Parallel mesh generation

Lutz Lämmer and Michael Burghardt*
Department of Mechanical and Chemical Engineering,
Heriot-Watt University, Edinburgh, UK

June 10, 1998

Abstract

The efficient parallelisation of the finite element method is based on geometric partitioning of the computational domain into an appropriate number of sub-domains. The problem size required for efficient application of parallel solution techniques is usually large. The problem description in terms of a large number of finite element nodes and elements is complicated and difficult to handle with respect to the required amount of memory and file size.
We describe a parallel solution method to perform mesh partitioning and mesh generation completely in parallel without the preceding serial mesh generation process. The approach avoids this serial bottleneck. The finite element data are generated exactly at the memory location where they are processed in the subsequent analysis process.
The geometric description of the computational domain consists of vertices, edges, and faces, boundary conditions, loads, and mesh density parameters. The geometric description is recursively partitioned, whereby the domain interfaces are minimized with respect to the number of interface nodes. Load balance is guaranteed for graded and locally refined meshes by an especially adapted estimation procedure. Applications for two-dimensional models in structural mechanics are demonstrated.

1 Introduction

Mesh generation is a well-known hard problem in the finite element method. The finite element method allows modeling of physical phenomena on unstructured, graded meshes in contrast to finite difference method using structured or block-structured computational meshes. The usability and flexibility of the finite element method is proved in a vast amount of applications. Usually, the input data for the unstructured meshes result from CAD data.
The need to generate finite element meshes quickly is a common requirement in most computational fields. This is especially true for adaptive finite element processes. Therefore, the need for parallel mesh generation is well justified. Parallel solution techniques are mainly based on the partitioning of the computational data and mapping them onto a particular parallel machine. For distributed memory parallel architectures geometrically motivated partitioning is very efficient to construct fast solvers. Additionally, parallel mesh generation avoids the bottleneck of serial mesh generation arising from storing, partitioning and loading large computational meshes.
The partitioning problem is known to be NP-complete (proof see [23]). That means, the problem cannot be solved in polynomial time. Parallelising the partitioning approach reduces significantly the time for partitioning a given mesh. The mapping problem is known to be NP-complete, too. Although, useful heuristic approaches are available (see [5, 10]).
A large number of research groups works in the field of parallel mesh generation [33, 24, 13, 25, 15, 32, 35]. The usual attempt to generate and partition a given problem in parallel is made in two steps using a so-called background mesh.

Step 1 Solve the partitioning problem for the background mesh.

Step 2 Refine the background mesh to achieve the required mesh density.

Step 1 is usually performed in a recursive manner resulting in a number of idle processors during the first recursion levels. Step 2 is inherently parallel.
The motivation of the development of these algorithms is twofold:

1. The size of the partitioning problem for the background mesh is significantly reduced compared to the original problem size resulting in a significant shorter runtime.

2. The mesh refinement process is parallelized using the (geometric) locality.

*Institut für Numerische Methoden und Informatik im Bauwesen, Technische Universität Darmstadt

The generated partitioning should ensure connected and well-shaped sub-domains on every processor and equally distributed numbers of elements or nodes for the non-overlapping domain decomposition approach.
The mapping problem is usually not solved by the approaches in the techniques referenced. But the parallel generation of the mesh is motivated by the elimination of the mesh loading and distribution process. Re-assigning the mesh to a more appropriate mapping is contra-productive. Therefore, the generated partitioning should also ensure a near optimal mapping.
The strategies based on background meshes are dual to the multilevel methods developed mainly for the high quality partitioning algorithms based on the spectral bisection method. Origins of this method can be found in the works of Barnes and Hoffman [3], and it was first proposed by Pothen [27] for improving graph ordering for sparse matrix factorisation. The recursive partitioning is well suited for a near optimal mapping of the partitions to hypercube and mesh topologies.
Although spectral bisection produces best quality decompositions (compare [30, 34, 22]), the complexity of the Eigenvalue problem to be solved is significant. Therefore, multilevel methods were developed reducing the Eigenvalue problem by combining elements of the computational mesh to larger elements. The partitioning is performed for the smaller problem with the larger elements (see [2, 14, 17]). Jones and Plassman [16], Barnard and Simon [1], Roose and Driessche [8, 7, 9], and Karypis and Kumar [18] are proposing parallel versions of the algorithm to reduce runtime.
Nevertheless, the quality of the final partitioning of the refined mesh has to be improved by local techniques to minimize cut length and optimize the sub-domain shape. Techniques applicable both for serial and parallel improvements are Kernighan-Lin heuristic [19] or Helpful-set strategy [6].
In contrast to the background mesh approach we propose a method for parallel mesh generation working directly on the CAD input data and without a predefined background mesh. Sub-domain boundaries are introduced as additional geometric constraints to the original data. The quality of the partitioning is directly ensured by straight cut lines. The resulting sub-domains are well shaped. We believe that a final improvement by a local heuristic involving communication is not necessary. The generated partitions are nearly optimal mapped to the underlying hardware topology. The parallel mesh generation algorithm is described in section 2. Results obtained for unstructured uniform and graded meshes are discussed in Section 3 and Section 4. Two mesh generation schemes are compared, namely the advancing front mesh generation method [26] and Delaunay triangulation based mesh generation method [29] based on Ruppert's Delaunay triangulation [28] and Lawson's algorithm for point insertion [21], respectively.

2 Parallel Mesh Generation

In the following subsection we describe the input data for the proposed meshing algorithm. Next, we describe briefly the inertial method applied to the partitioning problem. Furthermore, we explain more detailed aspects of the sub-domain boundary generation.

2.1 Input data

The two-dimensional geometry is described by linear primitives. These data origin from a CAD description and consist of points, lines and polygons describing the outline and holes of a two-dimensional computational domain, properties assigned to sub-domains, polygons (or boundaries of sub-domains), lines (segments of polygons) and single points (see Figure 1).
Information on the expected finite element size are assigned to particular points. The expected element size is interpolated between these points. Lines are defined as linear connections of previously defined points. Polygons consist of an ordered set of lines. Polygons are assumed to be closed. One of the polygons describes the outline of the two-dimensional region. Holes are described by special tags to polygons. Additional lines and points describe lines or points which will be part of the computational model.

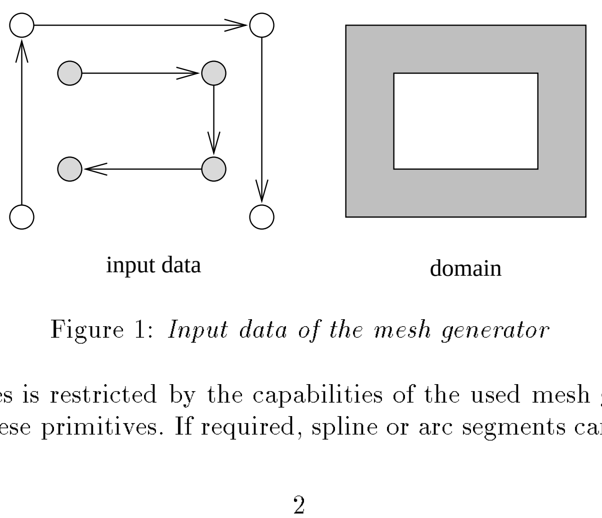

Figure 1: *Input data of the mesh generator*

The choice of geometric primitives is restricted by the capabilities of the used mesh generation procedure. Obviously, our method is not restricted to these primitives. If required, spline or arc segments can be easily added to the proposed

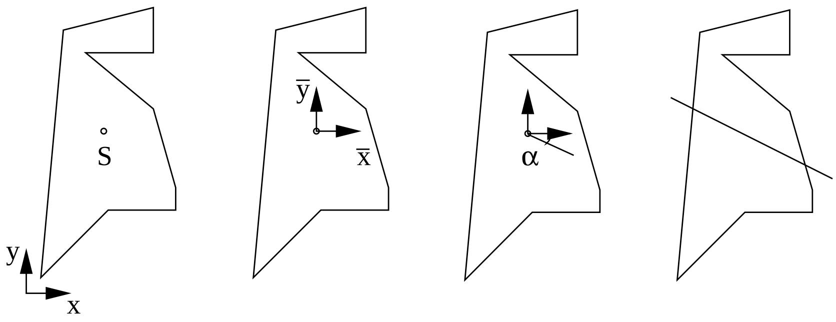

Figure 2: *Recursive bipartition by the inertial method*

mesh generation scheme.

2.2 Original inertial method

The main idea for the partitioning process itself origins from the inertial method. See for example [11] for a detailed description of the method directly applied to finite element meshes.
The original algorithm is recursively formulated. It treats either the nodes or the elements of the finite element mesh as point masses. The point mass of an entity i results directly from its weight w_i and is assigned to its co-ordinates (x_i, y_i, z_i). The inertial moments of the mesh can be calculated by the Steiner principle with respect to the center of gravity (x_s, y_s, z_s) by:

$$\begin{aligned}
I_{xx} &= \sum_i w_i((y_i - y_s)^2 + (z_i - z_s)^2) \\
I_{yy} &= \sum_i w_i((x_i - x_s)^2 + (z_i - z_s)^2) \\
I_{zz} &= \sum_i w_i((x_i - x_s)^2 + (y_i - y_s)^2) \\
I_{xy} = I_{yx} &= \sum_i w_i(x_i - x_s)(y_i - y_s)^2 \\
I_{xz} = I_{zx} &= \sum_i w_i(x_i - x_s)(z_i - z_s)^2 \\
I_{yz} = I_{zy} &= \sum_i w_i(y_i - y_s)(z_i - z_s)^2 \qquad (1)
\end{aligned}$$

The domain is split into two equally weighted sub-domains along the axis of the larger inertial momentum. The quality of the cut is problem dependent. For a detailed discussion see [11].

2.3 Applied inertial method

Our method is directly based on the partitioning of the input CAD geometry. We apply the inertial method adding interface lines to the model. These interface lines cut the original geometry without changing the nature of the meshing problem. Therefore, the same mesh generator as in the serial approach may be applied.
The inertial partitioning is performed in the following steps by all parallel working processes starting with the complete input CAD geometry:

1. Calculate the centre of gravity and the principal axises of the geometry.
2. Split the geometry along the axis of the larger inertial momentum.
3. Choose a sub-geometry (the other part will be handled by another process).
4. Update sub-domain connectivity information.
5. Continue with Step 1.

Handling mesh density functions The mesh density is described by a pointwise defined density function $f(x,y)$. Usually, such a function defines the requested element size at particular points or the number of elements surrounding these points. To ensure equally distributed computational load in the mesh generation and, more important, in the following finite element computation the number of elements should be the same on every processor. Therefore, we construct from the function describing the element size a function describing the computational load. In the linear finite element analysis the computational load is proportional to the number of elements. More sophisticated approaches would be required for a non-linear finite element analysis.
We are using the function $f(x,y) = \frac{1}{w^2}$ measuring the element maximum diagonal length or the surrounding circle, respectively to describe the mesh size w. Alternatively, we are using the function $f(x,y) = (\bar{w} - w)^2$ indicating an average or maximum element side length $\bar{w}$. Both functions are well suited for a weighting function applied to the domains describing the problem to be used in the inertial method.
Step 1 of the bipartition is performed as follows (see Figure 2). The geometry is triangulated by the Delaunay triangulation using the existing point set. Therefore, the existing point set has to describe the domain in a user acceptable way. The center of gravity and the mass is calculated for every volume element described by the triangle and the mesh density function. Therefore, the user has to ensure the modeling of the mesh density function is appropriate with the given set of points (see also 3.2). The mesh density function is interpolated for all geometry points defining the triangulation. For every triangle numerical integration is applied by

$$x_s = \frac{\int x \cdot f(x,y)dA}{\int f(x,y)dA}, \quad y_s = \frac{\int y \cdot f(x,y)dA}{\int f(x,y)dA} \tag{2}$$

to calculate the centre of gravity $\bar{x}, \bar{y}$. The inertial momentums are computed by

$$\begin{aligned} I_{\bar{x}} &= \int \bar{y}^2 \cdot f(\bar{x},\bar{y})dA, \quad I_{\bar{y}} = \int \bar{x}^2 \cdot f(\bar{x},\bar{y})dA \\ I_{\bar{x}\bar{y}} &= -\int \bar{x} \cdot \bar{y} \cdot f(\bar{x},\bar{y})dA. \end{aligned} \tag{3}$$

The coupling nodes on the sub-domain boundaries have to be generated deterministically to avoid additional communication. The problem is twofold. First, the co-ordinates of the nodes generated on the boundary between two or more sub-domains have to be the same to ensure conforming meshes. Second, the global name-space of the coupling nodes has to be unique.

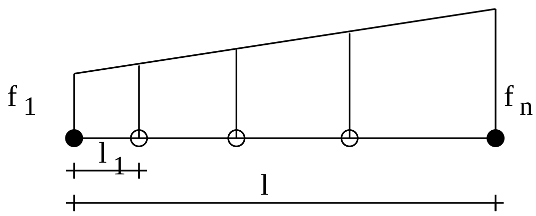

Figure 3: *Discretisation of a line segment*

Conform meshing of interfaces Based on the input data we are using a technique defining piecewise node placements on linear segments. A segment is defined by two nodes with prescribed mesh size values f_1 and f_n and a connecting straight line. The node with the smaller mesh size value f_1 is assumed to be the starting point of the recursive node generation process, which is illustrated in Figure 3.
The distances of the $n-1$ intermediate nodes on the segment are computed by the recursion

$$\begin{aligned} l_1 &= \beta f_1 \\ l_i &= l_{i-1} + \beta \sum_{j=1}^{i} j \end{aligned} \tag{4}$$

whereby β is computed in a preprocessing step.
Following [20] the length of the line segment to discretize L is calculated from the co-ordinates x_1, y_1 and x_n, y_n of the starting point and the endpoint, respectively. The mean value of the expected element sizes in the starting and the endpoint $\bar{f} = \frac{1}{2}(f_1 + f_n)$ is distributed linearly onto the number of intermediate line segments $n = \lceil \frac{L}{\bar{f}} \rceil$ in proportion to the weighted distances giving $n(n-1)$ fractions of length

$$\Delta l = \frac{2(f_n - f_1)}{n(n-1)}. \tag{5}$$

The total length of the distances is checked by

$$l_{tot} = n f_1 + \Delta l \sum_{i=1}^{n} i(n-i). \tag{6}$$

and β is determined by $\beta = \frac{\Delta l}{l_{tot}}$ to scale the single distance values l_i to fit into the total segment length L.
Resulting coupling nodes on sub-domain boundaries are numbered consecutively. The generation of boundary nodes is deterministic. The same algorithm is performed on every processor. Therefore the available information on the co-ordinates of the coupling nodes is the same on every processor and no communication is needed during and after the split.

Parallel data organisation The sub-domains are recursively and simultaneously generated in parallel. Initially, the problem is loaded onto all processors. All the processors perform the first split. Processors $0, \ldots, p/2-1$ are assigned the first part of the domain and processors $p/2, \ldots, p-1$ are assigned the second part of the domain. The assignment is performed straightforward by the identity mapping, namely sub-domain 0 is mapped onto processor 0, sub-domain 1 is mapped onto processor 1 and so on. This is optimal with respect to a hypercube-like parallel communication architecture. For mesh topologies more sophisticated assignment schemes have to be incorporated at this stage of the algorithm.
In the second and in all the following recursion levels the number of newly generated nodes is counted first in every group of processors performing the same split. Than the total number of coupling nodes is exchanged between the interested processor groups and last the globally unique names are assigned to the coupling nodes. Collective communication primitives (`MPI_Allreduce`) synchronise the processor groups.
This approach obviously involves parallel overhead by redundant operations. On the other hand, assuming a multiprocessor machine operating in single user mode, the subsequent splitting and distributing of work will not result in extra availiablity of resources. The redundant operations reduce the need for communication. This advantage is superior and guarantees the efficiency of the proposed method.
After the geometry split the mesh generation is performed sequentially on every processor without further communication on the last recursion level. the results of the parallel mesh generation are shown in the following Sections.

3 Results for the advancing front mesh generator

Results of the proposed method are compared with the traditional sequential approach of advancing front mesh generator implemented as described in [26]. Parameters to be compared are the mesh quality, the partitioning quality and the parallel efficiency of the mesh generation. The impact of the somewhat different mesh on the solution of PDE will be reported in a follow up-paper. Two examples are composed here with detailed result data. The first example (m1) has a simplified geometry without mesh grading. Figure 4 shows the mesh generated on 1 and on 4 processors, respectively. The other example is a graded mesh. Figure 5 shows the mesh generated an 1 and on 8 processors, respectively.

3.1 Mesh quality

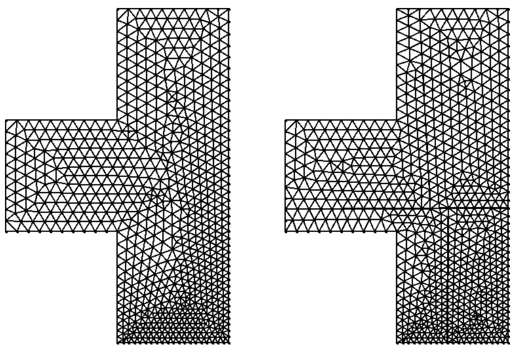

Figure 4: *Serial and parallel mesh generation example m1*

Meshes generated by the proposed parallel generation algorithm differ from sequentially generated meshes with respect to the additional line segments of the sub-domain cuts. These line segments are clearly visible in the parallel generated

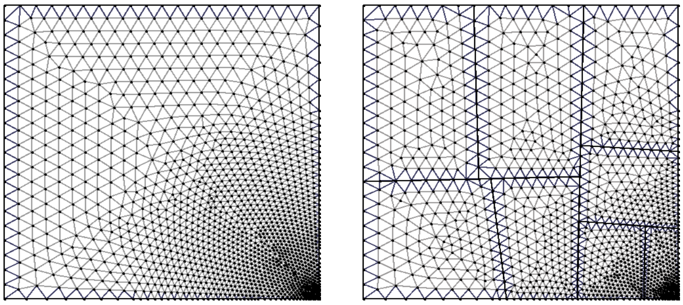

Figure 5: *Serial and parallel mesh generation example m2*

mesh. They are not part of the sequentially generated mesh. The main advantage of these line segments is the a-priori short cut length. Local improvement of the mesh by flipping single elements between sub-domains to shorten the sub-domain boundaries is not necessary.
Smoothing the element shapes in the parallel generation process is restricted to the elements of a single sub-domain. Nodes can only be moved inside the sub-domains. Intersubdomain smoothing is not implemented to avoid communication. Nevertheless, the comparison of the resulting minimum angle distribution for a number of triangulations shows similar results for the sequential and the parallel case, respectively (see Table 1 with results for the generation of Example m2 (compare Figure 5) on different numbers of processors).

Table 1: *Minimum angle distribution for Example m2 (compare Figure 5)*

Number of processors	1	2	4	8
$\bar{\alpha}$	59.73	59.70	59.70	59.66
σ	7.43	7.44	7.74	7.96

3.2 Partitioning quality

The partitioning has to meet the following requirements

- load balance,
- interprocessor communication and
- network complexity or sub-domain connectivity.

The attraction of the inertial method results from the perfect solution to the last two requirements ([11, 14]). The inertial method is a member of the recursive bisection partitioning class. Therefore, the mapping of the resulting mesh is well investigated and the network complexity is bounded. Straight line cuts produced by the application of the inertial method to the original, unrefined sub-domain geometry result in extremely short sub-domain boundaries. Minimum interprocessor communication is achieved without additional smoothing and improving a preliminary partitioning like in the sequential approach based on existing meshes.
The load balance of the resulting partitioning is more critical. Load balance is measured by the difference from an average load per sub-domain in the least square sense. The known sequential implementations of the mesh based inertial method guarantee a perfect load balance.

Table 2: *Difference from average load 4*

Number of processors	Example m1	Example m2
2	12.18%	4.12%
4	13.01%	4.56%
8	6.60%	3.45%

For the examples shown in Figures 4 and 5 the resulting difference from average load are shown in Table 2. The quality of the proposed method depends on three characteristic features

- quality and appropriateness of the input data,
- quality (accuracy and smoothness) of the mesh density function, and
- the number of the processors.

The calculation of the principal moments of the mesh density function is an approximation to the actual mesh distribution. The accuracy of the approximation depends on the accuracy of the numerical integration. The numerical integration is performed for a triangular mesh generated from the given point set. If the given point set does not reflect the users description of the mesh distribution or the triangular mesh is to coarse to transfer sufficient information the estimation of the mesh density and the load distribution will be less accurate. For a given problem the loss in accuracy grows with the number of processors. Errors in the estimation of the load distribution are propagated with the recursion levels.

3.3 Parallel efficiency

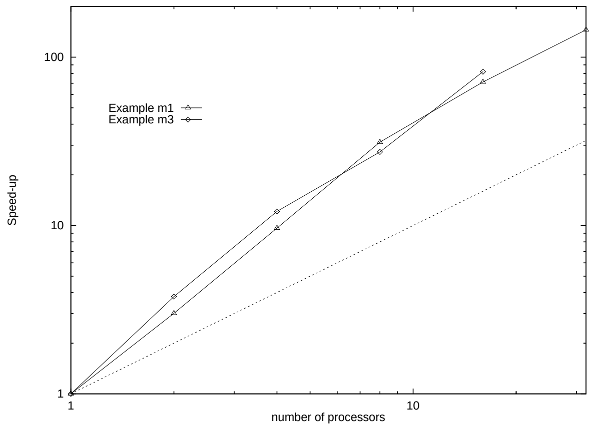

Figure 6: *Speed-up of parallel mesh generation*

Parallel efficiency depends on the time complexity of the mesh generator itself and on the parallel overhead due to additional or not parallelisable parts of the algorithm. The parallelisation overhead introduced by additional communication is not significant for the proposed method. All the communications needed are performed completely synchronised and can be implemented as collective communications of type `ALLREDUCE` within processor groups (compare details of MPI implementations in [12]). These communications are known to be implemented very efficiently in logarithmic time on a large number of multiprocessor architectures (see [31, 4]).

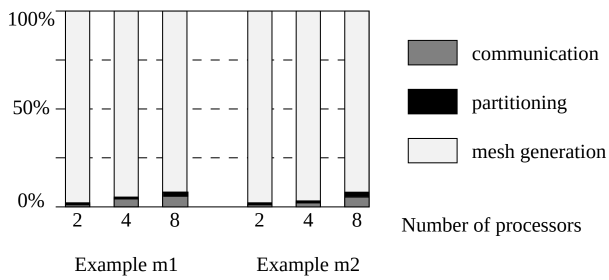

Figure 7: *Runtime distribution for parallel mesh generation*

The mesh generation process is perfectly parallel. Therefore, the application scales very well. Due to the over-linear complexity of the meshing algorithm the speed-up is super-linear (see Figure 6). Runtime analysis on a Parsytec PowerXPlorer-32 (MPC 601 with Parix 1.3 and gcc 2.7.1) shows the minimal impact of the additional parallelisation overhead as shown in Figure 7.
The ratio between the time for the mesh generation and the time spent on communication determines the efficiency of the parallel approach. The time for the communication is bound by the number of messages to be exchanged. The number of messages to be exchanged depends on the number of processors, only. It is independent from the problem size. Therefore, the efficiency of the proposed method can be tuned by scaling up the problem size - increasing the number of elements to be generated on every sub-domain - for a given processor configuration.

4 Results for the Delaunay triangulation mesh generator

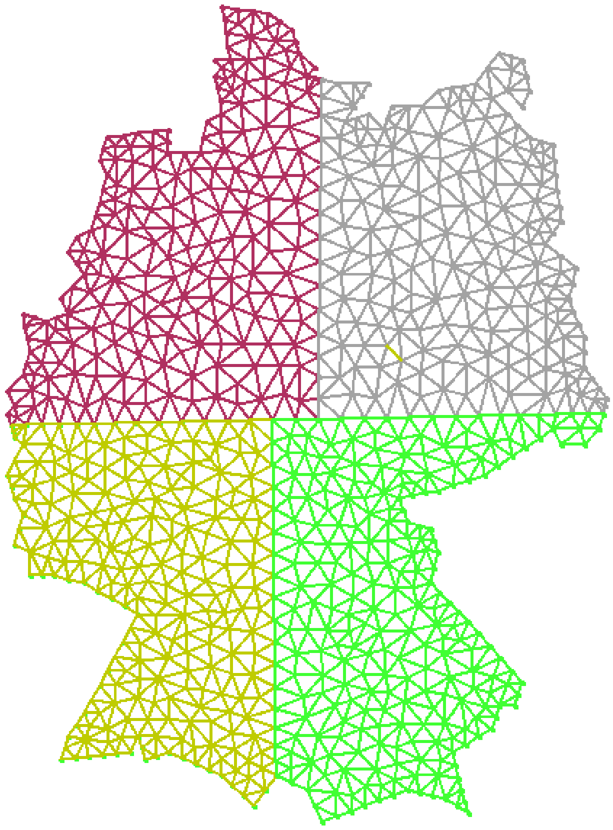

Figure 8: *Parallel mesh generation example g4*

Alternatively, we have implemented the quality mesh generator and Delaunay triangulator Triangle (version 1.3) as provided by J. Shewchuk [29], and referred to in the following as *voronoi*. The software is freely available from *netlib* in the *voronoi* directory and is copyrighted by the author.
We have introduced an additional example (g4, see Figure 8) with a complex boundary description resulting in more time consuming update of the sub-domain boundaries compared to the previously chosen examples.

4.1 Mesh and partitioning quality

The results obtained for mesh and partitioning quality are in the same range as for the advancing front mesh generator described before in section 3. Both for uniform and graded meshes the estimated element distribution and the calculated split result in a fairly balanced partitioning. Due to the different refinement strategies the resulting meshes are not identical.

4.2 Parallel speed-up

The main difference for the usage in our parallel mesh generation framework to the previously described application of the advancing front mesh generator is the bounded runtime complexity of the Delaunay based mesh generator which can be estimated roughly by $O(n\log n)$. Therefore, reducing the problem size will reduce the runtime not so dramatically. This has significant impact on the parallel efficiency, as shown in Figures 9 and 10.
Although, example g4 requires significantly more additional computations to check the sub-domain boundaries, the impact on the parallel runtime is rather small. Speed-up and efficiency are in between the range spanned by the previously introduced examples.
The degraded speed-up for more than eight processes is due to the fact that the parallel overhead outperforms the parallelisation gain of the extremely efficient algorithm. Efficiently parallelising the $O(n\log n)$ algorithm would require a perfect parallelising $O(\log n)$ overhead. Unfortunately, the computational load for the recursive splitting of the geometric domain into p sub-domains grows at least with $O(\log p)$ and is more or less independent from n. The splitting is the major part of the parallel overhead. Instead of reducing the overhead with increasing processor numbers or at least a slower increase than the increase in parallel work we find a fast growing parallel overhead and a drastically reduced parallel work.

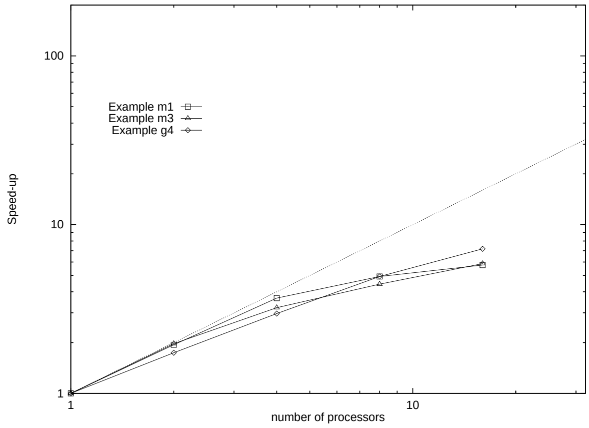

Figure 9: *Speed-up of parallel mesh generation*

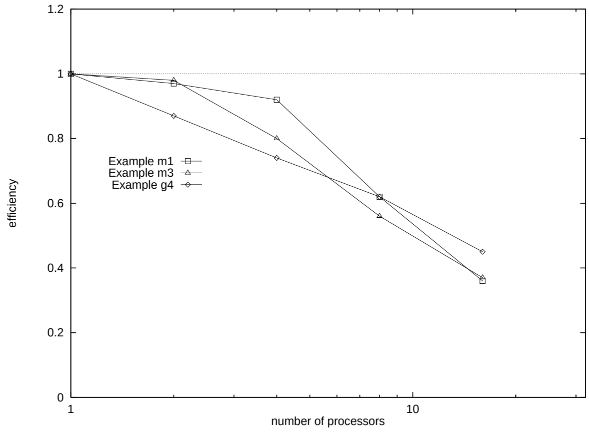

Figure 10: *Efficiency of parallel mesh generation*

Inspecting the actual runtime distribution given in Figure 11 shows the dominating partitioning time within the overall process. Nevertheless, the figure shows the ratio between different fractions of the algorithm. We observe still a reduction in the total runtime, even for eight and 16 processes.
This observations holds for the advancing front mesh generator shown in Section 3 as well. But the impact will be dominant for a significant larger number of processors. Obviously, parallelising a efficient serial algorithm is more challenging.

5 Summary

The method presented generates adaptive meshes in parallel. Input data origin from CAD description. The partitioning is based on the geometric description. Triangulation in advance is not necessary. The mesh generation process works on the partitioned geometric description.
Results are comprehensive and promising. The mesh quality is ensured in the parallel approach. The quality of the partitioning is comparable to the mesh based methods. Additional improvement by local heuristics, flipping of elements between sub-domains to minimise the interprocessor communication, and optimising sub-domain shapes is not necessary. The algorithm is scalable.

Acknowledgments

The work presented is part of a project concerned with parallelisation of structural mechanics applications. This project was funded by the Deutsche Forschungsgemeinschaft, Bonn-Bad Godesberg, Germany.
The authors acknowledge the assistance and the useful discussions with the Structural Engineering Computational Technology Group at the Mechanical & Chemical Engineering Department of the Heriot-Watt-University Edinburgh under the leadership of Prof. B.H.V. Topping. Co-operation with this group is funded by the Academic Research Council and the the Deutsche Akademische Austauschdienst under the ARC-project scheme.
The first author is grateful to the Alexander von Humboldt-Foundation for the Feodor-Lynen-scholarship.

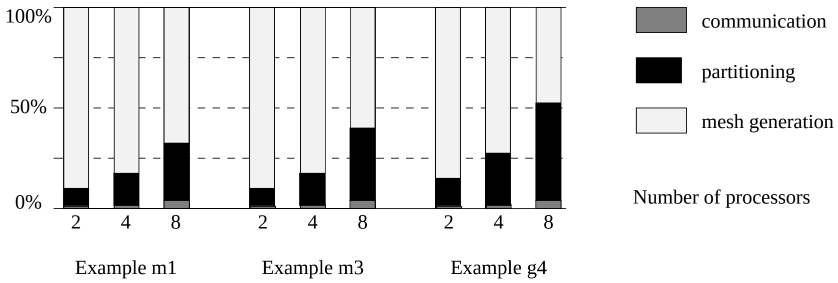

Figure 11: *Runtime distribution for parallel mesh generation*

The authors acknowledge the provided mesh generation software by J. Olden, Technische Universität Darmstadt and J. Shewchuk, Carnegie Mellon University.

References

[1] S. T. Barnard and H. Simon. A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. *Proceedings of the 7. SIAM conference on Parallel Processing for Scientific Computing*, 18:627–632, 1995.

[2] S.T. Barnard and H.D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[3] E.R. Barnes and A.J. Hoffmann. Partitioning, spectra and linear programming. Technical Report RC 9511 (No. 42058), IBM T.J. Watson Research Center, 1982.

[4] M. Barnett, D.G. Payne, R. van de Geijn, and J. Watts. Broadcasting on meshes with wormhole routing. Technical report, Univ. of Texas at Austin, November 1993.

[5] S.H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, 3:207–213, 1981.

[6] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. Technical Report RF-008-94, Universität Paderborn, June 1994.

[7] R. van Driessche and D. Roose. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, 21:29–48, 1995.

[8] R. van Driessche and D. Roose. Dynamic load balancing of iteratively refined grids by an enhanced spectral bisection algorithm. Workshop Dynamic load balancing on MPP systems, Daresbury, November 1995.

[9] R. van Driessche and D. Roose. Dynamic load balancing with a spectral bisection algorithm for the constrained graph partitioning problem. In B. Hertzberger and G. Serazzi, editors, *High-Performance Computing and Networking*, LNCS 919, pages 392–397. Springer, 1995.

[10] C. Farhat. On the mapping of massively parallel processors onto finite element graphs. *Computers & Structures*, 32(2):347–353, 1989.

[11] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering*, 36:745–764, 1993.

[12] Message Passing Interface Forum. Document for a standard message-passing interface. Technical Report CS-93-214, University of Tennessee, November 1993. Available on **netlib**.

[13] G. Globisch. PARMESH a parallel mesh generator. Technical Report SPC 93_3, Chemnitzer DFG-Forschergruppe "Scientific Parallel Computing", TU Chemnitz-Zwickau, Fakult"at Mathematik, June 1993.

[14] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND 93-1301, Sandia Natl. Lab., Albuquerque, NM, June 1993.

[15] D. C. Hodgson and P. K. Jimack. Parallel generation of partitioned, unstructured meshes. Technical Report 94/19, University of Leeds, School of Computer Studies, June 1994.

[16] Z. Johan, K.K. Mathur, and S.L. Johnsson. An efficient communication strategy for finite element methods on the connection machine CM-5 system. Technical Report TR 256, Thinking Machines, 245 First Street, Cambridge MA 02142, 1993.

[17] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, Dept. Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[18] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning ans sparse matrix ordering. Technical Report 95-036, Dept. Computer Science, University of Minnesota, Minneapolis, MN, March 1998.

[19] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 29:291–307, 1970.

[20] A.I. Khan and B.H.V. Topping. Parallel adaptive mesh generation. *Computing systems in Engineering*, 2(1):75–101, 1991.

[21] C. L. Lawson. Software for c1 surface interpolation. In J. R. Rice, editor, *Mathematical Software*, volume III, pages 161–194. Academic Press, New York, 1977.

[22] R. Leland and B. Hendrickson. An empirical study of static load balancing algorithms. In *Scalable High-Performance Computing Conf. 1994*, pages 682–685. IEEE Computer Society Press, May 1994.

[23] T. Lengauer. *Combinatorical algorithms for integrated circuit layout.* Teubner-Verlag, Stuttgart, 1990.

[24] R. Loehner, J. Camberos, and M. Merriam. Parallel unstructured grid generation. *Computer Methods in Applied Mechanics and Engineering*, 95:343–357, 1992.

[25] S. Moitra and A. Moitra. Parallel grid generation algorithm for distributed memory computers. Technical Report 3429, NASA, NASA, Langley research Centre, Hampton VA, 23681-0001, February 1994.

[26] J. Olden. Finite-Element-Analyse von Plattentragwerken durch adaptive Software-Techniken. *Dissertation, Technische Universität Darmstadt, Fachbereich Bauingenieurwesen*, November 1997.

[27] A. Pothen, H.D. Simon, and K.-P. Liou. Partitioning sparse matrice with Eigenvectors of graphs. *SIAM J. Matrix Appl.*, 11(3):430–452, 1990.

[28] J. Ruppert. A new and simple algorithm for quality 2-dimensional meh generation. Technical Report UCB/CSD 92/694, Computer Science Division, Univ. of California at Berkeley, Berkeley, CA 94720, 1992.

[29] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133. Association for Computing Machinery, May 1996.

[30] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2/3):135–148, 1991.

[31] E.A. Varvarigos and D.P. Bertsekas. Communication algorithms for isotropic tasks in hypercubes and wraparound meshes. *Parallel Computing*, 18:1233–1257, 1992.

[32] C. Walshaw, M. Cross, and M.G. Everett. A localized algorithm for optimising unstructured mesh partitions. *Int. J. Supercomputer Appl.*, 9(4):280–295, 1996.

[33] D.R. Williams. Adaptive parallel meshes with complex geometry. Technical Report CCSF-7-91, Concurrent Supercomputing Facilities, California Institute of Technology, Pasadena CA, 1991.

[34] D.R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–491, 1991.

[35] P. Wu and E.N. Houstis. Parallel adaptive mesh generation and decomposition. *Engineering with Computers*, 12:155–167, 1996.