Distributed High-Performance Parallel Mesh Generation with ViennaMesh

Jorge Rodríguez[1], Josef Weinbub[1], Dieter Pahr[2],
Karl Rupp[1,3], and Siegfried Selberherr[1]

[1] Institute for Microelectronics, TU Wien
[2] Institute for Lightweight Design and Structural Biomechanics, TU Wien
[3] Institute for Analysis and Scientific Computing, TU Wien
Vienna, Austria

Abstract. The ever-growing demand for higher accuracy in scientific simulations based on the discretization of equations given on physical domains is typically coupled with an increase in the number of mesh elements. Conventional mesh generation tools struggle to keep up with the increased workload, as they do not scale with the availability of, for example, multi-core CPUs. We present a parallel mesh generation approach for multi-core and distributed computing environments based on our generic meshing library ViennaMesh and on the Advancing Front mesh generation algorithm. Our approach is discussed in detail and performance results are shown.

1 Introduction

A mesh as partitioning of a physical domain is required to model continuous phenomena by means of discretized equations in the discrete domain of a computer [6]. The continually growing demand for increased accuracy and the ability to simulate on more and more complex geometries introduces the need to increase the number of the mesh elements. Today meshes with around 10^9 vertices are utilized in large-scale scientific computations [9]. Increasing the mesh size, however, intensifies the role of mesh generation, as conventional non-parallel mesh generation tools simply take too long to compute the partitioning [4]. As a remedy to this problem, a domain decomposition technique can, for example, be utilized, where basically the initial domain is partitioned, distributed, locally (re-)meshed, and utilized [5]. Merging the mesh generation step with the simulation on distributed computing nodes significantly increases the overall efficiency, as the communication overhead is minimized. In this work we investigate a self-consistent volume mesh generation approach for multi-core CPUs and distributed computing environments based on the Message Passing Interface (MPI). We show that our concise approach achieves a considerable scaling behavior, thus we may conclude that our approach is a means to significantly accelerate the generation of large volume meshes.
This work is organized as follows: Section 2 puts the work into context. Section 3 introduces our approach, and Section 4 validates the work by depicting performance results.

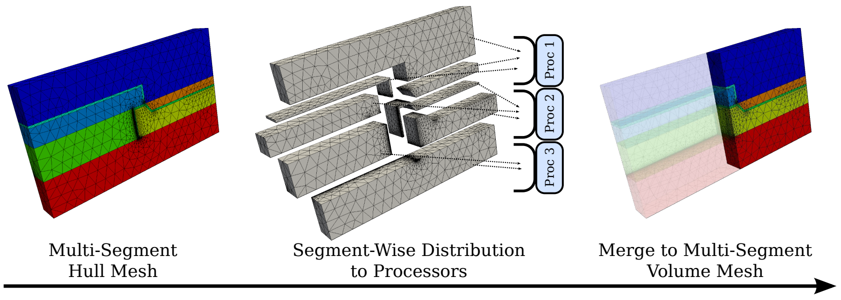
Proc 1
Proc 2
Proc 3
Multi-Segment Hull Mesh
Segment-Wise Distribution to Processors
Merge to Multi-Segment Volume Mesh

Fig. 1. The input multi-segment hull mesh (**left**) is distributed segment-wise to the individual processes (**middle**) and finally the multi-segment volume mesh is merged from the partial segment volume meshes (**right**)

2 Related Work

Different methods for parallel mesh generation are available, which offer various advantages and disadvantages [3]. For example, Delaunay based methods can be used, where basically the input domain is initially just roughly meshed and then gradually refined. The refinement process is parallelized which is quite challenging due to required synchronization steps between the individual point insertions. Another example would be Advancing Front based methods which start the volume meshing from an initial surface and gradually attach new elements to this surface. Hence, it can informally be seen as growing the volume mesh from an initial surface towards the interior. Different parallelization approaches are used, for instance, so-called partially coupled methods, where parallelizable regions in the various mesh sub-domains are identified prior to the mesh generation process. A self-consistent parallel volume meshing approach based on a shared-memory model has already been investigated previously [7]. This approach utilizes the Advancing Front technique for partial volume meshing steps, but obviously can not scale beyond a multi-core CPU.

3 Our Approach

Our approach is based on the Advancing Front meshing technique, in which the algorithm preserves the input hull mesh during the volume meshing process. Therefore, the communication overhead is minimized, as interface changes do not have to be communicated through the parallelized meshing environment. We utilize the ViennaMesh library which offers a unified interface to various mesh related tools [8]. We use the generic interface of ViennaMesh to utilize the Netgen volume mesh generation tool [1]. The input mesh is expected to be partitioned, which in our case is a reasonable assumption due to the availability of CAD tools, for example, the Synopsys Structure Editor can be utilized in the field of semiconductor device simulation [2]. We refer to each partition as a segment of the mesh, thus we call the whole mesh a multi-segment mesh. Furthermore, the individual segments of the input multi-segment mesh are hull meshes, meaning that each segment contains the surface of the sub-domain which has to be meshed. Figure 1 depicts the schematic principle of our approach. The individual hull segments are transmitted to the processes, where they are meshed. The root process is used for driving the overall parallelized mesh generation, whereas the other available processes are solely used for the partial volume meshing tasks. The partial volume mesh results are then sent back to the root node, where they are merged in a final step to the resulting multi-segment volume mesh.

4 Performance

In the following we present the performance of our approach. Our test environment consists of three workstations, namely two AMD Phenom II X4 965 with 8 GB of memory, and one INTEL i7 960 with 12 GB of memory, connected by a gigabit Ethernet network. We investigate two different types of meshes. First, two artificial test hull meshes containing 96 segments with ∼150k and ∼590k vertices, respectively (Figure 2). The number of vertices per segment is constant, allowing to investigate the optimal case, where each segment represents a constant workload for a process. Second, a hull mesh from the field of semiconductor device simulation is investigated, containing ∼110k vertices, 8 segments, and a varying number of vertices per segment (Figure 3). This mesh is used to outline the decrease in efficiency for small numbers of segments. The results depict that the meshing step, which includes the volume mesh generation on the nodes and the related MPI communications, scales reasonably well for meshes with approximately a ten times larger number of segments than the number of cores. Figure 2 depicts an efficiency of about 80% for 10 cores and different mesh

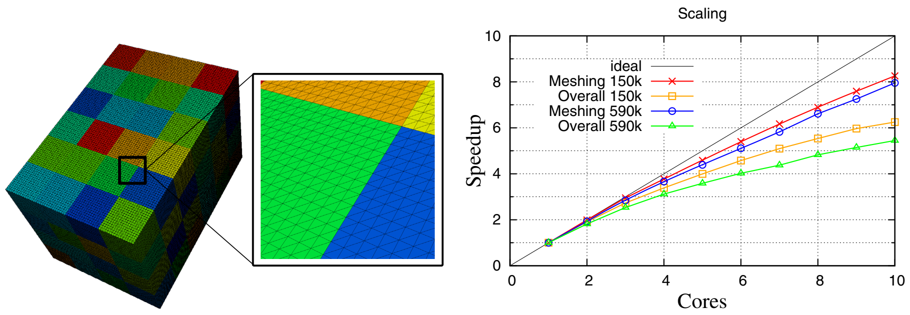

Fig. 2. Left: An artificial test mesh is analyzed in two different ways. The mesh offers 96 segments, 110k, and 590k vertices, and an equal number of vertices per segment. The colors indicate different segments. **Right:** The meshing step offers reasonable scalability for both mesh sizes (80% efficiency for 10 cores). However, the speedup decreases due to the overhead of mesh merging.

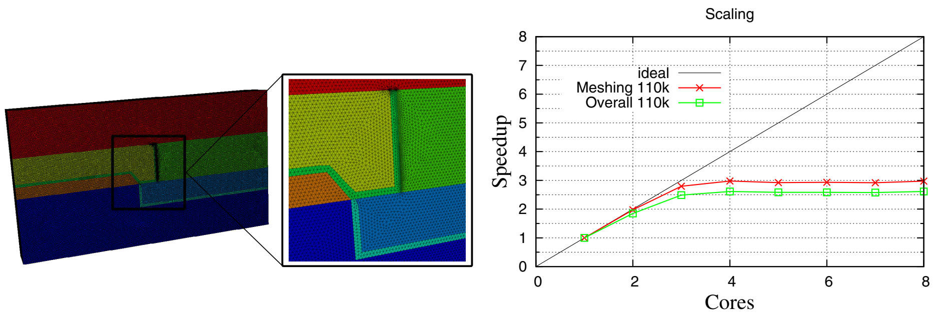

Fig. 3. Left: A mesh from the field of semiconductor device simulation is analyzed. The mesh offers 8 segments, 110k vertices, and a varying number of vertices per segment. The colors indicate different segments. **Right:** Excellent scaling can be achieved for up to 3 cores. However, due to the small number of segments and differently sized segments the scaling saturates at a speedup of 3.

sizes. However, the efficiency is reduced for larger core numbers, as the final step of merging the partial mesh results on the root node becomes large relative to the overall execution time. For example, for the 150k mesh and 2 cores, 7.7% of the overall execution time is used for the final mesh merging, where with 10 cores it is already 25%. Figure 3 outlines that our approach achieves a considerable speedup of 3 for meshes offering a small number of 8 segments. However, the scaling saturates for 4 cores and a speedup of 3, due to the small number of segments relative to the number of cores and due to the varying number of vertices in the different segments.

5 Conclusion

Our approach offers good scalability for meshes with approximately ten times larger number of segments than the number of cores. Even for meshes with approximately three times the number of segments than the number of cores, we achieve a considerable speedup. Therefore we conclude, that our presented self-consistent parallel mesh generation approach is indeed a meaningful way to significantly accelerate the volume mesh generation. However, a flexible mesh partitioning approach is required to enable an improved speedup for larger distributed environments. The mesh merging step in the root process has to be further improved, to achieve higher efficiency for large-scale meshes.

Acknowledgments. This work has been supported by the Google Summer of Code 2011 and by the European Research Council through the grant #247056 MOSILSPIN. Karl Rupp gratefully acknowledges support by the Graduate School PDETech at the TU Wien.

References

1. Netgen, `http://sourceforge.net/projects/netgen-mesher/`
2. Synopsys, `http://www.synopsys.com/`
3. Chrisochoides, N., et al.: A Survey of Parallel Mesh Generation Methods. Brown University (2005)
4. Ito, Y., et al.: Parallel Unstructured Mesh Generation by an Advancing Front Method. Mathematics and Computers in Simulation 75(5-6) (2007)
5. Magoules, F.: Mesh Partitioning Techniques and Domain Decomposition Methods. Saxe-Coburg Publications (2008)
6. Shewchuk, J.R.: Unstructured Mesh Generation. In: Combinatorial Scientific Computing. pp. 259–297. CRC Press (2012)
7. Stimpfl, F., et al.: High Performance Parallel Mesh Generation and Adaptation. In: Proc. PARA (2008)
8. Weinbub, J., et al.: High-Quality Mesh Generation Based on Orthogonal Software Modules. In: Proc. SISPAD (2011)
9. Zhou, M., et al.: Tools to Support Mesh Adaptation on Massively Parallel Computers. Engineering with Computers (2011)