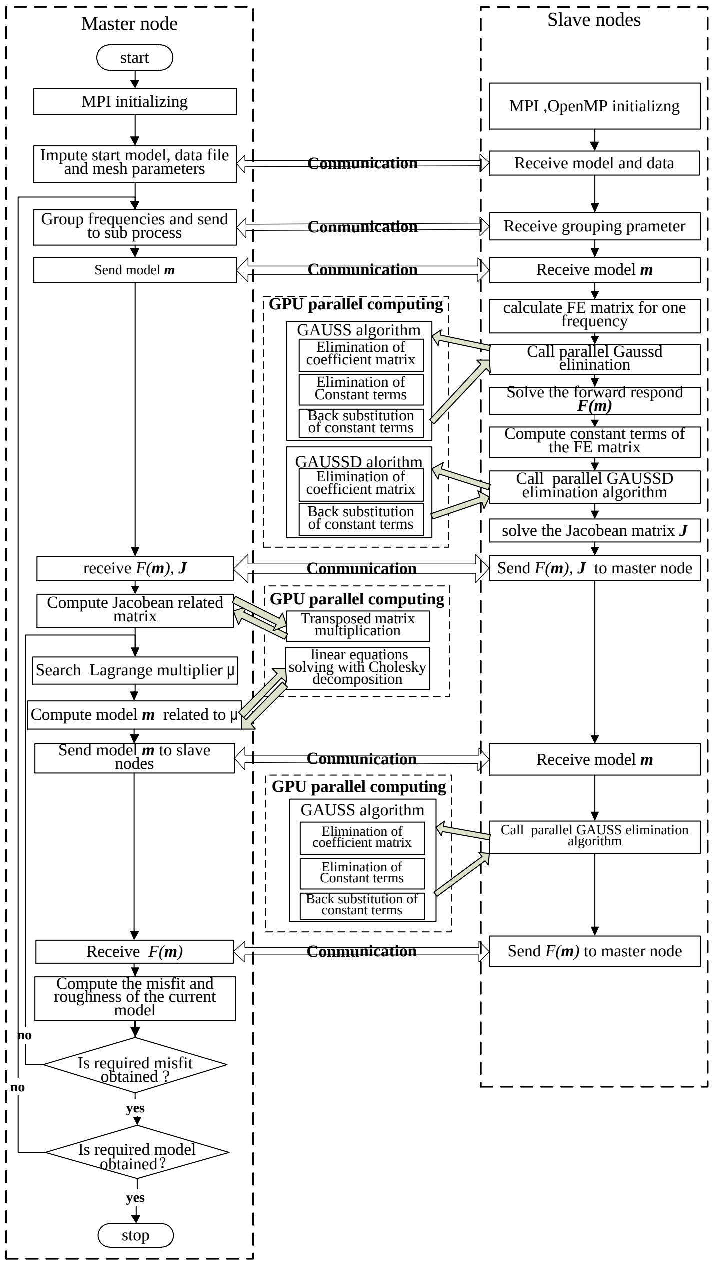
Master node
start
MPI initializing
Impute start model, data file and mesh parameters
Group frequencies and send to sub process
Send model m
receive F(m), J
Compute Jacobean related matrix
Search Lagrange multiplier μ
Compute model m related to μ
Send model m to slave nodes
Receive F(m)
Compute the misfit and roughness of the current model
no
Is required misfit obtained ?
yes
no
Is required model obtained?
yes
stop
Communication
GPU parallel computing
GAUSS algorithm
Elimination of coefficient matrix
Elimination of Constant terms
Back substitution of constant terms
GAUSSD alorithm
Transposed matrix multiplication
linear equations solving with Cholesky decomposition
Slave nodes
MPI ,OpenMP initializng
Receive model and data
Receive grouping prameter
Receive model m
calculate FE matrix for one frequency
Call parallel Gaussd elinination
Solve the forward respond F(m)
Compute constant terms of the FE matrix
Call parallel GAUSSD elimination algorithm
solve the Jacobean matrix J
Send F(m), J to master node
Receive model m
Call parallel GAUSS elimination algorithm
Send F(m) to master node

Figure 3. A Simplified Flow Chart of the Hybrid Parallel Inversion Scheme

4.1. Distributed Storage Parallel Algorithm

The upper level distributed parallel model is used to implement parallelization for different frequency response and Jacobian matrix calculation in the forward modeling process, through task decomposition of large granularity. The main input of the forward calculation is the inversion model **m**, the auxiliary parameters include finite element mesh, survey point location and frequency data. The output is the corresponding frequency's modeling response and Jacobian matrix. Each frequency point's forward calculation is an independent process, thus can be directly partitioned based on frequency big granularity task. Before the computation, the model data and parameters are distributed between computing nodes and then, the forward response and partial derivative are collected through communication. In the process of task mapping, the master node is responsible for reading data, sending data to and receiving data from the slave nodes. The slave nodes receive and send data in passive way, keeping in block state before confirming that the master node has received the data.

In algorithm 1, according to the actual number of nodes in the task mapping the *frequencies* are divided into multiple subset *Fi (I = 1, n, n < nfre)* and each node calculates frequency points within the corresponding subset. Since the calculations for each frequency point are relative equilibrium, for *nproc* nodes, frequencies can be divided equally in sequence, so that each process calculates *nproc/nfre* (+ 1) frequency points. Experiment results show that this task assignment can keep the load balance between nodes. In consideration that master node is responsible for global communication control and have certain overhead, computing tasks mapped to the master node should be the least of all the nodes.

Algorithm 1 forward modeling on distributed memory system

```
input: m
output: F(m),J
  rank ← index of processes ∈ [0, $n_{proc}$−1]
  GLOBAL COMMUNICATION
    BCAST{ frequencies, MESH, NRL}
  {f_s, f_e}← selection(rank)
  GLOBAL COMMUNICATION
    BCAST{m}
  for every i = f_s,…, f_e do
    freq←frequencies[i]
    ·   ·   ·
    Calculate   F(m) [freq]
    if(calculation J) then
      ·   ·   ·
      Calculate   J [freq]
    endif
  end for(i)
  GLOBAL COMMUNICATION
    GATHER{F(m) }
  if(calculation J) then
  GATHER{J}
  endif
```

4.2 Intra-node Parallel Algorithm

The intra-node parallel model includes multi-core CPU and mass core GPU parallel programming model. Through the analysis of Occam algorithm, the parallelism within a node concentrates on the process of finite element coefficient matrix assembly and linear equation group solving with Gauss elimination algorithm in the process of forward modeling, the jacobian matrices calculation and Cholesky decomposition in the process of inversion.

4.2.1. Forward Calculation on Shared Memory System

Based on the shared memory parallel model, the computing tasks mapped to nodes by upper parallel model are further divided and mapped to the multi-core processor system. Based on the same principle, the calculation for each frequency in the process of forward modeling are data independent to each other and share the data such as the inversion model, finite element mesh, survey point locations and frequency data.

Algorithm 2 forward modeling on shared memory system

```
imput: m
create T threads
t ← index of threads
freq[t]←(t)
($k_i$[t], $p_i$[t])← finit Element (freq(t))
K[t] ← K[t]+$k_i$[t]
P[t] ← P[t]+$p_i$[t]
$u$(freq[t])← gauss(K[t], P[t])
if(calculation $J$) then
  $p_i'$[t]← derivative Element (freq[t])
  $P'$[t]←$P'$[t]+$p_i'$[t]
  $u'$(freq[t])← gaussd(K[t], $P'$[t])
endif
join threads
calculate $F(m)$, $J$
```

Algorithm 2 describes the parallel algorithm of forward calculation within a node. If a node is about to calculate T frequency points then T threads will be created before algorithm implementation and a copy of the private variables for each thread will be created at the same time. The thread choose their corresponding frequency *freq[t]* from the frequency group of the node, and use the private variable K[t] to complete the calculation of forward response and partial derivative, t represents the number of threads. The shared memory parallel programming model has characteristic that multi-threads in a node share the same piece of storage space, so the access latencies to the memory are all the same. Since the execution status of a thread are unpredictable at some time points, in order to guarantee the correctness of the data in time, the scheme adopt the way of creating thread private storage space for data isolation.

Using multi-thread parallel processing to complete the forward calculation within a node can reduce the Gaussian elimination method's time complexity to $O(NE^2)$. On the contrary, in order to handle multi-thread shared memory data access conflicts, private copies for each thread will increase the algorithm space complexity accordingly.

4.2.2. CUDA Based 2-D Bandwidth Gaussian Elimination

Finite element forward modeling algorithm adopt the FE-method to divide strata structure into a finite number of non-overlapping units and convert the forward problems into solving partial differential equations.

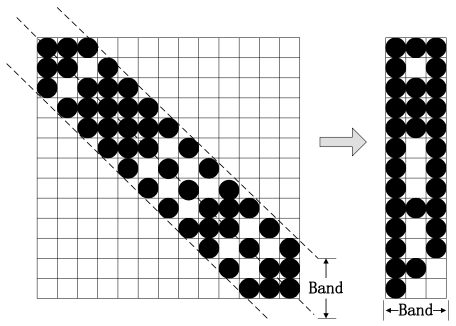

Figure 4. Two-dimensional Bandwidth Storage Structure

In the algorithm, the Gaussian elimination solving the linear equations account for a great proportion of the whole. As FE coefficient matrix has the characteristics of sparse, ribbon, symmetry and phalanx, MT Occam invoke a two-dimensional bandwidth storage structure to store data and complete Gaussian elimination on this compressed storage structure which is shown in figure 4. A basic Gaussian elimination CUDA program is given in Code 1.

Code 1 The process of coefficient matrix elimination

```
offset = (blockIdx%x-1)*height*width
idx = ID(threadIdx%x)
do i=1,NNODE-1
   if(threadIdx%x <= NBAND) then
      pivot(threadIdx%x) = S(offset+threadIdx%x)
   endif
   call syncthreads()
   temp=K(offset+idx)
   temp=temp-pivot((idx/width)+1)/pivot(1)*pivot(idx/width+MO
   D(idx,width))
   K(offset+idx) = temp
   if(threadIdx%x <= NBAND .and. threadIdx%x /= 1) then
      K(offset+threadIdx%x) = pivot(threadIdx%x)/pivot(1)
   endif
   offset = offset+width
   call syncthreads()
enddo
```

In Code 1, offset represent the location before the first element of first row of the coefficient matrix in each block. The matrix moves up row by row in the loop and synchronization function syncthreads() has been invoked by each cycle for threads synchronization in a block. idx express thread relative position in the working triangle.

In order to reduce the traffic of global memory and improve the CUDA application efficiency, data are firstly loaded into the shared storage and then read out by each thread to complete the calculation.

4.2.3. CUDA based Matrices Computation and Cholesky Decomposition

NVIDIA now supports CUDA implementation of BLAS function library called cublas which provides support for transposed matrix multiplication. We directly invoke the routine to calculate cross-product matrix and the key codes are presented in Code 2. In calculation of matrices that depend on Jacobian, Occam algorithm firstly by partial derivative calculation to get weighted Jacobian matrix *WJ*, then calculate WJ^TWJ. The low dimension is *ND* and the high is *NP*, so the first column of matrix *WJ* is saved first in accordance with the principle of column-major order in Fortran. But in the real calculation, the data are calculated from top to bottom row by row, and this sequence will cause some cache conflicts. Therefore, the original program has been modified by storing WJ^T instead of *WJ*, so as to keep the data calculation order same with the storage order to increase the cache hit. In the transposed matrix multiplication, let $A = WJ^T$, and the original equation change to $AA^T = WJ^TWJ$.

Code 2 Cross-matrices computation on CUDA

```
lerr = cublasCreate(h)
d_WJT = WJT
I = cublasSsyrk_v2(h, CUBLAS_FILL_MODE_UPPER,
CUBLAS_OP_N, nParams, ND, 1., d_WJT, nParams, 0.,
d_WJTWJ, nParams)
WJTWJ = d_WJTWJ
```

CULAtools also provide the CUDA version of LAPCK routine library and we realize the

Cholesky decomposition by calling device kernel cula_device_SPOSV. The key codes are presented in Code 3.

Code 3 Cholesky decomposition on CUDA

```
lerr = cula_initialize()
d_aMat = A
d_pwk4(:,1) = B
lerr = cula_device_SPOSV('U', N, 1, d_aMat, N, d_pwk4, N)
CALL CULA_CHECK_STATUS(lerr)
x(:) = d_pwk4(:,1)
```

The output of matrix multiplied by its transpose is always symmetric matrix, so in calculating the cross-product matrix, only the upper triangle need to be saved, thus to reduce the storage space.

5. Implementation and Experimental Results Analysis

5.1. The Testing Platform

All testing are implemented on the small heterogeneous cluster we have built. The cluster includes 4 compute nodes connected via gigabit network. Each compute node configure an Intel Core i5-3470 CPU with 4 physical cores and 8 GB dual channel DDR3 host memory, and an NVIDIA GTX 680 GPU with 1536 stream processing units and 2GB GDDR5 device memory. As a whole, this platform provides 16 CPU cores and four GPUs. The operating system is CentOS 6.4 and the development tool is the Cluster Development Kit of PGI company, which includes C/C + + and Fortran compilers that directly support MPI, OpenMP and CUDA programing model and provides performance analysis and debugging tools at the same time.5.2 Inversion model design.In order to assess the actual execution efficiency of the MT Occam algorithm in parallel, six inversion models of two scales (only model 1 and 5 are shown) have been designed. Four main parameters of two scale model are shown in table 1. The largest number of inversion is limited to 20.

Table 1.The Parameters of Inversion Model

scale	N_r	N_m	NE （y×z）	nfre
1	21	889	103×35	20
2	41	1549	193×35	36

On the basis of scale 1 model, scale 2 increase the frequency number and measuring points, refine the finite element grid and the inversion model mesh.

Model 1 (see Figure 5) is designed as a horst with resistivity of 1000 Ω.m and the cross section size of 5 Km by 12.5 Km. The upper part is a 100 Ωm homogeneous layer. Model 2 (see Fig. 6) is designed as 2 2D prisms with different resistivity in the uniform half space. The resistivity of the half space is 100Ω.m which is covered by a high resistance layer. For scale 1 models, the survey points are fixed to 21 and the forward grid is 105(vertical) x 35(horizontal). The inversion grid number (model blocks) is 889 and the number of frequency points is 20. For scale 2 models, the number of survey points increases to 41 and forward grid density increase to 193 x 35. Accordingly, the inversion grid number increases to 1549 and the number of frequency points increases to 36. The starting model of inversion is 100 Ω.m homogeneous half spaces. In order to simulate real environment, 5% random noises have been added to the theoretical data.

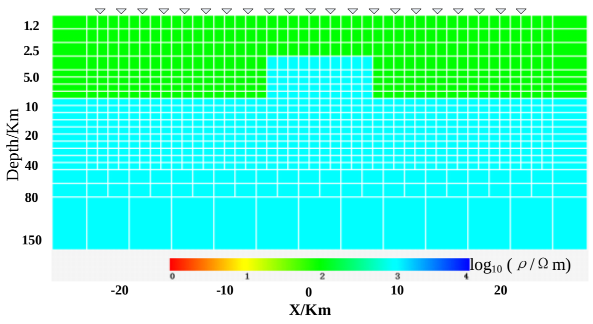

Figure 5. Schematic Diagram of Model 1(scale 1)

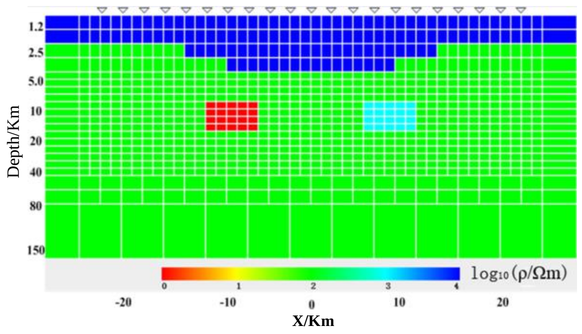

Figure 6. Schematic Diagram of Model 5(scale 1)

5.3. The Experimental Results Analysis

We have calculated inversions for 6 model of scale 1 with serial and hybrid parallel algorithms. The deviation values are defined as the differences (absolute value) of model misfit and roughness in each iteration between serial and parallel algorithms.

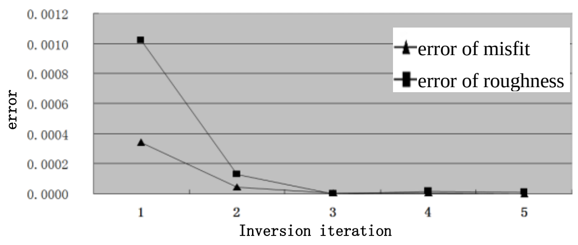

Figure 7. Error Curve of Model 1(scale 1)

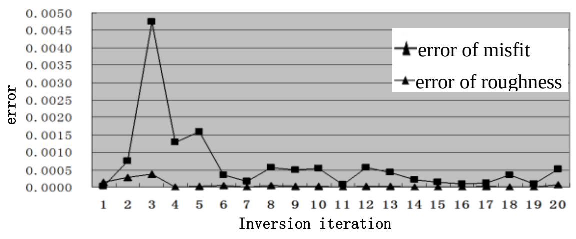

Figure 8. Error Curve of Model 5(scale 1)

The inversion process statistical results are given in Table 2 and Table 3 respectively for serial and parallel algorithms. The results indicate that there is a subtle difference of forward calculation counts between serial and parallel modes with some individual models. By program tracing and analysis, we found that it is caused by different floating point arithmetic precision of CPU and GPU and the final inversion results have not been affected.

Table 2. Inversion Process Statistical Results of Scale 1 Models

Model	Iterations used		Forward counts		Execution time(ms)		Speedup ratio
	serial	parallel	Serial	parallel	serial	parallel	
1	5	5	69	69	125.86	20.84	6.04
2	20	20	156	160	324.23	55.14	5.88
3	15	15	129	129	261.90	48.48	5.40
4	6	6	58	58	113.82	16.00	7.11
5	20	20	134	134	298.02	48.76	6.11
6	10	10	205	219	371.76	181.37	2.05

Table 3. Inversion Process Statistical Results of Scale 2 Models

Model	Iterations used		Forward counts		Execution time(ms)		Speedup ratio
	serial	parallel	Serial	parallel	serial	parallel	
1	5	5	53	53	510.97	36.27	14.09
2	20	20	208	169	2317.87	139.42	16.62
3	20	20	161	161	2185.22	130.89	16.70
4	6	7	54	64	526.84	46.15	11.42
5	20	20	509	248	3947.62	168.02	23.49
6	20	20	185	187	2254.37	142.41	15.83

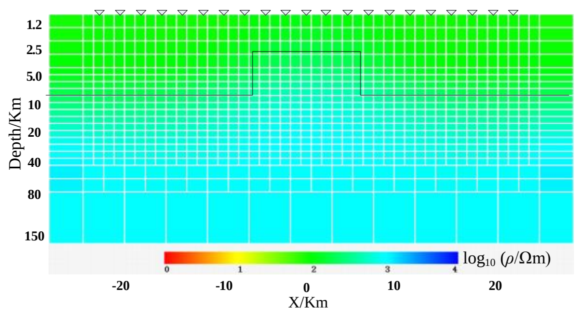

Figure 9. Inversion Result of Model 1(scale 1)

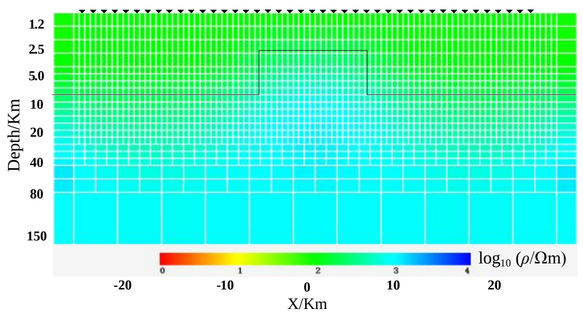

Figure 10. Inversion Result of Model 1(scale 2)

Figure 9-12 show the output results of the parallel inversion for model 1 and model 5 of scale 1 and scale 2, with black solid line to outline the theory model structure. In the case of model 1, inversion results has no much differences between scale 1 and scale 2 and the inversion process is relatively stable. But as to model 5, the inversion results of scale 2 are relatively closer to the theoretical model compare to scale 1. Compare this hybrid parallel scheme with pure messaging model of document [16, 17, 18, 21], the speedup ratio have large improvement and the scheme is feasible. In our tests, the speedup ratios vary from model to model, depending on the amount of calculation mapped to different levels. On the whole, the models with larger calculation such as scale 2 models have lager speedup ratio. In the case of 4 nodes environment we have built, scale 1 models obtain the maximum speedup ratio of 7.11 and scale 2 models achieve a speedup ratio as high as 23.49, showing that this scheme is more suitable for models with multi survey points, multi frequency points and big density grid.

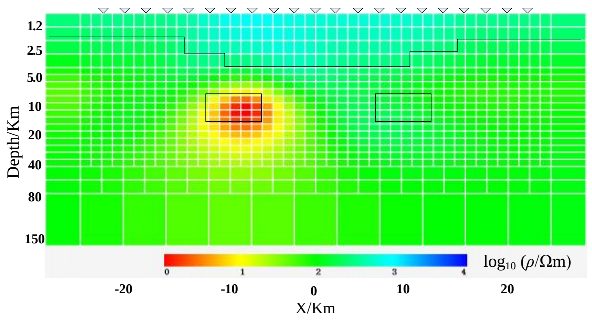

Figure 11. Inversion Result of Model 5(scale 1)

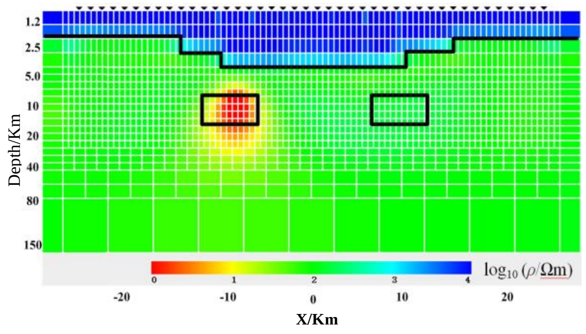

Figure 12. Inversion Result of Model 5(scale 2)

6. Conclusions

In this paper, we have constructed a hybrid programing environment for a small cluster by combining the message passing interface MPI, OpenMP programming and CUDA programing based on share memory. We realized the MT Occam inversion algorithm in this environment, including Gaussian elimination, Jacobian matrix calculation, cross products and Cholesky decomposition. We utilized several theoretical models to test this hybrid method and have deeply analyzed the experiment results.

MT Occam algorithm has relatively large amount of calculation, and their potential parallelism of multiple level is obvious especially in its forward calculation. So it can be implemented with a hybrid parallel model, on which the compute tasks with different granularity are mapped to different level. Compare with pure message passing interface, the hybrid parallel model can achieve greater speedup ratio. In our experiments, the speedup ratio is up to 23.49. The study also show that with the constant division of computing tasks, the communication overhead also rise gradually, with the maximum of 30% of the total computing time in our three level model. This indicates that data communication has become the main limiting factors of this solution efficiency and further optimization for communication need to be down.

Acknowledgments

The work reported in this paper has been supported by The National Natural Science Foundation of China under research project 41264005 and project 41374079. The authors would like to express sincere gratitude to professor Steven Constable for providing the source code of MT Occam inversion. We would also like to thank the anonymous reviewers for their comments and valuable suggestions.

References

[1] Constable S.C, Parker R L, Constable C G. Occamy The National Natural Science Foundation of China under research project 41264005. The authors would like to express sin.
[2] Catherine deGroot-Hedlin and Steven Constable. Occam inversion: a practical algorithm for generating smoothrical Anomaly[J].J. Geomag. Geoelectr., 1993,45: 985–999
[3] Ou Yang L H , Wang J L , Wu J S. The Application of OCCAM method to the inversion of surface wave dispersion curves[J]. Computing Techniques for Geophysical and Geochemical Exploration, 2003, 25(1):1-4
[4] Yang C F , Lin C Y , Chen J Y, et al. One-dimensional inversion for the magnetotelluric data in Lanzhou region by OCCAM'S and generaL inverse methords[J]. Northwestern Seismological Journal, 2002, 24(4): 289-294.
[5] Zhou D Q, Tan H D,Wang W P. The OCCAM inversion in FAEM data processing[J].Geophys Ical & Geochem Ical Exploration, 2006,30(2):162-165
[6] Hu Z Z , Hu X Y , Wu W L,et al Compared study of two-dimensional magnetotelluric inversion methods.Coal Geology &Exploration, 2005, 33(1): 64-68.
[7] Wu X P, Xu G M.Improvement of Occampared study of two-dimensionalta Geophysical Sinica, 1998, 41, (4): 547-554.
[8] Siripunvaraporn W, Egbert G. An efficient data-subspace inversion method for 2-D magnetotelluric data[J]. Geophysics, 2000,65(3):791-803
[9] Chen X B.New forward and inversion algorithms and a visual intergrated system for MT data[D].Institude of Geology,China Seismological Bureau,2003
[10] Chen X B,Zhao G Z,Tang J,et al. An adaptive regularized inversion algorithm for magnetotelluric data[J].Chinese Journal Of Geophysics, 2005,48(4):937-946
[11] Newman GA, Alumbaugh DL. Three-dimensional massively parallel electromagnetic inversion elluric data[J].Chinese ,China Seismological Burea
[12] Zyserman FI,Santos with domain decomposition for three-dimensional magnetotelluric modeling[J]. J.Appl.Geophys, 2000,44(4):337-351.
[13] Chen J C, Dai G M. Micro-Computer Networked Computing and 2.5-D CSAMT Forward Parallel Computing[J]. Computing Techniques for Geophysical and Geochemical Exploration, 1997,19(2): 103-107.
[14] Tan HD,Tong T, Lin C H. The parallel 3D magnetotelluric forward modeling algorithm[J]. Applied Geophysics, 2006,3(4): 197-202.
[15] Liu Y,Wang J Y.PC cluster based magnetotelluric 2-D Occamforward modeling algorithm[J]. Applied Geophysics, 2006,3(4): for petroleum, 2006,45(3):311-315.
[16] Liu Y. PC-Cluster Based Parallel Computation Research on 2-D Magnetotelluric Occam Inversion[D]. China University of Geosciences,Wuhan.2006
[17] Li Y,Hu X Y, Kim K, etal. Research of 1-D magnetotelluric parallel computation based on MPI[J].Progress in Geophysics, 2010,25(5):1612-1616.
[18] Li Y,Hu X Y,Wu G J,et al.Parallel computation of 2-D magnetotelluric forward modeling based on MPI.Seismology and Geology,2010,32(3):392-401.
[19] Hu X Y,Li Y,Yang W C,et al. Three-dimensional magnetotelluric parallel inversion algorithm using data space method[J]. Chinese Journal Of Geophysics, 2012,55(12): 3969-3978
[20] Lin C H,Tan H D,Tong T.Parallel rapid relaxation inversion of 3D magnetotelluric data[J],Applied Geophysics, 2009,6(1): 77-83.
[21] Chen L J.Research on parallel computation of 3-D borehole-surface EM based on MPI[D]. Chengdu University of Technology,2006
[22] Wang J Y. Geophysical inversion theory [M] Wuhan: China university of geosciences press.1998
[23] Constable S C, Parker R L, Constable C G. Occam-D borehole-sur practical algorithm for generating smooth models from EM sounding data[J]. Geophysics,1987,52(1):289-300

Authors

Yu Liu, He was born in China in 1961 and received his Ph.D. degree at the China University of Geoscience in 2006. He has been a professor of computer science at the Guilin University of Technology in China since 2007. His research interests are in parallel computing technique and geophysical inversion.

Renhao Xiong. He was born in China in 1987 and received M.S. degree in computer science at the Guilin University of Technology in China in 2016. His current research interests include parallel algorithms and scientific computations.

Yi xiao, He was born in China in 1989 and received M.S. degree in computer science at the Guilin University of Technology in China in 2015. His current research interests include parallel algorithms and scientific computations.