Improving MPI-Scalability of Multifrontal Direct Solver for 3D Helmholtz Equation with Data Compression

S. Solovyev, V. Kostin

Institute of Petroleum Geology and Geophysics SB RAS

Abstract. We present a MPI based direct solver for a system of linear equations that arise as finite difference approximations of a boundary value problem for the 3D Helmholtz equation with Perfectly Matched Layers. The solver uses data compression (Low Rank approximation and Hierarchically Semi Separable formats for storing intermediate data) to reduce memory consumption and improve performance. For good scalability of the solver, factorization of matrix columns that correspond to the middle level of the elimination tree is combined with computations of Schur complements at the highest level of the elimination tree. This trick helps to achieve scalability close to ideal for up to 16 cluster nodes though results are not so impressive on bigger clusters.

Keywords: Helmholtz equation, Low-Rank approximation, HPC systems

1 Method

The equation under consideration is the Helmholtz equation

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

for the pressure $u(r)$ in some parallelepiped domain D. In (1), ν is the frequency, $V = V(\overline{r})$ – the sound velocity, $\overline{r}_s$ – the source position, f – the source function. The domain is immersed in some bigger parallelepiped domain D_{PML} with Perfectly Matched Layers (PMLs) [1] to decrease waves reflection from outer boundaries.

To get a system of linear equations (SLAE) $AU = F$, we approximate (1) using a 27-point finite difference stencil on an equidistant parallelepiped grid. Matrix A is sparse, complex valued (due to PMLs) and symmetric (but not Hermitian). It is decomposed by use of LDL^T-factorization. Lower triangular matrix L of this factorization is also sparse though has much more nonzero elements than matrix A. This phenomenon is called *fill-in* - the term reflects situation where zero elements of lower triangle of matrix A become nonzero in factor L. The fill-in depends on the order used for matrix rows indexing (and

columns, respectively). To reduce the fill-in, the order is defined by the Nested Dissection Algorithm [2].

To further decrease memory consumption, we use low-rank approximation of the off-diagonal blocks of L-factor and compress the diagonal blocks by HSS technique [3]. This also helps to reduce floating point operations count. Low-rank approximation is performed by block modification of the cross approximation technique proposed in [4]. Patterns of L-factor and low-rank/HSS compressed L-factor are presented in the left image of Figure 1 where grey shading is used to illustrate fill-in factor in matrix' blocks (ratio of the number of nonzero elements to the total number of block elements) - the darker the block the bigger the respective fill-in factor. White color means the block is fully zero (the fill-in factor is zero) and for the black blocks the fill-in factor is close to one.

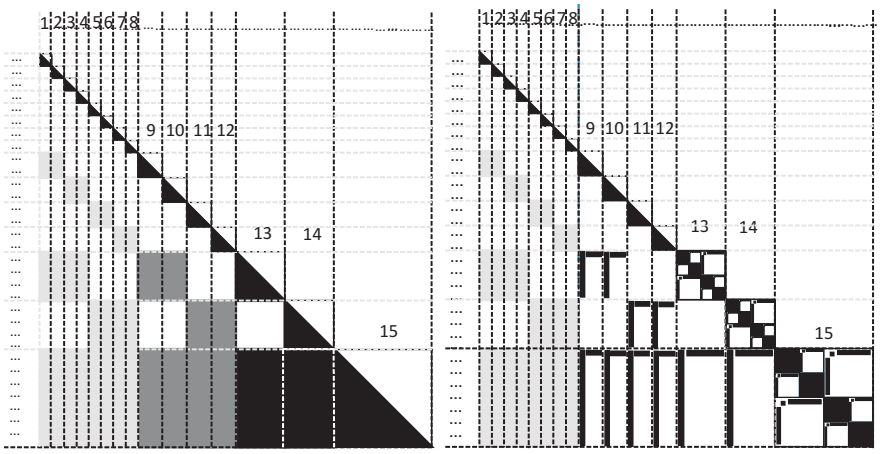

Fig. 1. Left image: patterns of L-factor coloured using grey scale. Right image: some - subdiagonal blocks are low-rank approximated; diagonal blocks are presented in HSS format.

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 decreases because low-rank matrices are used for computing the Schur complements.

Due to low-rank approximation, we obtain an approximation $\tilde{L}$ of L-factor. The solution $\tilde{x}$ of the system of linear equations $\tilde{L}D\tilde{L}^T = b$ 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 [5].

After applying the Nested Dissection reordering, the factorization process and L-factors are presented by binary Elimination Tree (ET) (see Figure 2).

The ET nodes (ETNs) correspond to matrix panels. The ET 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 at the same ET 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 factored after completing the factorization nodes 1,2,3,4,9,10.

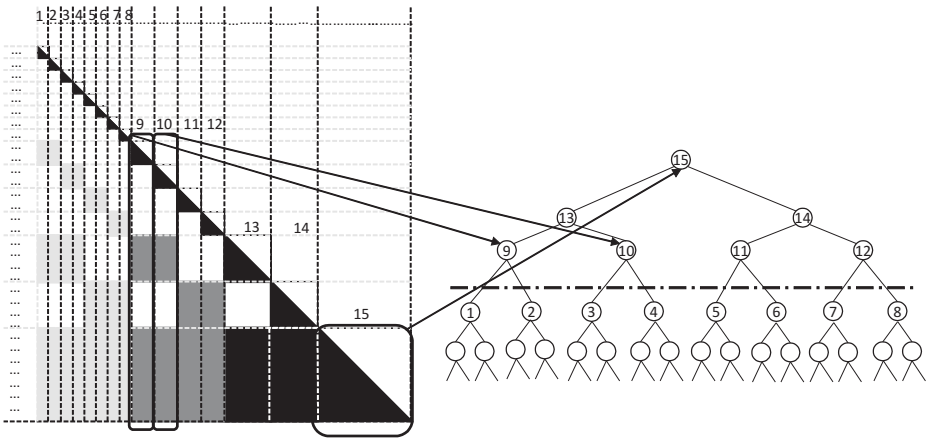

Fig. 2. The pattern of L-factor in exact arithmetic (left image) and it elimination tree (right image).

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

The factorization process starts from bottom levels of ET by factoring subtrees simultaneously on different CNs without any MPI communications. The factorization of the matrix panels that correspond to ETNs located upper red line is performed with use of data received from their children nodes. These data are necessary to compute Schur complement via updating current matrix columns by the previously factored data. At the end of factorization process, the distribution of L-factor across CNs is similar to initial matrix A distribution. The mapping columns on CNs can be predicted prior to the factorization process start because the number of non-zero elements in L-factor can be estimated just after reordering step.

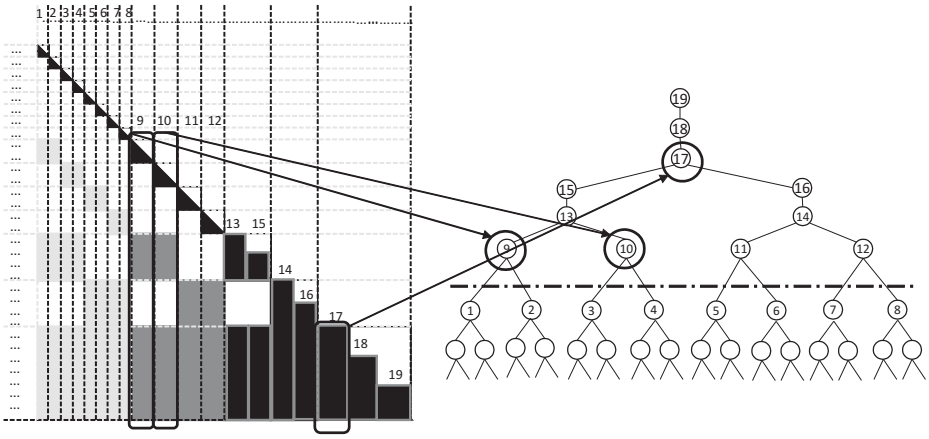

Fig. 3. The patterns of L-factor adapted to the cluster version (left image) and the elimination tree (right image).

The sketched above cluster algorithm is referred below as the basic MPI version. The basic version might be not optimal. As far as using low-rank approximation helps to save RAM, an optimized algorithm should be able to solve problems of bigger size than with the basic version. Below it will be shown that the scalability of the basic version may be an issue.

Two ways of improving the basic MPI version can be applied: the "memory-saving" (MS) approach with dynamical distribution of the ETNs and the "performance-efficient" (PE) approach with preliminary distribution of jobs across CNs.

The basic version of the algorithm as it is explained above assumes some kind of correspondence between the ETNs and CNs. In a case of very big problems such correspondence is impossible to establish. In the described below MS algorithm along with factorization data are collected on CNs designated to store results. The disadvantage of such approach is excluding of CNs assigned to store data from the computational process.

The idea of MS algorithm is explained on an example. Assume we 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 across CNs. Each ETN contains a part of the matrix and temporary data of size of the non-compressed block of L-factor. Because of lack of memory, not all ETNs are distributed. In the left image of Figure 4, each CN marked with dashed lines performs factorization of some ETN (solid black circles).

At the end of the first step all factored data are compressed and redistributed across CNs. So, factored data are collected on some CNs, other CNs become free-of-data. At the second step, free-of-data CNs receive next portion of ETNs to factor, other CNs save factored data and send necessary data to other nodes for updating (computing Schur complements). That said, while performing factori-

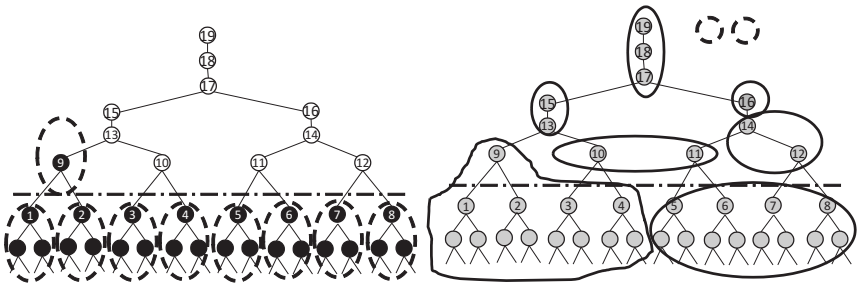

Fig. 4. Left image: data distribution across cluster nodes at the first step of factorization process. Dashed lines denote cluster nodes assigned for processing respective data. Right image: data distribution after the last step. Seven solid lines group data stored on separate cluster nodes. Two dashed line correspond to two cluster nodes that are free of data.

zation process there are two types of CNs: the nodes that keep factors (factored ETNs) in their RAM and send factors by request to the nodes that perform factorization. As the factorization process goes on, the CNs may change specialization - more CNs save factored data and less nodes perform factorization. Finally, all factored data are distributed across the CNs (respective ETNs are surrounded by solid lines), but some CNs can be free-of-data (marked with dashed lines) (Figure 4, right image).

The MS algorithm serves pretty well to save memory but its scalability is far from desired. Firstly, it is caused by exclusion of a part of CNs from the computational process. They are used only to send necessary data to those CNs that do computations, by their requests. Another reason of poor scalability is a structure of the ET – while moving along the ET during factorization the mutual dependence of data become stronger. The idea of PE algorithm is redistribution of jobs among CNs. This redistribution is done so that some jobs that in the basic algorithm was scheduled to be accomplished on the upper level of ET is reassigned to lower level of the ET. In fact, components of the Schur complements, their Low-Rank approximation and some other stuff can be done at lower levels just when respective data become available. For example, parts of the Schur complement of ETN 19 (see Figure 3) become available after completion of processing nodes $1 \ldots 8$. Respectively, other parts of the Schur complement for ETN 19 can be prepared while processing ETNs 9, 10, 11 and 12.

2 Numerical experiments

In this section, we describe a few numerical experiments. The experiments were performed on OS Linux supercomputer Shaheen II (2×Intel Xeon CPU E5-2698 v3 @2.3 GHz per CN, 128 GB RAM) at King Abdullah University of Science and Technology (KAUST). We used one MPI process per CN. The stopping

criterion for the iterative refinement was set as the relative residual is small enough: $\frac{||AU-F||}{F} < 10^{-5}$.

We ran tests on two sizes: $9 \cdot 10^6$ equations and $50 \cdot 10^6$ equations. Experiments show memory advantages of about 30% of the MS algorithm vs. PE one. However, the PE version has better performance and its scalability is ideal up to 16 MPI processes (see Figure 5).

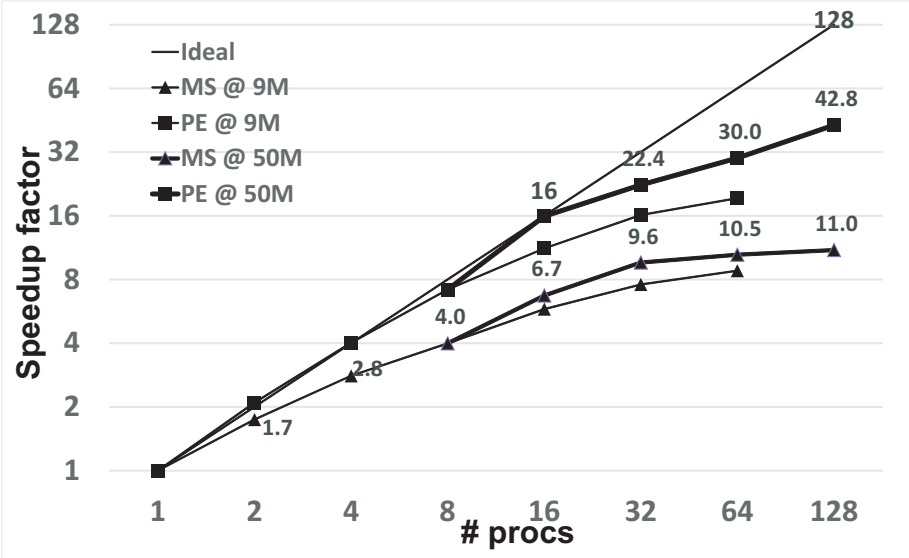

Fig. 5. MPI scalability of MS and PE algorithms measured for sizes 9M and 50M (see the legend in the Figure). Better scalability of the PE algorithm is clearly seen. For convenience, "ideal" scalability is also shown as a straight line.

3 Conclusions

We have presented two parallel algorithms of the multifrontal low-rank sparse solver to solve 3D Helmholtz problem. Proposed algorithms target on HPC systems with distributed memory (clusters) by using hybrid OpenMP and MPI parallelization. The MS algorithm is based on dynamical distribution of the ETNs across CNs. The PE algorithm is based on effective parallel distribution of the jobs across CNs. The numerical experiments prove 30% memory advantage of the first algorithm vs the second one. Respectively, the second algorithms has better performance and ideal scalability up to 16 MPI processes.

Acknowledgments. The research is supported by the RSCF grant 17-17-01128. We appreciate KAUST for providing access to Shaheen II supercomputer.

References

1. Collino F., Tsogka C. Application of the perfectly matched layer absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media Geophysics. 2001. 66. 294–307.
2. George, A., Liu, J. and Ng. E. Computer Solution of Sparse Linear Systems, Florida Academic Press, 1994.
3. Xia J. Robust and efficient multifrontal solver for large discretized pdes. High-Performance Scientific Computing. 2012. 199–217.
4. Tordeux S. Solovyev S. A. An efficient truncated svd of large matrices based on the low-rank approximation for inverse geophysical problems. Siberian Electronic Mathematical Reports, 12:592–609, 2015.
5. Solovyev S. A. Application of the Low-Rank Approximation Technique in the Gauss Elimination Method for Sparse Linear Systems Vychisl. Metody Programm. 15, 441460 (2014), in Russian.