

Workflow for Parallel Processing of Sequential Mesh Databases

Ondřej Meca

VŠB - Technical University of Ostrava
IT4Innovations - National
Supercomputing Center
Ostrava, Czech Republic
ondrej.meca@vsb.cz

Lubomír Říha

VŠB - Technical University of Ostrava
IT4Innovations - National
Supercomputing Center
Ostrava, Czech Republic
lubomir.riha@vsb.cz

Tomáš Brzobohatý

VŠB - Technical University of Ostrava
IT4Innovations - National
Supercomputing Center
Ostrava, Czech Republic
tomas.brzobohaty@vsb.cz

ABSTRACT

This paper presents a workflow for parallel loading of sequentially stored mesh databases. It can be used as a connection between tools for the creation of complex engineering models along with parallel solvers to allow broader usage of HPC by the engineering community. Scalability tests show that the proposed algorithm is able to prepare a mesh with hundreds of millions of nodes/elements in several seconds.

KEYWORDS

parallel mesh loader, parallel mesh decomposition, format converter

1 INTRODUCTION

A wide range of engineers from industrial practice use numerical methods to simulate physical processes. The simulation process can be divided into two parts – *creation* of a numerical model (finite element mesh, definition of boundary conditions, contact interfaces, etc.) and its *solution*. Since the solution quality is dependent on the numerical model, it is necessary to use an efficient and robust tool for its creation. Typical examples are problems in structural mechanics, where the creation of a high-quality numerical model is one of the most complicated (and important) stage during the simulation process (especially for complex geometries).

In general, high-quality models are usually provided by (commercial) tools that produce sequential mesh databases. Hence, some parallel open source tools such as Elmer, Calculix, and OpenFOAM provide a converter of external (sequential) database files into their (parallel) database structure. For example, in OpenFOAM, the user has to convert the external database to the OpenFOAM format sequentially, and then sequentially create an initial decomposition that can be redistributed in parallel. This approach greatly slows down or inhibits the connection between tools that generate sequential data and parallel solvers. It is one of the factors contributing to the low utilization of HPC by the mainstream engineering community.

Our motivation is to create a library connecting tools for the creation of complex engineering models (such as ANSYS, HyperMesh, ANSA, ABAQUS, etc.) with open source parallel solvers, enabling the broader usage of HPC by the engineering community. The result of this direct connection is robust preprocessing together with possibility to connect various highly parallel solvers that are able to solve non-standard problems.

We propose an algorithm that is able to load and decompose a sequentially stored mesh database. Thus, it can be used as a parallel converter for various formats. Moreover, scalability tests show that the algorithm can also be used as a direct loader and preprocessor

of massively parallel solvers. The algorithm is composed of several commonly used approaches that together lead to a robust and fast solution. This paper provides a rough outline of these steps.

2 THE WORKFLOW

The workflow is composed of several steps. At the beginning a mesh database is read and parsed. Since the parsed data can be randomly scattered among MPI processes, we apply parallel sorting to get a known data distribution. Then we are ready to assign regions and make a spatial clusterization. This step assures the scalability of the next stage of the processing, which prepares a mesh for a parallel solver. The following subsections describe the steps in more detail.

2.1 A Mesh Database File

The poster presents reading of mesh database files from ANSYS. However, we are not restricted to this format only. We suppose that a mesh is composed of nodes (points in 2D/3D), elements (composed from nodes), and regions (of nodes, boundaries, or elements). Both nodes and elements are distinguished through a unique ID. These IDs are used for definition of regions. A mesh database can also contain a description of materials or boundary conditions. Since these data can be easily reproduced to all processes, we omit them in our description.

2.2 Loading and Understanding Data

In the case of ANSYS, a mesh description is in a single file, based on a command structure. We copy it to memory via a collective MPI file reader in such a way that each process reads approximately the same number of bytes. Then, data are scanned for commands that define boundaries of nodes and elements blocks. This scanning is necessary, since the input data do not contain any kind of overall description. Synchronization of blocks' boundaries across processes allows the parsing of data in parallel.

2.3 Parallel Sorting

Parsed data are randomly scattered among processes that do not share any information. Therefore, we have to link everything together to be able to address elements, nodes, and assign regions. To do this we balance both nodes and elements among processes and sort them according their IDs. This allows the description of the distribution of both sets via a vector of boundaries. Based on this vector, we can send regions data to a process that holds a given node or element.

2.4 Spatial Locality

The scalability of parallel mesh processing is dependent on the uniform distribution of elements and the number of neighbours of each process. The former is assured by the previous sorting. Unfortunately, IDs are usually randomly distributed across the mesh (see the poster) which leads to a situation where all processes are adjacent to each other. To minimize the number of neighbours, elements must be spatially clustered.

The spatial clusterization is based on the Hilbert space filling curve (SFC). It provides a scalable solution, since SFC is only the function that maps 2D/3D points to SFC indices. If two indices are close, the SFC mapping ensures that origin points are close also.

Since SFC works with coordinates we firstly map nodes. Then, each element asks for the SFC index of its node, which is stored on the closest process (we again take advantage of nodes being sorted). SFC indices are used for the second parallel sort. Applying the second sorting according to SFC indices, processes obtain spatially located nodes and elements. According to the distribution of SFC indices, we compute processes' neighbours and the node duplicity (information about all processes where a given node occurs).

2.5 Improving the Quality

The main priority of the previous step is not the decomposition quality. The spatial locality only assures reasonable times of the next processing that includes a dual graph based decomposition (we use the state-of-the-art tool ParMETIS). This type of decomposition is a better fit to FETI based solvers' (implemented in our in-house library ESPRESO [5]) requirements than geometric based approaches.

3 METHODS USED IN PARTICULAR STEPS

The skeleton of the above described workflow is based on algorithms that provide an efficient parallel sorting. In our approach we use the histogram parallel sort [1]. The spatial locality is achieved through the use of the Hilbert space filling curve [3]. During the parallel sorting, data are exchanged according to the recursive halving algorithm [4]. We use METIS and ParMETIS [2] for the final decomposition of the mesh before a parallel solver is called.

4 MEASUREMENTS

The algorithm was implemented in the ESPRESO library as a parallel mesh loader. The poster presents its performance on the Salomon supercomputer at IT4Innovations National Supercomputing Center. Salomon consists of 1008 compute nodes (2 x Intel Xeon E5-2680v3, 2.5 GHz, 12 cores; 128GB DDR4@2133 MHz) interconnected by InfiniBand FDR56 / 7D hypercube. Since measurements present strong scalability we use only MPI parallelization. The number of MPI processes used for reading is enough for the ESPRESO library, which is able to utilize hybrid parallelization (MPI/OpenMP).

Graphs are composed from parts that correspond with the above described steps. The first (green) part shows times of reading and parsing a file. The highlighted part shows the increasing time of MPI_File_open. It introduces an overhead for the large number of MPI processes (the time is not dependent on the size of a read file). We include this function to measurements since we present the total time needed to prepare a mesh for a solver. On the other hand, MPI I/O is out of the scope of the poster. Hence, we have

not implemented any kind of optimizations that can improve the performance.

The second (orange) part shows times of the parallel sorting according to IDs, assigning regions, and clusterization of a mesh via SFC. The last (purple) part shows times of improving the quality of the decomposition. It includes computation of the dual graph and calling ParMETIS (its computation time is also highlighted). The last part also contains the second level decomposition (using METIS), a synchronization of regions, rearranging nodes and elements, etc. These steps are required for correct settings of our parallel solver. These requirements are similar to other parallel solvers.

To emphasize the importance of the clusterization, the graphs also contain the time of the ParMETIS decomposition when elements are not spatially located. The effect of the spatial locality can be observed in comparison between the *manifold* and *disc brake* examples. Even the time for decomposition of spatially located elements is similar, the disk brake is decomposed much faster when the clusterization is not applied. The reason is division of the disk brake into blocks that assure a partial spatial locality (similarly to the teeth in the drill bit example in the poster).

5 CONCLUSION

The poster presents a workflow that can be used for parallel loading of sequential mesh databases. It has been shown that despite the sequential nature of input data, they can be effectively parsed in parallel. The workflow was implemented in the ESPRESO library as a proof of concept. In the future, we will provide a library with this algorithm for other researchers.

ACKNOWLEDGEMENT

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project "IT4Innovations excellence in science – LQ1602" and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070". This work is partially supported by project EXPER-TISE - models, EXperiments and high PERformance computing for Turbine mechanical integrity and Structural dynamics in Europe, <http://www.msca-expertise.eu>. The work was also partially supported by the SGC grant No. SP2018/159 "Hardware acceleration of matrix assembler and GUI development of the ESPRESO library", VŠB - Technical University of Ostrava, Czech Republic.

REFERENCES

- [1] L. V. Kale and S. Krishnan. 1993. A Comparison Based Parallel Sorting Algorithm. In *Parallel Processing, 1993. ICPP 1993. International Conference on*, Vol. 3. 196–200. <https://doi.org/10.1109/ICPP.1993.17>
- [2] G. Karypis and V. Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20, 1 (1998), 359–392. <https://doi.org/10.1137/S1064827595287997>
- [3] H. Sagan. 1994. *Hilbert's Space-Filling Curve*. Springer New York, New York, NY, 9–30. https://doi.org/10.1007/978-1-4612-0871-6_2
- [4] R. Thakur, R. Rabenseifner, and W. Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (Feb. 2005), 49–66. <https://doi.org/10.1177/1094342005051521>
- [5] L. Riha, T. Brzobohatý, A. Markopoulos, O. Meca, and T. Kozubek. 2016. Massively Parallel Hybrid Total FETI (HTFETI) Solver. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '16)*. ACM, New York, NY, USA, Article 7, 11 pages. <https://doi.org/10.1145/2929908.2929909>