Parallel Substructuring Method With Memory Cost Limits*

Nikita Nedozhogin[1], Sergey Kopysov[1,2], and Alexander Novikov[1]

[1] Institute of Mechanics Udmurt Federal Research Center UB RAS, Izhevsk, Russia
[2] Udmurt State University, Izhevsk, Russia
nedozhogin@inbox.ru

Abstract. In decomposition methods, the memory costs for solving the interface problem are increasing significantly with the number of subdomains increasing. Using of the substructuring method allows you to reduce the number of iterations when the system is being solved. At the same time, the global boundary stiffness matrix takes up more memory in comparison with global stiffness matrix. This imposes restrictions on the maximum size of the problem for which you can apply this method. Different approaches to reduce the costs and limitations of memory on stage of the construction and solving of the interface system exist. A layer-by-layer approach to the partitioning of a triangulated multiply connected domain into connected subdomains without branching of inner boundaries was presented. This makes it possible to avoid conflicts with concurrent operations of the assembly type without using synchronization and critical sections. Parallel algorithm of the construction global boundary stiffness matrix with distributed storage of the matrix are considered when implementing using OpenMP and MPI technologies. This approach allows not only to reduce the limits on the maximum size of the solved problem, but also to resolve conflicts of shared memory access by increasing the number of independent parallel tasks.

Keywords: Parallel algorithms · Decomposition method · Substructuring method · OpenMP · MPI

1 Introduction

The substructuring method is the non-overlapping domain decomposition [1]. However, the global boundary stiffness matrix has more nonzero elements with smaller dimension in comparison with global stiffness matrix [2]. Due to these properties, even the use of sparse matrix storage formats requires considerable memory costs. Block [3] and hierarchical [4] methods with low-rank approximation have received great development due to the current trends in reducing the amount of memory per processor core. In this paper, parallel substructuring method with distributed storage of the matrix is considered. The distributed

* The reported study was partially funded by RFBR according to the research project 17-01-00402-a.

construction makes it possible to apply this approach not only in the classical substructuring method, but also in the hierarchical or block one, or in the case of constructing a preconditioner on the based on the substructuring method.

The second memory limit is the access speed. To accelerate calculations, it is necessary the number of simultaneous accesses to the same memory area both during reading and writing. To increase the parallelism of the method, it is required to arrange the grid and elements in such a way that independent threads simultaneously work with their own sections of memory without overlapping. In this case, We propose using a layer-by-layer partitioning method. In the course of this method, any unstructured mesh can be arranged.

The paper presents the results of parallel version of the substructuring method realized with the help of OpenMP and MPI technology. The potential for parallelization with the help of CUDA technology for computation on graphic accelerators is considered

2 Layer-by-layer Partitioning

We generalize the ordering of the 3D unstructured mesh T on a subdomains without branching of internal boundaries using layer-by-layer partitioning. We define in T the some set of the cells $T^* \subset T$, assuming $L_1 = T^*$. We construct the layers $L_j = \{\sigma \in T \mid L_j \bigcap L_c \neq \varnothing \,\text{when}\, j = c \pm 1\}$, $j \in [2, m-1]$, according to the Algorithm 1. Here m – number of the layers, σ - cell of the mesh T.

Algorithm 1: Partitioning unstructured mesh T on layers.

```
   Input data: $T$ — unstructured mesh of cells $\sigma \in T$, $T^*$ — given set
               $\{\sigma\} \subset T$.
   Result: $\{L_j\}$ — set of the layers of cells.
1  Define the cells corresponding to the vertices of $T$;
2  $\forall \sigma \in T$ find cells having a common vertex;
   /* Form the layers of mesh cells                              */
3  Let $j = 1$ and layer $L_j = T^*$;
   while $\left(\sum_{c=1}^{j} |L_c| < |T|\right)$ do
4      Construct layer $L_{j+1}$ of cells $T$;
5      Let $j \leftarrow j + 1$;
```

The result of the Algorithm 1 is an ordered set of m layers L_j, in which the layers adjacent to L_j have numbers $j-1$ and $j+1$. The resulting layers are discrete analogs of a domain of $\mathbb{R}^3$ of type $(g, 2)\ :\ g \geqslant 0$. All the vertices of the cells lie on the surface bounding the layer. In general, layer L_j consist of a set of sublayers $\{L_j^{(c)}\}$, where each sublayer $L_j^{(c)}$ is the set of cells in L_j having at least one intersection with the cell in L_j. If some layer L_j consists of one sublayer $L_j^{(c)}$, we assume that the layer L_j is connected.

Thus, the problem of forming subdomains of an unstructured mesh $\{T_i : \bigcup_i T_i = T\}$ based on obtained partitioning on layers L_j is as follows:

find the union of the layers L_j in the set of subdomains $\{T_i\}$ satisfying certain restrictions. One of the conditions is the connectivity of the resulting subdomains T_i. The number of layers included in the subdomain is bounded below by two layers, which is a sufficient condition for the existence of vertices in T_i that do not belong to the boundary T_i.

In order to construct connected union of layers in the subdomain of the mesh, it is necessary to find sublayers and their connections. The algorithm 2 provides a search for sublayers and the definition of their connections using a dual graph $G_d(V,E)$ of connected cells in L_j along the faces. Further, the connected components $G_d(V,E)$ are found. The set of the mesh cells corresponding the connected components $G_d(V,E)$ will be denoted as sublayers $L_j^{(c)}, c = 1, 2, \ldots, \kappa_j$, where $\kappa_j = |\{L_j^{(c)}\}|$ — the number of the sublayers in layer L_j, $\kappa^* = \max\limits_j \kappa_j$.

Algorithm 2: The union of the layers $\{L_i\}$ in linked subdomains $\{T_l\}$.

```
   Input data: $\{L_j\}$ — set of the layers of cells.
   Result: $\{T_i\}$ — set of the mesh $T$ subdomains.
   /* Find sublayers in each layer $L_j$ */
1  for $L_j, \forall j \in [1,m]$ do
2      Construct dual graph $G_d(V,E)$ of the links of the cells of $L_j$ along the faces;
3      Define connected components $G_d(V,E)$;
4      Assign sublayers $L_j^{(c)}$ to the connected components $G_d(V,E)$;
5  Construct the graph of sublayers $G(V,E)$;
   /* Form subgraphs $G_l$ of th graph $G(V,E)$: */
6  $i = 1$, $m^* = 1$, $mark(L_j) = 0$, $\forall j \in [1,m]$;
7  while $m^* < m$ do
       /* Find an unmarked layer with $\kappa_j = \kappa^*$ */
8      while $\kappa^* > 1$ do
9          Adjoin adjacent layers to the initial layer of the domain;
10         Define $\kappa^*$ — the number of the connected components $G_i$;
11         $j \leftarrow j + 1$;
12     $i \leftarrow i + 1$;
   if $v \in V_G \wedge deg(v) = 1 \wedge v \Leftrightarrow L_j^{(c)}$ then
       Adjoin $\{v\} \Leftrightarrow L_j$ to $G_j \Leftrightarrow L_{j-1}$, $j \in [1,i]$.
   Partitioning/ordered mesh $T$, assuming $T_i \Leftrightarrow G_i$.
```

The Algorithm 2 extends the partitioning on layers [5] on multi-connected domains. Partitioning without branching on the basis of the neighborhood relation reduces to the use of the Algorithm 1 for the layer-by-layer partitioning of the triangulation T and the union layers $L_j, \forall j \in [1,m]$ by the Algorithm 2 in th subdomains T_i.

3 Substructuring Method

Let the mesh domain Ω be split into n_Ω non-overlapping subdomains $\Omega = \bigcup_{i=1}^{i\leqslant n_\Omega} \Omega_i^{(1)}$, where $\Omega_i^{(1)} \bigcap \Omega_j^{(1)} = \emptyset$, when $i \neq j$. Formally unite the subdomains, preserving the structure of the decomposition (interior nodes remain interior, boundary – boundary). We introduce the decomposition of the second level with number of the subdomains equal number of the computational modules n_p such that $\Omega = \bigcup_{k=1}^{k\leqslant n_p} \Omega_k^{(2)}$, and $\Omega_k^{(2)} = \bigcup_{i=k\frac{n_\Omega}{n_p}}^{i\leqslant(k+1)\frac{n_\Omega}{n_p}} \Omega_i^{(1)}$.

For decomposition the first level, the system of equations is formed in such a way that the unknowns corresponding to the interior and boundary nodes take the form:

$$\begin{pmatrix} {}^1(A_{II}^{(1)}) & 0 & \cdots & {}^1(A_{IB}^{(1)}) \\ 0 & {}^2(A_{II}^{(1)}) & \cdots & {}^2(A_{IB}^{(1)}) \\ \cdots & \cdots & \cdots & \cdots \\ {}^1(A_{BI}^{(1)}) & {}^2(A_{BI}^{(1)}) & \cdots & A_{BB}^{(1)} \end{pmatrix} \begin{pmatrix} {}^1u_I^{(1)} \\ {}^2u_I^{(1)} \\ \cdots \\ u_B \end{pmatrix} = \begin{pmatrix} {}^1f_I^{(1)} \\ {}^2f_I^{(1)} \\ \cdots \\ f_B \end{pmatrix}.$$

In this system, the unknowns stand for the boundary nodes are found from the solution of the system $S^{(1)}u_B = \tilde{f}_B$. Here

$$\tilde{f}_B = \sum_{i=1}^{i\leqslant n_\Omega} (f_B - {}^i(A_{BI}^{(1)})\,{}^i(A_{II}^{(1)})^{(-1)}\,{}^if_I^{(1)})$$

is the right-hand-side vector and $S^{(1)}$ is the global boundary stiffness matrix (also known as the Schur complement matrix [2]) and is the sum of the local boundary stiffness matrices $S_i^{(1)} = A_{BB}^{(1)} - {}^i(A_{BI}^{(1)})\,{}^i(A_{II}^{(1)})^{(-1)}\,{}^i(A_{IB}^{(1)})$ such that $S^{(1)} = \sum_{i=1}^{i\leqslant n_\Omega} S_i^{(i)}$, where index i corresponds to subdomains of the first level.

4 Construct Global Boundary Stiffness Matrix

Global boundary stiffness matrix $S_k^{(2)}$ corresponding to these subdomains are supplemented by zeros up to size global matrix $S^{(1)}$ in this $S^{(1)} = \sum_{k=1}^{n_p} S_k^{(2)}$. Note that, Matrices $S_k^{(2)}$ do not depend on each other during the formation stage. This excludes conflicts when reading elements data and writing results of summing of local matrices $S_i^{(1)}$.

For reduce computational costs of RAM, each computational modules forms one row l of the matrix $S_i^{(1)}$ at the same time. The order of the construction of rows is given by subdomains. Parallel algorithm of the construction for n_p computational modules using OpenMP technology on the example of two subdomains would be as follows (note that $n_p \leqslant n_\Omega$):

Algorithm 3: Parallel algorithm of the global boundary stiffness matrix construction $S^{(1)} = \sum_{k=1}^{n_p} S_k^{(2)}$ for OpenMP

```
  // Forming of $S_1^{(2)}$
  for $i = 1 \dots n_\Omega/n_p$ do
1     Function of the inverse matrix ${}^i(A_{II}^{(1)})^{-1}$ ;
      // Parallelize loop by OpenMP
      for $l = 1 \dots n_{B_i}$ do
2         $\mathbf{v} = {}^iA_{BI}^{(1)}(l,)$ ;
3         $\mathbf{v} = \mathbf{v}\ {}^i(A_{II}^{(1)})^{-1}$ ;
4         $S_i^{(1)}(l,) = {}^iA_{BB}^{(1)}(l,) - \mathbf{v}\ {}^iA_{IB}^{(1)}$ ;
5         Record the $S_i^{(1)}(l,)$ elements in the corresponding row of the matrix $S_1^{(2)}$ ;
  // Forming of $S_2^{(2)}$
  for $i = n_\Omega/n_p \dots n_\Omega$ do
      Function of the inverse matrix ${}^i(A_{II}^{(1)})^{-1}$ ;
      // Parallelize loop by OpenMP
      for $l = 1 \dots n_{B_i}$ do
          $\mathbf{v} = {}^iA_{BI}^{(1)}(l,)$ ;
          $\mathbf{v} = \mathbf{v}\ {}^i(A_{II}^{(1)})^{-1}$ ;
          $S_i^{(1)}(l,\cdot) = {}^iA_{BB}^{(1)}(l,) - \mathbf{v}\ {}^iA_{IB}^{(1)}$ ;
          Record the $S_i^{(1)}(l,\cdot)$ elements in the corresponding row of the matrix $S_2^{(2)}$ ;
```

Based on the computing experiment (See Table 1), run-time costs on each subdomain have the following order (top-down):

1. Inverse of the matrix ${}^i(A_{II}^{(1)})$ (Step 1 of Algorithm 3);
2. Record the row l of the local matrix $S_i^{(1)}(l,\cdot)$ in the corresponding row of the matrix S_1^2 (Step 5 of Algorithm 3);
3. Operations of taking the row from the matrix, the matrix-vectors multiplications and the difference of the vectors (steps 2-4 of Algorithm 3, denote by MatVec).

Taking the row from the matrix, the matrix-vector multiplications and the difference of the vectors (see steps 2-4 of Algorithm 3) are parallelized using OpenMP technology, that for these operations gave an acceleration close to linear. Also, in the future, it is planned to parallelize these operations for use on graphics accelerators using CUDA technology.

Recording of the row l of the local matrix $S_i^{(1)}(l,\cdot)$ to the corresponding row of the matrix $S_k^{(2)}$ (see step 5 of Algorithm 3) largely depends on the matrix storage formats used. Matrices $S_k^{(2)}$ is formed in a compressed CSR format, because of its large size. Using arrays of vectors (C++ container `std::vector`) allows to reduce memory costs to a minimum, but the complexity of adding new elements to such storage formats is $O(n_B)$, where n_B is the number of equations corresponding to the boundary nodes To reduce the costs of adding new elements of the row, we use sorted associative containers (C++ container `std::map`) and not sorted (C++ container `std::unordered_map`). In the sorted container, the complexity of finding and adding a new element is $O(\ln n_B)$ (due to its red-black tree structure), but it significantly increase memory costs to store to store additional two pointers. The non-sorted container, for these operations has complexity form $O(1)$ to $O(n_B)$ (depending on the hash function) and request smaller memory sizes, compared to the container `std::map`, which is optimal for use in the constructing of matrices $S_k^{(2)}$.

Table 1. Characteristic formation times in one subdomain, sec.

Compute ${}^{i}(A_{II})^{(-1)}$		MatVec, see Algorithm 3	Assembly $S^{(1)}$
LU	SM		
$n_\Omega = 512$			
95.4972	41.69505	0.340893	1.53655
$n_\Omega = 1024$			
4.07287	1.947	0.130277	0.6955

To inverse the matrix at Step 1 of Algorithm 1, LU-factorization (LU) and Sherman-Morrison (SM) methods [6] were considered.

For a given non-singular matrix A there exists a decomposition on upper triangular matrix U and a lower triangular matrix L. If matrices A, U, L are invertible, then $A^{-1} = U^{-1}L^{-1}$. In the construction L and U, products with nonzero elements of the parent matrix were summed, since the factorization war carried out for matrices stored in the CSR format. Only sequential implementations of the LU-factorization on CPU was considered. This approach showed the best results for small matrices. Features of this algorithm do not allow to effective use all the capabilities of the massively parallel GPU architecture.

A parallel version of the Sherman-Morrison method implemented using OpenMP technology was considered.

Algorithm 4: Parallel algorithm of Sherman-Morrison method for $(A_{II}^{(1)})^{-1}$

```
1 $\mathbf{x}, \mathbf{y}$ - vectors of dimension $n$ ;
  for $k = 1 \dots n$ do
      // OpenMP
2     $\mathbf{x}^T = A_{II}^{(1)}(k, ) - e_k$;
      // OpenMP
3     $\mathbf{y} = \mathbf{x}^T A_{inv}$;
      // Loop of OpenMP on $i$
      for $i = 1 \dots n$ do
          for $j = 1 \dots n$ do
              $A_{inv}(j,i) = A_{inv}(j,i) - (\mathbf{y}_i * A_{inv}(j,k)/(\mathbf{y}_k + 1))$ ;
4 $(A_{II}^{(1)})^{-1} = A_{inv}$ ;
```

LU-factorization is faster than the Sherman-Morrison method in sequential implementations. In spite of it, LU-factorization does not parallelize well. Using parallel version of the Sherman-Morrison method provided a parallel acceleration about two times. To reduce the number of operations and the computational load, matrix ${}^i(A_{II}^{(1)})^{-1}$ is stored in its entirety before the completion of the constructing of $S_i^{(1)}$ in the current subdomains, since it is not known how many non-zero elements contain an inverse matrix. To solve the system of equations on the inner subdomains nodes ${}^iA_{II}^{(1)}{}^iu_I = {}^if_I - {}^iA_{IB}^{(1)}u_B$ Krylov subspace methods [7] are used.

In the future, parallelization on the GPU is planned. To increase the parallelism, the order of the formation of the local boundary stiffness matrices will be given by layer-by-layer partitioning. Thus, a block of parallel GPU threads will process one row of the matrix $S_i^{(1)}$. In total it will be possible to generate κ_i blocks that are able to write in different places of the same array without overlap and conflicts due to layer-by-layer ordering.

5 Solving the $S^{(1)}u_B = \tilde{f}_B$ System

Implementation of conjugate gradients method for solving the system $S^{(1)}u_B = \tilde{f}_B$ on one of n_p computing node is shown in Algorithm 5. Here the system matrix is represented in the form $S^{(1)} = \sum\limits_{k=1}^{n_p} S_k^{(2)}$. Index k corresponding to the number of computational module is omitted.

Algorithm 5: Algorithm of the preconditioned conjugate gradient method for distributed matrix

```
1  $u, r, p, q, z \in \mathbb{R}^{n_B}$ ;
2  $i = 0$;
3  $r_0 = \tilde{f}_B$ ;
4  $u_0 \leftarrow 0$ ;
5  $z_0 = Mr_0$ ; // $M \sim (S^{(1)})^{-1}$ -- preconditioner
6  $p_0 = z_0$ ;
7  $\rho_0 = (r_0, z_0)$ ; // sync point
   while $||r_i||_2/||f||_2 > \varepsilon$ do
8      Assembly $p_i$ ; // Using MPI::Allreduce
9      $q_i = S^{(2)} p_i$ ; // $k$ independent operations on each MPI
         process
10     $\alpha_i = (r_i, z_i)/(q_i, p_i)$ ; // MPI::Allreduce lead to sync point
11     $u_{i+1} = u_i + \alpha_i p_i$;
12     $r_{i+1} = r_i - \alpha_i q_i$;
13     $z_{i+1} = Mr_{i+1}$ ; // $M \sim (S^{(1)})^{-1}$ -- preconditioner
14     $\rho_{i+1} = (r_{i+1}, z_{i+1})$ ; // MPI::Allreduce lead to sync point
15     $\beta_{i+1} = \rho_{i+1}/\rho_i$;
16     $p_{i+1} = r_{i+1} + \beta_{i+1} p_i$;
17     $i = i + 1$ ;
```

In a matrix-vector product (See steps 8-9 of Algorithm 5) operation $q_i = S^{(1)} p_i$ is performed in two steps. First, the assembly of the vector p_i from all MPI processes, then n_p independent operations $(q_k)_i = S_k^{(2)} p_i$. The result of products is the set of vectors q_k such that $q_k \in \mathbb{R}^{n_B}$ and $q = \sum_{k=1}^{n_p} q_k$. Summed of vector q are not required. Vector is stored by parts on various compute nodes.

Parallelization of the operations with vectors is realized as follows: operands are parts of vectors consisting of components corresponding to mesh nodes of the subdomain $\Omega_k^{(2)}$ for each parallel MPI process. The result vector assembly are not required, each MPI process jobs with its local vectors, and synchronization occurs when the scalar products (r, z), (q, p), (r, r) are computed. Scalar products are performed in two steps. First, local scalar products for the part vectors in parallel MPI processes, then summation of the local sums (on this step, implicit synchronization occurs). Inside the MPI process, vectors operations are parallelized using OpenMP technology with number of the thread equal number of the CPU cores on computational nodes.

To parallelize the GPU, it is planned to use an approach similar to that presented in paper [2].

6 Results

Numerical experiments were carried out on the test problem of the theory of elasticity. The computational domain had the form of a parallelepiped. Mesh consisted of 539000 hexagonal cells. 8 threads of the OpenMP are running inside each MPI process. In Table 2, the result of experiment are presented for dimension on 512 and 1024 subdomains with using 2, 4, 6 MPI processes. The presented results are obtained on 6 computational nodes, each of which contains two processors Intel Xeon E5-2609 and 64GB RAM. For $n_\Omega = 512$, this problem took approximately 100GB of RAM, for $n_\Omega = 1024$, problem took approximately 80GB of RAM.

Table 2. Time of construction and solve in the Substructuring method

n_p	$\sum S_i^{(1)}$		$S^{(1)} u_B = \tilde{f}_B$		${}^iA_{II}^{(1)}\,{}^iu_I = {}^if_I - {}^iA_{IB}^{(1)} u_B$	
	LU, sec.	SM, sec.	#	Time, sec.	#	Time, sec.
$n_\Omega = 512$						
2	34954.6	14936.8	2324	1640.37	43	4.72359
4	18086.6	7615.15	2325	1391.23	43	2.60635
6	11713.4	5014.66	2325	1498.37	43	1.9185
$n_\Omega = 1024$						
2	5128.47	2818.16	2600	1464.09	35	3.15815
4	2622.52	1411.97	2600	1397.77	35	2.10205
6	1802.32	965.803	2600	1469.18	35	1.74885

The considered algorithm allows to significantly reduce the memory limits for the substructuring method. The partitioning of the matrix $S^{(1)}$ and its distributed storage makes it possible to realize coarse-grained parallelism at the formation stage of global boundary stiffness matrix, to reduce exchanges and to exclude synchronous access to one memory cell during the local matrices addition. At the same time, an increase in the number of involved computational modules permits achieving an acceleration close to linear at the stage of matrix construction. At the solver stage, the distributed storage of the matrix $S^{(1)}$ allows solving systems with a dimension proportional to the memory size of the involved computational modules. Scalability is limited to assembling the results of scalar products of vectors and depends only on the restrictions on the speed to data exchange over the network.

References

1. Toselli A., Widlund O.B.: Domain Decomposition Methods — Algorithms and Theory. Springer Series in Computational Mathematics, Vol. 34 (2005)
2. Kopysov S.P., Kuzmin I.M., Nedozhogin N.S., Novikov A.K., Sagdeeva Y. A.: Hybrid Multi-GPU solver based on Schur complement method. Lecture Notes in Computer Science, vol. 7979, pp. 65-79 (2013)

3. Amestoy P.R., Buttari A., L'Excelent J-Y., Mary T.: On the Complexity of the Block Low-Rank Multifrontal Factorization. SIAM Journal on Scientific Computing, vol. 39(4), pp. A1710-A1740 (2017)
4. Chen C., Pouransari H., Rajamanickam S., Boman E.G., Darve E.: A distributed-memory hierarchical solver for general sparse linear systems. Parallel Computing, vol. 74, pp. 49-65 (2018)
5. Novikov A.K., Piminova N.K., Kopysov S.P., Sagdeeva Y.A.: Layer-by-layer ordering in parallel finite composition on shared-memory multiprocessors. IOP Conf. Ser.: Mater. Sci. Eng., vol. 158(1), p.158 (2016)
6. Sherman J., Morrison W.J.: Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. Ann. Math. Statist., vol. 21(1), pp. 124-127 (1950)
7. Saad Y.: Iterative Methods for Sparse Linear System, Boston: PWS Publiching company, p. 447 (1996)