Abstract

Finite element programs require a discretization of the geometry like a refined computational mesh. The size of the resulting elements should often be different in different parts of the mesh. The mesh size parameters origin from a priori knowledge or from a posteriori error estimation in a self-adaptive finite element environment.

Large-scale problems in computational mechanics are solved preferably on parallel computers. Domain decomposition is usually necessary in these cases. This decomposition is often based on existing discretization.

In contrast to the common approach to generate and split the finite element mesh on a sequential machine to perform parallel solution we propose a parallel split and generate method without an pre-exisiting mesh to avoid the serial bottleneck and to improve the partitioning quality.

Input parameters for the proposed parallel adaptive mesh generator are geometry information from usual CAD systems, mesh density information and finite element modeling parameters like boundary conditions, loadings and material properties. Other input parameters can result from the computed problem like soil section in discretization of geological areas. The method ensures an equally distributed load according to equally distributed number of finite elements.

1 Introduction

Modeling a physical phenomena is often only possible by using unstructured, graded meshes. The finite element method allows to use this kind of discretisation in contrast to finite difference method using structured or blockstructured computational meshes.

The mesh generation of unstructured, graded meshes is a well-known hard problem in finite element method. Usually, the input data for the unstructured meshes result from CAD data.

The need to generate finite element meshes quickly is a common requirement of 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 partitioning of the computational data and mapping them onto a particular parallel machine.

For distributed memory parallel machines geometrically motivated partitioning is very efficient to construct fast solvers. Parallel mesh generation avoids additionally the bottleneck of serial mesh generation arising from storing and partitioning large computational meshes.

The partitioning problem is known to be NP-complete (proof see [21]). It 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, usefull heuristic approaches are available (see [5], [10]).

A large number of research groups works in the field of parallel mesh generation [31], [22], [13], [23], [15] [29] [32]. 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.

The generated partitioning should ensure connected and well-shaped subdomains on every processor and equally distributed numbers of elements (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 [25] for improving graph ordering for sparse matrix factorisation). The recursive partitioning is well suited for a near optimal mapping of the partitioning to hypercube and mesh topologies.

Although spectral bisection produces best quality decompositions (compare [27], [30], [20]) the complexity of the numerical Eigenvalue problem to be solved is significant. Therefore, multilevel methods reduce the Eigenvalue problem by gathering elements of the fine 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 Driesche [8], [7], [9] 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 subdomain shape. Techniques applicable both for serial and parallel improvements are Kernighan-Lin heuristic [18] 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. Partitioning cuts are introduced as additional geometric constraints to the original data. The quality of the partitioning is directly ensured by straight cut lines. The resulting subdomains are well shaped. Final improvement by a local heuristic is not necessary. The generated partitions are nearly optimal mapped to the underlying hardware topology.

2 Parallel Mesh Generation

2.1 Input data

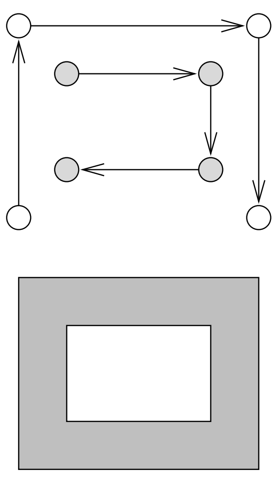

Figure 1: *Input data of mesh generator*

The two-dimensional geometry is described by linear primitives. These data origin from a CAD description and consists of points, lines and polygons describing the outline and holes of a two-dimensional computational domain, properties assigned to subdomains, polygons (or boundaries of subdomains), lines (segments of polygons) and single points (see Figure 1).

Points are assigned information on the expected finite element size. Expected element size is interpolated between known 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 outer boundary of the two-dimensional region. Holes are described by special tags to polygons. Additionally, lines and points can reflect lines or points in the computational model.

2.2 Inertial method

In contrast to the usual approach we devide the geometry before generating any mesh. This is done by the inertial method [11]. The domain is split by the inertial method into two equally weighted subdomains along the axis of the larger inertial momentum. The quality of the cut is problem dependend. The inertial mehtod is normally applied to pre-existing meshes.

For directly partitioning of the CAD geometry, cut lines are added to the geometry description of the CAD data and extend the input data of the unchanged meshing algorihm.

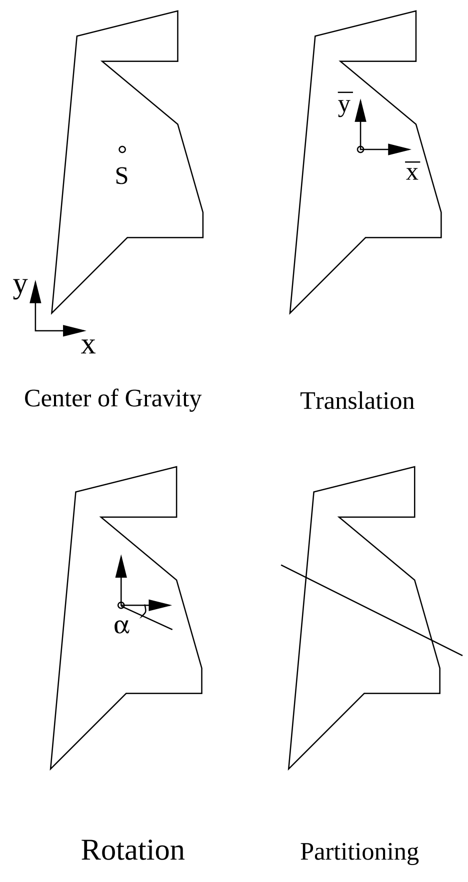

Figure 2: *Recursive bipartition by inertial method*

The mesh density is governed by a pointwise defined distribution function $f(x,y)$. Usually, such a function defines the requested element size. This element size defines the number of elements in the surrounding of these points. To ensure equally distributed computational load in the mesh generation and in the 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.

We are using in the two-dimensional case for an expected element size w the function $f(x,y) = \frac{1}{w^2}$ measuring the element maximum diagonal length or the surrounding circle, respectively. Alternatively, we are using the function $f(x,y) = (\bar{w} - w)^2$ with $\bar{w}$ indicating a mean or maximum element side length. Both functions are well suited for a weighting function applied to the domains describing the problem to be used in the inertial method.

One recursion step of bipartition is performed as follows (see Figure 2). Initially, the center of gravity of the polygonial bounded geometry is calculated by numerical integration with

$$x_s = \frac{\int x \cdot f(x,y) dA}{\int f(x,y) dA} \tag{1}$$

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

and the problem is translated into the $\bar{x}, \bar{y}$ co-ordinate system. The inertial momentum are computed by

$$I_{\bar{x}} = \int \bar{y}^2 \cdot f(\bar{x},\bar{y}) dA \tag{3}$$

$$I_{\bar{y}} = \int \bar{x}^2 \cdot f(\bar{x},\bar{y}) dA \tag{4}$$

$$I_{\bar{x}\bar{y}} = -\int \bar{x} \cdot \bar{y} \cdot f(\bar{x},\bar{y}) dA \tag{5}$$

giving the principal direction $tan 2\alpha = \frac{2I_{xy}}{I_x - I_y}$. The domain is split by the principal axis according to the largest principal inertial moment.

The numerical integration is performed for a triangulation constructed from the existing point set. The existing point set has to describe the domain in a user acceptable way. Usually, triangulation of a given domain is arbitrary. Therefore, the user has to ensure the modeling of the mesh density function is appropriate with the given set of points.

2.3 Coupling nodes on the subdomain boundaries

To avoid additional communication the coupling nodes on the subdomain boundaries have to be generated deterministically before generating the mesh on each processor. The problem is twofold. First, the global namespace of the coupling nodes has to be unique. Second, the co-ordinates of the nodes generated on the boundary between two or more subdomains have to be the same to ensure conform meshing.

The subdomains are generated in parallel on recursively smaller subcubes. 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. Assignment (mapping) is performed straightforward by the identity mapping, namely processor 1 holds subdomain 1, processor 2 holds subdomain 2 and so on.

Resulting coupling nodes on subdomain boundaries are generated as described below and are numbered consecutively. The generation of boundary nodes is deterministic. The same algorithm is performed on every processor. The available information on the coupling nodes is the same on every processor. Therefore, no communication is needed after the first split.

In the second and the following recursion levels first the number of newly generated nodes is counted in the subcubes, second the total number of coupling nodes is exchanged between the interested subcubes and third the globally unique names are assigned to the coupling nodes. The communication synchronises the processor pool.

The mesh generation is performed sequentially on every processor without further communication on the last recursion level. In our program we have implemented two different sequential mesh generators. The first is called *mesh* and described in [24], the other is called *voronoi* and described in [26].

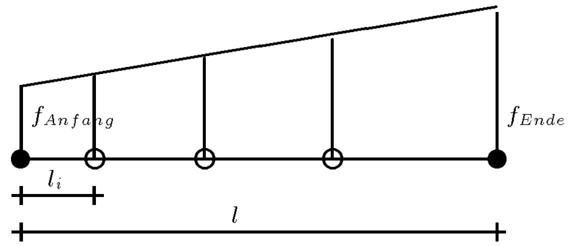

Figure 3: *Discretisation of a line segment*

2.4 Conform meshing on subdomain boundaries

Based on the input data we are using a technique defining piecewise node placements on segments. The 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 illustrated in Figure 3.

The distances of the $n-1$ intermediate nodes on the segment are computed by

$$\begin{aligned} l_1 &= \beta f_1 \\ l_i &= l_{i-1} + \beta \sum_{j=1}^{i} j \end{aligned} \tag{6}$$

whereby β is computed in a preprocessing step, similar to the approach described in [19] as follows: The length of the line segment L is calculated from the node co-ordinates x_1, y_1 and x_n, y_n. The mean value of the expected element sizes in the start and end node $\bar{f} = \frac{1}{2}(f_1 + f_n)$ is distributed linearly on to 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

$$\Delta l = \frac{2(f_n - f_1)}{n(n-1)} \tag{7}$$

The total length of the distances is checked by

$$l_{tot} = n f_1 + \Delta l \sum_{i=1}^{n} i(n-i). \tag{8}$$

and finally a factor 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.

3 Results

Results of the proposed method are compared to the traditional sequential approach of mesh generation as described in [24] and in [26]. Parameters to be compared are the mesh quality, the partitioning quality and the parallel efficiency. Two examples are provided with detailed result data. On example is a simpified geometry without mesh grading (Figure 4 shows the mesh generated on 1 and on 4 processors, respectively), the other example is a graded mesh, see Figure 6.

3.1 Mesh quality

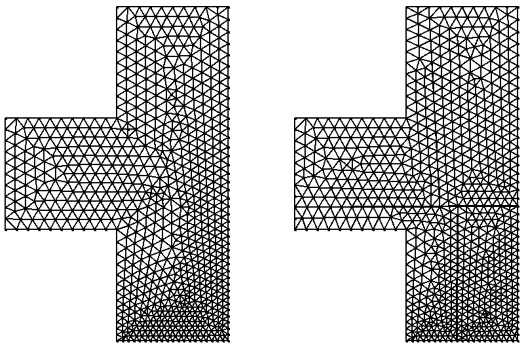

Figure 4: *Serial and parallel (4 procs.) mesh generation Example m1*

Meshes generated by the proposed parallel generation algorithm differ from sequentially generated meshes with respect to the additional line segments of the subdomain cuts. These line segments are part of the parallel generated mesh and 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 subdomains to shorten the subdomain boundaries can be avoided.

Smoothing the element shapes in the parallel generation process is restricted to the elements and nodes of a single subdomain. 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 m3 (Figure 6) on different numbers of processors).

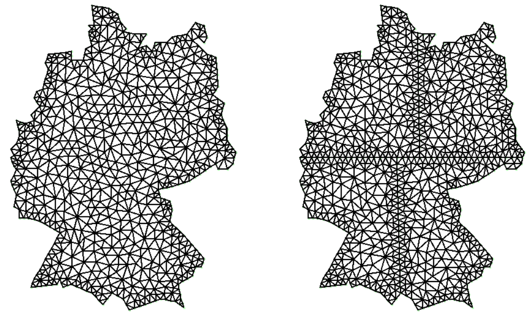

Figure 5: *Serial and parallel (4 procs.) mesh generation Example g4*

Table 1: *Minimum angle distribution for Example m3 (Figure 6) computed with mesh*

nProcs	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

Quality of the partitioning is estimated by the following features

- load balance,
- interprocessor communication and
- network complexity or subdomain connectivity, respectively.

The attraction of the inertial method results from the perfect solution to the last two features. 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 subdomain geometry results in extremely short subdomain boundaries. Minimum interprocessor communication is achieved without additional smoothing and improving a preliminary partitioning like in the sequential approach based on exisiting meshes.

The load balance of the resulting partitioning is critical. Load balance is achieved by the minimum mean difference from an average load per subdomain in the least square sense. The known sequential implementations of the mesh based inertial method guarantee a perfect result.

For the examples shown in Figures 4 and 6 the resulting mean difference from average load are shown in Table 2. The quality of the proposed method depends on three characteristic features

- quality and appropriatness of the input data,

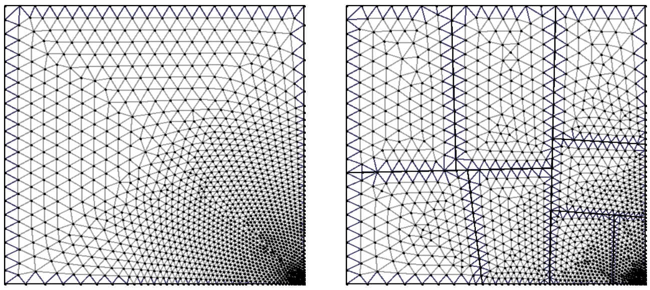

Figure 6: *Serial and parallel (8 procs.) mesh generation Example m3*

Table 2: *Mean difference from average load for examples in Figure 4 computed with mesh*

nProcs	Example m1	Example m3
2	12.18%	4.12%
4	13.01%	4.56%
8	6.60%	3.45%

- quality (accuracy and smoothness) of the mesh density function and
- the number of the processors.

Errors in the estimation of the load distribution are propagated with the recursion levels. The accuracy of this estimation depends on the accuracy of the numerical integration. The numerical integration is performed for intermediate triangular meshes generated with 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 on the mesh the estimation of the load distribution will be less accurate. For a given problem the loss in accuracy grows with the number of processors.

3.3 Parallel efficiency

Parallel efficiency depends on the time complexity of the method itself and on the parallel overhead due to additional or not parallelisable parts of the algorithm. Parallelisation overhead introduced by additional communication is not significant for the proposed method.

The algorithm works with logarithmic communication complexity. 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 in logarithmic time very efficiently on a large number of multiprocessor architectures (see [28], [4]).

The mesh generation process is perfectly parallel. The complexity of the two implemented sequential mesh generation algorithm are very different. The complexity of the algoritm *mesh* is overlinear so the speedup is superlinear (see Figure 9). Runtime analysis shows the minimal impact of the additional parallelisation dependent overhead as shown in Figure 7.

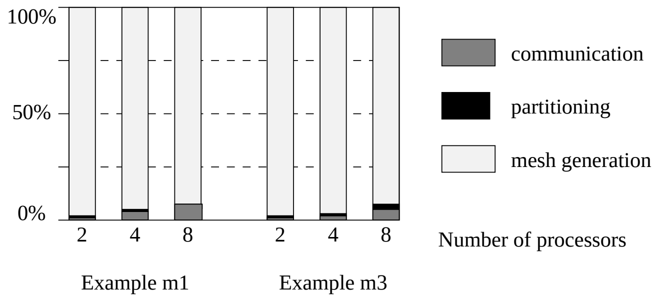

Figure 7: *Runtime distribution for parallel mesh generation computed with mesh*

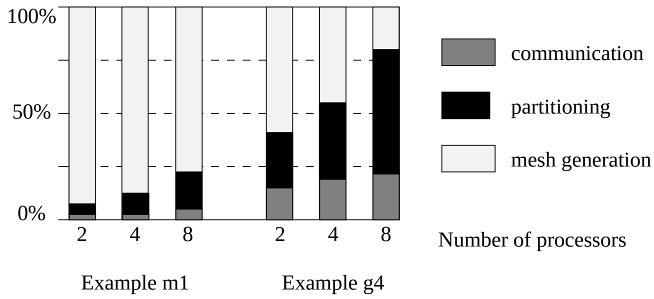

Figure 8: *Runtime distribution for parallel mesh generation computed with voronoi*

The other meshing algorithm has linear complexity. The speed-up for this case is shown in Figure 10. Runtime analysis shows that in this algoritm the impact of the additional parallelisation dependent overhead is large by using upper numbers of processors. In figure 11 is shown a computation of the example *m3* with a constant number of elements per processor. The number of elements per processor is in this case 150.000. Here you can see the scalebility of the parallel mesh generation algorithm.

The ratio between the time for the mesh generation to 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 on 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 subdomain - for a given processor configuration.

4 Conclusion

The method presented generates adaptive meshes in parallel. Input data origin form 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. Mesh quality is ensured in the parallel approach. The quality of the partitioning is comparable to the mesh based methods. Addional improvement by local heuristics, flipping of elements between subdomains to minimise the interprocessor communication and optimise subdomain shapes is

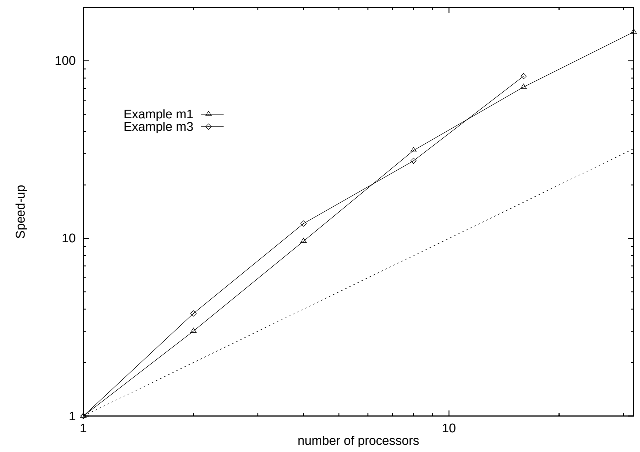

Figure 9: *Speed-up of parallel mesh generation computed with mesh*

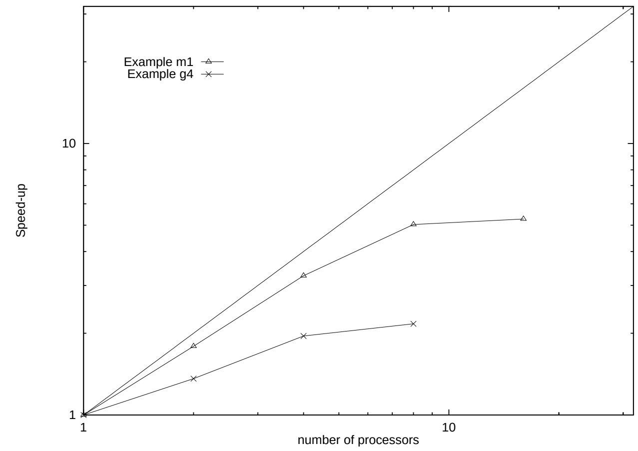

Figure 10: *Speed-up of parallel mesh generation computed with voronoi*

not necessary. The parallel overhead can be neglected. The parallel algorithm is scalable.

Acknowledgments

The work presented is part of a project concerned with parallelisation of structural mechanics applications. This project is currently funded by the Deutsche Forschungsgemeinschaft, Bonn-Bad Godesberg, Germany.

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** (1995), 627–632.

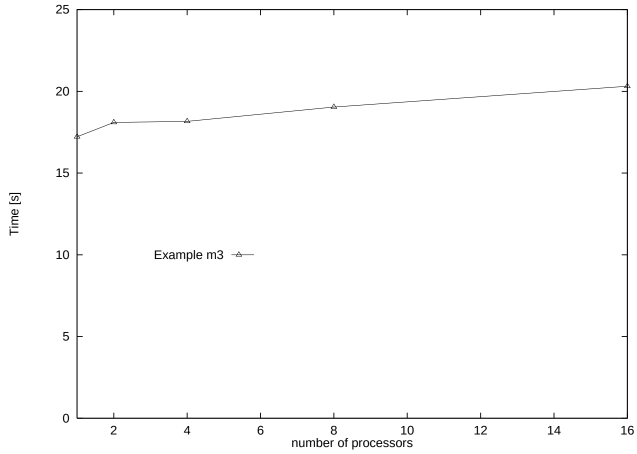

Figure 11: *Time of parallel mesh generation with a constant number of elements per transputer computed with voronoi*

[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** (1994), no. 2, 101–117.

[3] E.R. Barnes and A.J. Hoffmann, *Partitioning, spectra and linear programming.*, Tech. 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*, Tech. report, Univ. of Texas at Austin, November 1993.

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

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

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

[8] ———, *Dynamic load balancing of iteratively refined grids by an enhanced spectral bisection algorithm*, Workshop Dynamic load balancing on MPP systems, Daresbury, November 1995.

[9] ———, *Dynamic load balancing with a spectral bisection algorithm for the constrained graph partitioning problem*, High-Performance Computing and Networking (B. Hertzberger and G. Serazzi, eds.), LNCS 919, Springer, 1995, pp. 392–397.

[10] C. Farhat, *On the mapping of massively parallel processors onto finite element graphs*, Computers & Structures **32** (1989), no. 2, 347–353.

[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** (1993), 745–764.

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

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

[14] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, Tech. 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*, Tech. 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*, Tech. 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*, Tech. Report 95-035, Dept. Computer Science, University of Minnesota, Minneapolis, MN, 1995.

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

[19] A.I. Khan and B.H.V. Topping, *Parallel adaptive mesh generation*, Computing systems in Engineering **2** (1991), no. 1, 75–101.

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

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

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

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

[24] J. Olden, *Finite-Element-Analyse von Plattentragwerken durch adaptive Software-Techniken*, Ph.D. thesis, Darmstadt, Techn. Hochsch., 1996.

[25] A. Pothen, H.D. Simon, and K.-P. Liou, *Partitioning sparse matrice with Eigenvectors of graphs*, SIAM J. Matrix Appl. **11** (1990), no. 3, 430–452.

[26] J.R. Shewchuk, *Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, First Workshop on Applied Computational Geometry (Philadelphia, Pennsylvania) (1996), 124–133.

[27] H.D. Simon, *Partitioning of unstructured problems for parallel processing.*, Computing Systems in Engineering **2** (1991), no. 2/3, 135–148.

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

[29] C. Walshaw, M. Cross, and M.G. Everett, *A localized algorithm for optimising unstructured mesh partitions*, Int. J. Supercomputer Appl. **9** (1996), no. 4, 280–295, to appear.

[30] D.R. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations.*, Concurrency: Practice and Experience **3** (1991), no. 5, 457–491.

[31] ———, *Adaptive parallel meshes with complex geometry*, Tech. report, Concurrent Supercomputing Facilities, California Institute of Technology, Pasadena CA, 1992.

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