Tu C 06

Cluster Implementation of Low-Rank Multifrontal Direct Solver for 3D Helmholtz Problem

S. Solovyev* (Institute of Petroleum Geology & Geophysics SB RAS), V.I. Kostin (Institute of Petroleum Geology & Geophysics SB RAS) & D. Vishnevskiy (Institute of Petroleum Geology & Geophysics SB RAS)

SUMMARY

The modern methods of processing the geophysical data, such as Reverse Time Migration (RTM) and Full Waveform Inversion (FWI) require solving series of forward problems where the main step is solution of Systems of Linear Algebraic Equations (SLAE) of big size. For big sizes, it is time and memory consuming problem.
In this paper, we present a parallel direct algorithm to solve boundary value problems for 3D Helmholtz equation discretized with help of finite differences. The memory consumption has been resolved due to Nested Dissection approach, low-rank approximation technique and HSS format. OpenMP parallelization is based on standard BLAS and LAPACK functionality. For MPI parallelization, we propose a novel algorithm that uses dynamical distribution of the elimination tree nodes across cluster nodes. Numerical experiments show performance benefits of the proposed cluster algorithm compared to the not parallel version and demonstrate significant memory advantages over direct solvers without low-rank approximation.

Introduction

Reverse Time Migration (RTM) and Full Waveform Inversion (FWI) require solving series of forward problems where the main step is solution of Systems of Linear Algebraic Equations (SLAE) of big size which are time and memory consuming problems.
In this paper, we present a parallel direct algorithm to solve boundary value problems for 3D Helmholtz equation discretized with help of finite differences. The memory consumption has been resolved due to Nested Dissection approach, low-rank approximation technique and HSS format. For MPI parallelization, we propose a novel algorithm that uses dynamical distribution of the elimination tree nodes across cluster nodes. Numerical experiments show performance benefits and demonstrate significant memory advantages of the proposed cluster algorithm over direct solvers without low-rank approximation.

Method

The equation under consideration is the Helmholtz equation

$$\Delta u + \frac{(2\pi\nu)^2}{V^2} u = \delta(\bar{r} - \bar{r}_s) f , \qquad (1)$$

for displacement u in some parallelepipedal domain D. The domain is surrounded with Perfectly Matched Layers used to decrease wave reflection from outer boundaries. In (1), ν is frequency, V – sound velocity, $\bar{r}_s$ – source coordinate, f – source. To get a SLAE $AU = F$ we approximate (1) using 27-point finite difference stencil on a parallelepipedal grid. We apply LDL^t factorization to the complex symmetric sparse coefficient matrix A (Collino, 2001).
To decrease the fill-in of *L*-factor we use the Nested Dissection Reordering approach. To get additional performance improvement and decrease memory consumptions we use low-rank approximation of the off-diagonal blocks of *L*-factor and compress the diagonal blocks by HSS technique (Xia, 2012). Low-rank approximation is performed by blocking modification of the cross approximation technique proposed by the author (Solovyev S, Tordeux S, 2015). Patterns of *L*-factor and low-rank/HSS compressed *L*-factor are presented in the Figure 1 with grey scale colouring - light grey blocks are sparse, the grey is lighter for smaller sparsity, and the black blocks are dense.

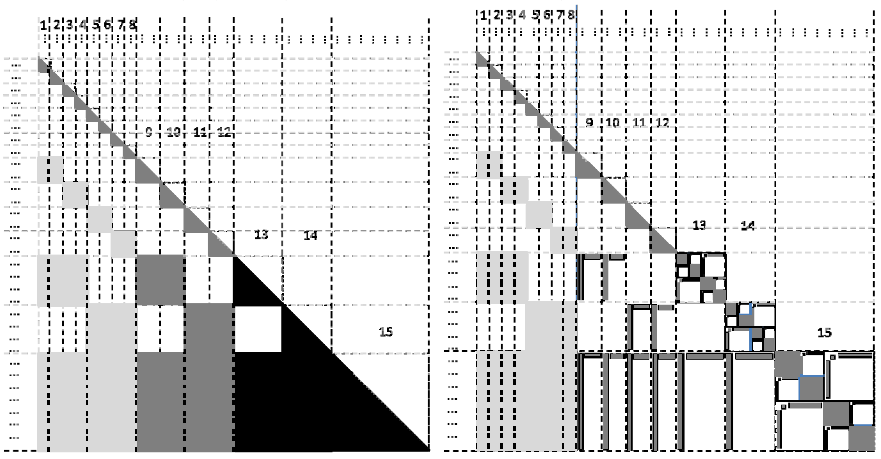

Figure 1 *Left image: patterns of L-factor coloured using grey scale. Right image: some - subdiagonal blocks are low-rank approximated; diagonal blocks are presented in HSS format.*

During factorization process, sub-diagonal *m*-by-*n*-blocks of *L*-factor are approximated by products of *m*-by-*r* and *r*-by-*n* with some comparatively small value *r*. Necessity to compute low-rank approximations may look like some increase in computational costs. In fact, the total computational time even decreases because low-rank matrices are used for computing the Schur complements.
Due to low-rank approximation used in factorization, we obtain an approximation $\tilde{L}$ of *L*-factor. The solution $\tilde{x}$ of the system of linear equations $\tilde{L}D\tilde{L}^t\tilde{x} = b$ approximates the solution of initial system.

Approximation errors directly correlate with low-rank approximation errors. This correlation and the details of algorithms are described in the paper (Solovyev S, 2014).
After applying the Nested Dissection reordering, the factorization process and L-factors are presented by binary elimination tree (see Figure 2). The elimination tree describes dependencies among nodes to be processed: any node can be processed only after completion of processing all its children nodes whereas the nodes at the same level can be processed concurrently. Columns of *L*-factor from different blocks of the same elimination tree level are independent and can be computed concurrently (see columns panels 9 and 10 in the Figure 2). On the other hand, the factorization process of the columns from some node cannot be started before completing factorization matrix columns from all children nodes. So, block 13 can be factorized after completing the factorization nodes 1,2,3,4,9,10.

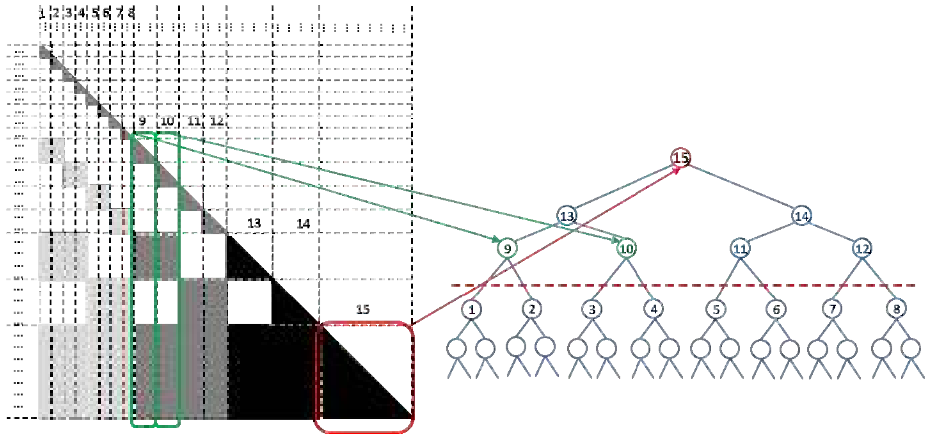

Figure 2 *The pattern of L-factor in exact arithmetic (left image) and it elimination tree (right image).*

We distribute matrix columns across cluster nodes by the following way: one cluster node contains columns of an elimination sub-tree located below a red line; upper this line each Elimination tree Node (EtN) belongs one Cluster Node (CN) (Figure 2, right image). If some block does not fit cluster node RAM then this block is split in few parts. The pattern of L-factor becomes a bit modified (Figure 3, left image) and the elimination tree becomes semi-binary, i.e. one node has either one or two children (Figure 3, right image).

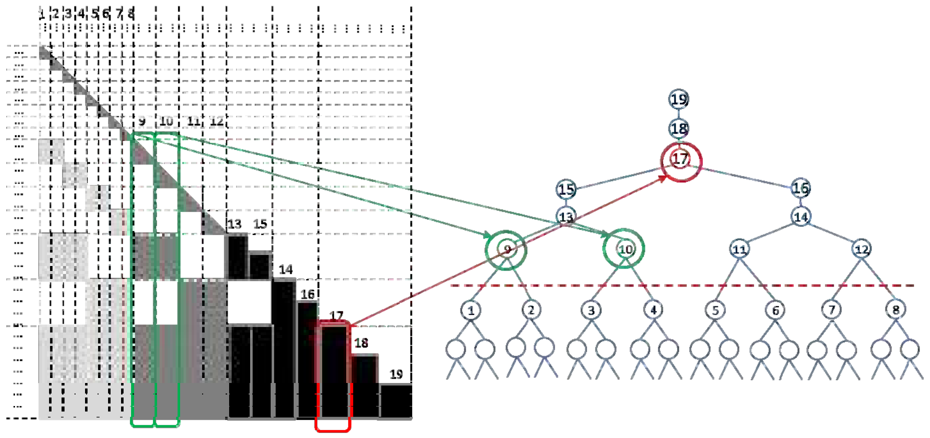

Figure 3 *The patterns of L-factor adapted to the cluster version (left image) and the elimination tree (right image).*

The factorization process starts from bottom levels of elimination tree by factorizing sub-trees simultaneously on different cluster nodes without any MPI communications. The factorization of the elimination tree nodes (located upper red line) performs using MPI data received from their children nodes. These data are necessary to compute Schur complement via updating current matrix columns by the previous factorized data. At the end of factorization process, the distribution of *L*-factor across cluster nodes is similar to initial matrix *A* distribution. The mapping columns on cluster nodes can be predicted before the factorization process start because the number of non-zero elements in *L*-factor can be estimated just after reordering step.

Use of this cluster algorithm in the low-rank multifrontal solver would not be optimal. The low-rank solver needs less memory, so it should be able to solve problems bigger than the direct solver. In this case, the mapping columns on the cluster nodes cannot be predicted, because the low-rank approximation is performed "on the fly" and we don't know the size of compressed L-blocks in advance. Therefore, we propose the dynamical elimination tree nodes distribution, which will describe on special case.
Let us have a cluster with 9 nodes and a SLAE which cannot be solved on this cluster by direct solver because of lack of memory. The first step is distributing EtNs between CNs (we suppose that each EtN will contain part of initial matrix and temporary data, size of them equal the non-compressed block of L-factor). Because of lack of memory, not all EtNs are distributed. In our example, each CN marked as "green" performs factorization of some EtN circle (marked as "black") (Figure 4, left image).

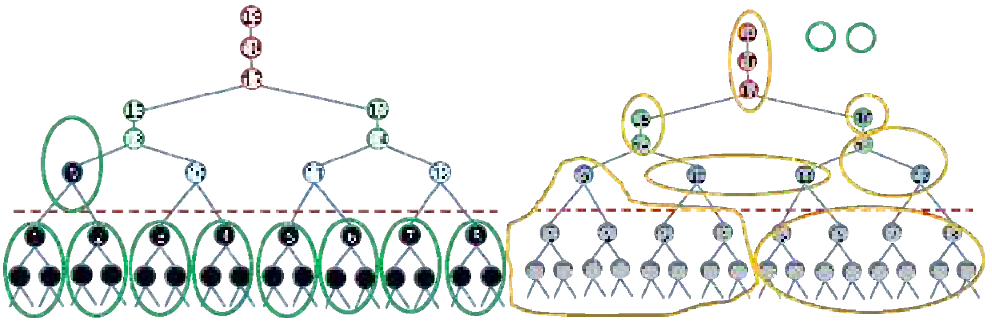
Figure 4 *Data distribution across cluster nodes at the first step of factorization process (left image) and after the end step (right image).*

At the end of the first step all factorized data are compressed (low-rank approximation) and redistributed across CNs, so factorized data are collected on the some CNs, other CNs become free-of-data. At the second step, free-of-data CNs receive next EtNs to factorize, other CNs save factorized data and send necessary data to other nodes for updating (computing Schur complements).
Therefore, while performing factorization process there are two types of cluster nodes: nodes that keep factors (factorized elimination tree nodes) in RAM and send them by request to nodes that perform factorization. As the factorization process goes on, the cluster nodes may change specialization - more clusters nodes save factorized data and less nodes perform factorization. Finally, all factorized data are distributed across the cluster nodes (marked as "yellow"), but some cluster nodes can be free-of-data (marked as "green") (Figure 4, right image). Proposed scheme is memory effective and can solve problems if compressed L-factor fits total cluster RAM (CRAM).

Numerical experiments

In the first test, we compare performance of the cluster version of our algorithm with one-node version. Computational domain is a cub 251x251x251, number of unknowns is 16*10^6, internal low-rank accuracy is 10^-6, number of OMP threads is 4, number computational nodes on cluster – 32. Direct solvers need about 800G to solve such problem, the proposed low-rank solver needs about 3 times less – 240G. Table 1 shows parameters of computational resources and timing. The "Processor performance ratio" was estimated on the small tests and shows speedup of server processor against cluster one.
The second test aimed at demonstrating performance and memory consumption on the large problems. We took inhomogeneous cluster with one 1000GB RAM node and four nodes with 96GB RAM . Operation system cash uses about 10% of all memory, so we have about 1250G free RAM. Each node has Intel(R) Xeon(R) CPU X5675 @ 3.07GHz SSE4.2 with 12 threads.
The computational domain is 12km*12km*6km, spatial discretization with step 30m, frequency 4Hz, velocity 2400m/s. size of PML is 600m (10 grid points) in each direction. Therefore, the grid is 460x460x240=~50*10^6 grid points.

Any direct solvers require about 3 500G total free RAM to handle with such problem. Proposed low-rank one uses about 860G RAM of cluster and spend about 14 hours to factorize data and about 3 min to inverse LDL^t factors with one right hand side.

	Server, 512G RAM	Cluster NSK-30T, 16G RAM
Processor Intel(R) Xeon(R)	E5-2690 v2 @ 3.00GHz AVX	E5540 @ 2.53GHz SSE4.2
Processor performance ratio	x2.2	x1.0
Factorization time	17 198 (x2.2)	7 715 (1x)

Table 1 *Comparison of one-node version with cluster one.*

Conclusions

We have presented a parallel algorithm of the multifrontal low-rank sparse solver to solve 3D Helmholtz problem. Proposed algorithm targets on HPC systems with distributed memory (clusters) by using hybrid OpenMP and MPI parallelization. OpenMP parallelization based on standard BLAS and LAPACK functionality. MPI parallelization is performed by novel proposed algorithm, which is "key point" of the paper. It is based on dynamical distribution of the elimination tree nodes across cluster nodes. The numerical experiments show the performance benefits and memory advantages over direct solvers without low-rank approximation.

Acknowledgements

The research was supported by RFBR grants 14-05-00049, 14-05-93090, 16-01-00460, 16-05-00800. Computations were performed on cluster NSK-30T (Siberian Supercomputer Centre).

References

Collino F. and Tsogka C. [2001] Application of the PML absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media. *Geophysics*, **66**(1), 294-307.

Solovyev, S.A. [2014] Application of the Low-Rank Approximation Technique in the Gauss Elimination Method for Sparse Linear Systems. *Vychisl. Metody Programm*. **15**, 441-460 [in Russian].

Solovyev, S.A., and Tordeux, S. [2015] An efficient Truncated SVD of large matrices based on the low-rank approximation for inverse geophysical problems. *Siberian Electronic Mathematical Reports*, **12**, 592-609.

Xia, J. [2012] Robust and efficient multifrontal solver for large discretized PDEs. *High-Performance Scientific Computing*. London: Springer, 2012. 199-217.