1995

Parallel Adaptive Mesh Generation and Decomposition

Poting Wu

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
95-012

PARALLEL ADAPTIVE MESH GENERATION AND DECOMPOSITION

Poting Wu
Elias N. Houstis

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-95-012
February 1995

Parallel Adaptive Mesh Generation and Decomposition

Poting Wu and Elias N. Houstis

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907-1398, U.S.A.
e-mail: wu@cs.purdue.edu

October, 1994

Abstract

An important class of methodologies for the parallel processing of computational models defined on some discrete geometric data structures (i.e., meshes, grids) is the so called *geometry decomposition or splitting* approach. Compared to the sequential processing of such models, the geometry splitting parallel methodology requires an additional computational phase. It consists of the decomposition of the associated geometric data structure into a number of balanced *subdomains* that satisfy a number of conditions that ensure the load balancing and minimum communication requirement of the underlying computations on a parallel hardware platform. It is well known that the implementation of the mesh decomposition phase requires the solution of a computationally intensive problem. For this reason several fast heuristics have been proposed. In this paper we explore a decomposition approach which is part of a parallel adaptive finite element mesh procedure. The proposed integrated approach consists of five steps. It starts with a coarse background mesh that is *optimally* decomposed by applying well known heuristics. Then, the initial mesh is refined in each subdomain after linking the new boundaries introduced by its decomposition. Finally, the decomposition of the new refined mesh is improved so that it satisfies the objectives and conditions of the mesh decomposition problem. Extensive experimentation indicates the effectiveness and efficiency of the proposed parallel mesh and decomposition approach.

1. Introduction

The problem of finite element mesh generation, especially for three dimensional regions, is a well known hard problem that has occupied the attention of many researchers for long time. Although significant progress has been made in devising its solution for certain classes of geometric domains [1-5], its solution for general domains is still an open issue. The need to generate finite element meshes quickly is a common requirement of most computational fields and it is an inherent requirement of any adaptive process. Therefore, the need for developing parallel mesh generation techniques is well justified. Several of the proposed parallel solution methodologies for finite element equations are based on the partitioning of the corresponding geometric data and their mapping to the target parallel architecture. Specifically, for message passing machines, the proposed parallel methodologies are based on some "optimal" decomposition of the associated finite element mesh. A formulation of this approach and a description of several mesh decomposition algorithms can be found in ref. [5] and [6]. The disadvantage of this approach is the fact that the associated partitioning problem is NP-complete for general regions and even for the case of polynomial time solutions the degree of the polynomial is too high [7] to have practical importance. Thus, the parallelization of the mesh partitioning phase is necessary.

In this paper we present a parallel methodology that addresses the parallel solution of the mesh generation and its decomposition simultaneously. In addition, we report the performance of its implementation on the nCUBE II machine for realistic two dimensional domains. The paper is organized as follows. Section 2 presents the steps of the proposed parallel mesh generation and decomposition methodology. Section 3 discusses the approached used for mesh generation and gives several justifications for its selection. Section 4 discusses the partitioned techniques implemented for the partitioning of the background mesh. Section 5 defines the subdomain boundary linking phase. The generation of the final mesh and its decomposition are discussed in Sections 6 and 7. A preliminary performance evaluation of this approach is presented Section 8 together with a discussion of the results. Finally, the description of the various algorithms needed to support the implementation of the five phases of the parallel mesh generation and decomposition are listed in the appendix of ref. [8] in some pseudo language.

2. A Methodology for Parallel Mesh Generation and Decomposition

The problems of mesh generation and its partitioning have been addressed by many researchers. In this section we present a formulation of a parallel methodology for solving these two problems simultaneously. Fig. 1 depicts the outline of this methodology for a 2-D region and 4-processor machine configuration. It consists of five phases that can be described as follows.

1. ***Generation of a coarse background mesh.*** In this step we generate an initial "course" adaptive mesh which we call the background mesh. This initial mesh is used to split the given domain into "equivalent" subdomains and to generate in parallel a refinement of the initial mesh in each subdomain (processor). The method used to generate the initial mesh and its local refinements is based on the quadtree approach. The implementation of this phase is described in Section 3.

2. ***Partitioning of the background mesh.*** An "optimal" partitioning of the initial mesh is generated. Some of the partitioning criteria applied include load balancing, minimum interface length, and aspect ratio. Several mesh decomposition schemes are supported which are discussed in Section 4.

3. ***Formulation of the subdomain boundaries.*** For the parallel mesh generation the subdomain boundaries have to be identified and formed from the mesh decomposition data.

4. ***Parallel mesh refinement.*** An adaptive quadtree based mesh refinement scheme is applied in each subdomain. The use of the quadtree node distribution data structure allows the determination of the node refinement before the actual node generation. Therefore, the communication between the processors is reduced to a minimum.

5. ***Refinement of initial domain decomposition.*** After the refinement the subdomain interfaces may be large even with perfect partitioning of the initial mesh. In this phase the initial partitioning is modified based on the refined mesh. The implementation of this phase is described in Section 7.

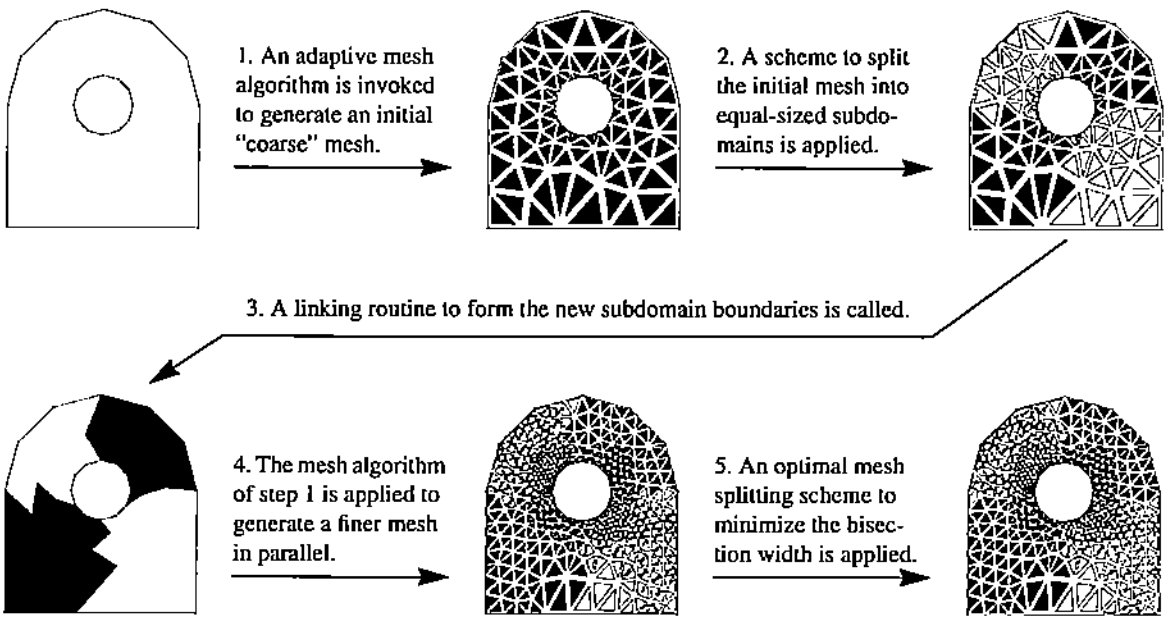
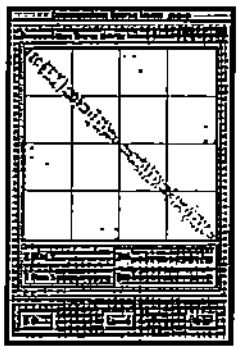


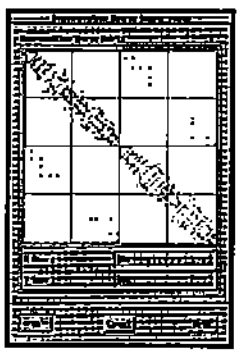
Fig. 1. A methodology for parallel mesh and mesh-decomposition.

Next, we describe the implementation of the above five phases and the needed algorithmic infrastructure. The user interface of the software tool that supports the above methodology is depicted in Fig. 2 and described in ref. [9].

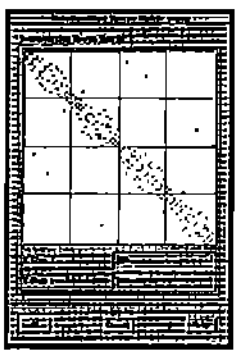
Triangular element mesh generation & element-wise domain decomposition

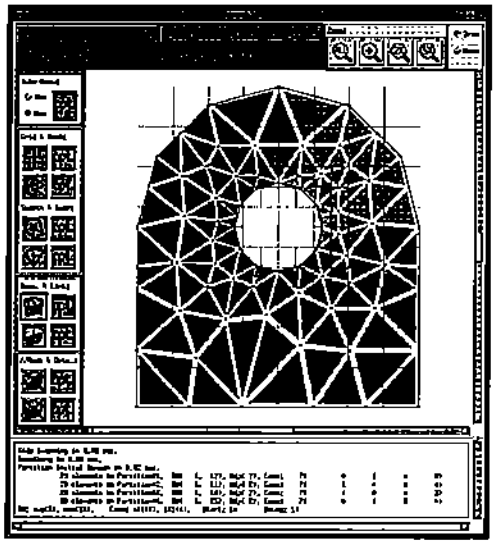


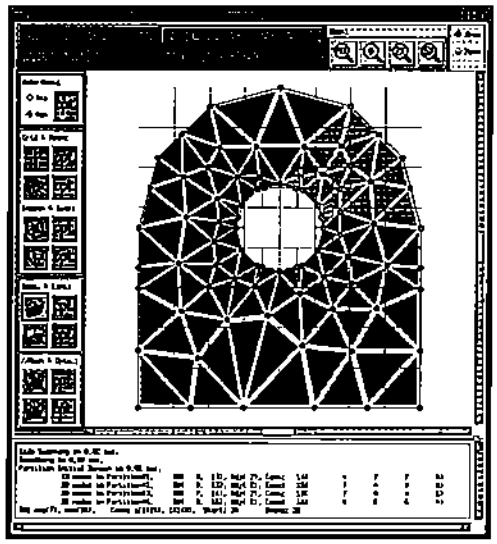
Triangular element mesh generation & node-wise domain decomposition



Quadrilateral element mesh generation & element-wise domain decomposition







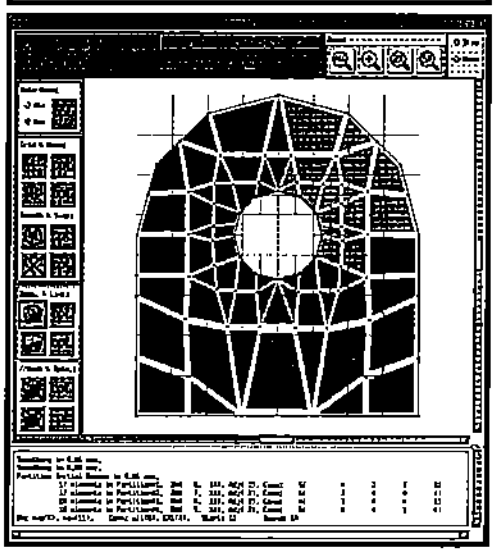


Fig. 2. Interfaces of the mesh generation and decomposition methodology defined in Fig. 1. The structure of the finite element matrix associated with these meshes/ decompositions is depicted on the left windows.

3. The Initial Refinable Background Mesh Phase

Two of the most often used groups of adaptive methods for generation of unstructured meshes are based on the *advancing front* and *quadtree/octree* techniques. The methods in the first group [10-13] are applied on the specified or computed boundaries. In contrast with the first group, the methods of the second group are applied on an initial coarse mesh which is modified by repeated refinement. For the implementation of the methods in the last group, the quadtree/octree data structures are used. A general discussion of these schemes can be found in ref. [14]. A qualitative comparison of these two groups of methods indicates that the quadtree approach can 1) *automatically* derive its input from the geometric information given, 2) easily manage the *smoothness* on the outer boundary and in the inside of the object, 3) support *adaptation* easily, 4) support the implementation of *dynamic* mesh decomposition, and 5) be efficiently *parallelized* since it allows the refinement strategy to be formulated before the actual generation of elements. We have decided to implement the quadtree approach in all phases of mesh generation approach proposed in this paper. For completeness, we next describe the steps of this approach and indicate the various options currently available in the system. These steps are:

Step 1. *Decompose domain into quadtree data structure*: We have implemented the quadtree scheme proposed in ref. [15] that defines the node distribution on its related hierarchical data structure. We create the data structure during the generation of the initial background mesh and maintain a local or global refinement.

Step 2. *Generate element mesh*: In this step, triangular or quadrilateral element meshes are generated by connecting the precomputed nodes in the previous phase. Fig. 3 indicates the topology of elements that are currently supported. The available adaptive refinements include: a) element-wise strategies consisting of regular division and bisection as shown in Fig. 4 [16], and b) node-wise strategies implemented by using the well-defined quadtree data structure. Fig. 5 depicts an example of a node-wise refinement. In this step we have adopted the so called *neutral file output format* [17] for the element mesh.

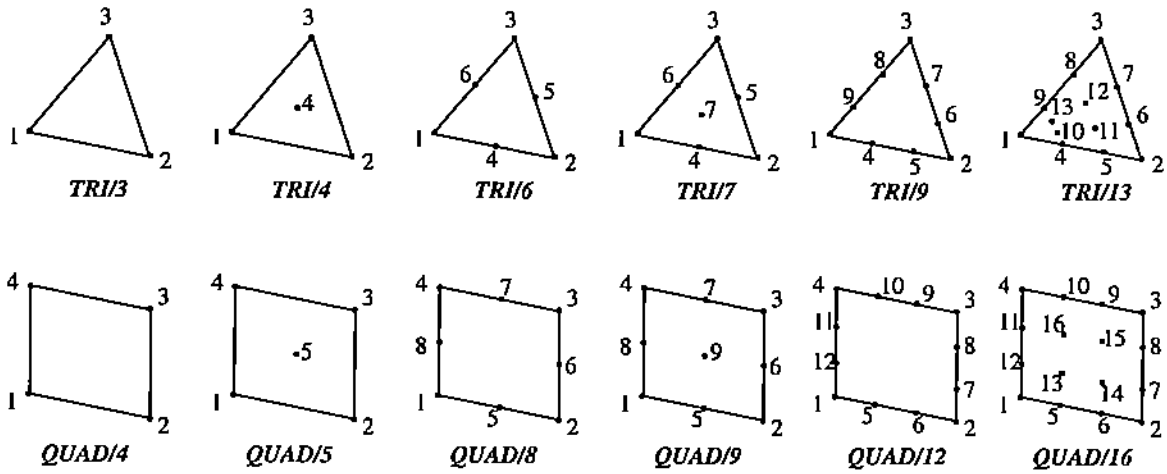


Fig. 3. Mesh element topology supported by the current parallel quadtree implementation.

Fig. 4. Element-wise refinement of a triangle by (i) regular division and (ii) bisection.

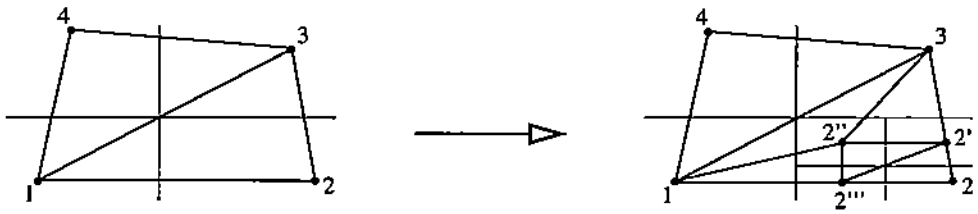


Fig. 5. Example of node-wise refinement.

Step 3. *Adjust generated mesh*: One of the difficulties in unstructured FEM mesh generation is to avoid degenerate elements in certain regions. The usual way to solve this problem is to adjust the element and node distribution. This adjustment includes mesh smoothing and side swapping.

Step 4. *Maintain adjacency lists*: Four types of adjacency lists are maintained in the process of mesh generation [18]. They include node-node adjacency, node-element adjacency, element-node adjacency, and element-element adjacency.

4. The Background Mesh Partitioning Phase

The formulation and implementation of this phase is done at the topological graph of the finite element mesh $G = (V, E)$ of a domain Ω, where V denotes the set of elements or nodes and E is the set of edges of G that represent the connectivity of the vertices V with its neighbors. In this section we describe a number of heuristics for the element-wise or node-wise partitioning of finite element meshes (i.e., graph G) that we have implemented and evaluated in the context of the proposed parallel mesh generation and decomposition strategy. For this we introduce some notation and the partitioning criteria. Throughout, we denote by d_v the number of adjacent vertices to each vertex v of V, $D = \{D_i\}$ the partitioning of G or domain Ω, N_s the set of subdomains in D, and $|V| = N_n$ or N_e the size of the mesh or graph G. The optimality criteria applied is *load balancing* (the subdomains are of almost equal size), *minimum interface length* (minimum number of common edges or nodes between subdomains), and *minimum subdomain connectivity.*

4.1. Neighborhood Search Schemes

First we have implemented a class of enumerative heuristics that are based on some neighborhood search scheme utilizing the connectivity information of the mesh graph G. For these schemes, the partitioning of G is equivalent to the construction of a traversal tree from graph G. Two well known neighborhood search schemes, the *Depth-First Search* (DFS) and the *Breadth-First Search* (BFS) have been considered [19]. If the traversal order scheme remains fixed for the entire mesh

graph G, then the searching strategy is called *strip-wise*. In case the traversal order is allowed to change after the formulation of each subdomain, then the search is called *domain-wise* [20]. The optimality of these searching strategies depends on the starting vertex. It is usually selected as the one with minimum degree of connectivity that usually coincides with a boundary node or element. The criterion for evaluating the optimality of the partitioning obtained by the above schemes is the *bandwidth* (w) of the coefficient of the associated finite element matrix. It has been shown [21] that the maximum partitioning interface C is given by the relation $C = (N_s \times w) / N_n$.

A common search strategy that yields partitionings corresponding to finite element matrices with small bandwidth or profile is the so called *pseudo-peripheral node* (PPN) [22-25]. Their basic idea is to minimize the maximum width w or maximize the depth of the traversal tree, since the bandwidth of the sparse matrix is between w and $2w-1$. One of the common disadvantages of the neighborhood searching strategies is that they often produce disconnected subdomains. One way to prevent this from happening is to follow a traversal order that is based on the degree of connectivity of the graph G. A well known ordering scheme is the *Reverse Cuthill-McKee* (RCM) [26, 27]. In the appendix of ref. [8] we present our implementations of the above search strategies.

4.2. Eigenvector Spectral Search (ESS)

According to this search the vertices V are visited in the order defined by an eigenvector of the Laplacian matrix $L(G)$ of the graph G. The elements of $L(G)$ are defined [28] as follows

$$L_{ij} = \begin{cases} -1 & \text{if } (v_i, v_j) \in E \\ d_i & \text{if } i = j, \text{ where } d_i \text{ is the degree of } v_i \\ 0 & \text{otherwise.} \end{cases} \qquad (1)$$

If we assign a discrete value variable x_i to each vertex of V, then the partitioning problem can be formulated as the following optimization problem

$$\begin{array}{ll} \text{Minimize} & 1/4\, x^T L\, x \\ \text{Subject to} & x^T I = 0, \qquad x^T x = N_n . \end{array}$$

Fielder recognized that the second eigenvector of L represents a good measure of the connectivity of graph G [29-31]. Hendrickson [32] considered combining the use of other eigenvectors to reduce the computing cost and the communication overhead. He introduced the *spectral quadrisection* and *spectral octasection* schemes. For the improvement of the performance of the ESS scheme, a multilevel implementation of ESS has been introduced that involves additional steps such as contraction, interpolation and refinement. These search procedures can be applied in strip-wise and domain-wise form.

A recursive domain-wise implementation of ESS heuristic (RSB) is presented in ref. [33]. A strip-wise implementation of ESS is described in ref. [34]. A multilevel implementation of MRSB appeared in ref. [28]. We have implemented and evaluated the MRSB scheme.

4.3. Coordinate Axis Splitting

This is another class of enumerative schemes whose main characteristic is that they ignore the connectivity information of the mesh graph G. We have implemented three such schemes. They are based on coordinate sorting and partitioning along Cartesian, polar, and symmetric inertial axis of the graph G.

Cartesian axis splitting: In these schemes the Cartesian coordinates of the mesh nodes or the element center of mass are sorted and split along each axis. We have non-recursive and recursive implementations in both strip-wise and domain-wise form. In the case of recursive schemes the bisection direction can vary for each recursive step. One can choose this direction by splitting along the longest expansion which can be easily determined [33]. We have implemented a version of this scheme that compares the "communication cost" of the produced partitioning in both possible directions and chooses the one corresponding to less cost. This scheme turns out to be more expensive than the previous one but gives more accurate partitionings.

Polar/spherical axis splitting: Their basic idea is similar to cartesian axis splitting schemes. In the polar/spherical axis partitioning schemes, the sorting of the coordinates of nodes or element center of mass is done along R, Θ, and Z/α axis. In addition to the available options in Cartesian axis splitting schemes, the origin point can be selected as either center of inertia or center of mass. An implementation of this scheme is described in ref. [35]. Due to the periodicity of the cartesian to polar coordinates transformations, these schemes can produce disconnected subdomains with high probability. In our implementation of this scheme this is avoided by appropriate angle shifting [8].

Inertia axis splitting: This scheme first computes the main symmetry axis from the node coordinates of the mesh or the coordinates of element mass [35]. Then, it splits the domain into several subdomains along this axis. It repeats this step until the predefined number of subdomains is reached. The symmetry axis is obtained by computing the eigenvector corresponding to the largest eigenvalue of the inertia matrix $\boldsymbol{I} = \boldsymbol{A}^T\boldsymbol{A}$ [36]

$$I = A^T A = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i z_i \\ \sum y_i x_i & \sum y_i^2 & \sum y_i z_i \\ \sum z_i x_i & \sum z_i y_i & \sum z_i^2 \end{bmatrix} \tag{2}$$

where $\boldsymbol{A}$ is the matrix of the mesh coordinates. We have implemented this scheme both in strip-wise and domain-wise forms. The domain-wise version is based on recursive bisection approach. An alternative way to compute the main symmetry axis is to use any of the three eigenvectors of the following inertia matrix

$$I' = \begin{bmatrix} \sum\left(y_i^2 + z_i^2\right) & -\sum x_i y_i & -\sum x_i z_i \\ -\sum y_i x_i & \sum\left(z_i^2 + x_i^2\right) & -\sum y_i z_i \\ -\sum z_i x_i & -\sum z_i y_i & \sum\left(x_i^2 + y_i^2\right) \end{bmatrix}. \tag{3}$$

It turns out that the eigenvalues and eigenvectors of I and I' are related [21]. In addition to the axis, the user has to specify its origin that in our implementation can be selected either as the mesh center of inertia or its center of mass.

5. Subdomain Boundary Linking Phase

The natural way to identify (linking) the new boundaries of subdomains is to partition the mesh of the geometric object element-wise before linking. In case of node-wise partitioning the linking routine transforms the node-wise partitioning to element-wise one. In some instances the partitioning schemes might generate disconnected subdomains and additional holes. To eliminate these side effects, some refinement decomposition algorithms, to be discussed later, are applied on the initial partitioning before linking. After the refined mesh decomposition, the linking routine connects the new boundary for each subdomain before proceeding with the final mesh generation in parallel. This is accomplished by separating the outer boundary polygon from the hole polygon(s) [9] using the element-element adjacency list to identify the boundary element and the node-element adjacency list to locate the next boundary element.

6. Parallel Mesh Generation

For the parallel mesh generation various strategies have been proposed. One strategy [37] is to generate the mesh of each subdomain in parallel and generate the mesh of the inter-subdomain region sequentially. In this strategy communication between processor nodes is required for the generation of the inter-subdomain mesh. Moreover, the generated mesh in each subdomain does not always have a global smooth node distribution. In our proposed approach, we have selected the quadtree data structure to supervise the node distribution. Thus, it is easy and efficient to refine it globally before generating the mesh in parallel. Therefore, during the generation of the parallel mesh, there is no need for communication between processors. Furthermore, the global smoothness of the node distribution assures more uniform mesh elements. In addition, we can use the same algorithm like a sequential mesh generator to generate the unstructured mesh on each subdomain in parallel. Similarly, the parallel local mesh smoothing and side swapping can be done by the use of sequential global mesh smoothing and side swapping. Thus, our approach resembles the operation of existing sequential mesh generation codes. Fig. 6 depicts a block view of the implemented parallel mesh generation.

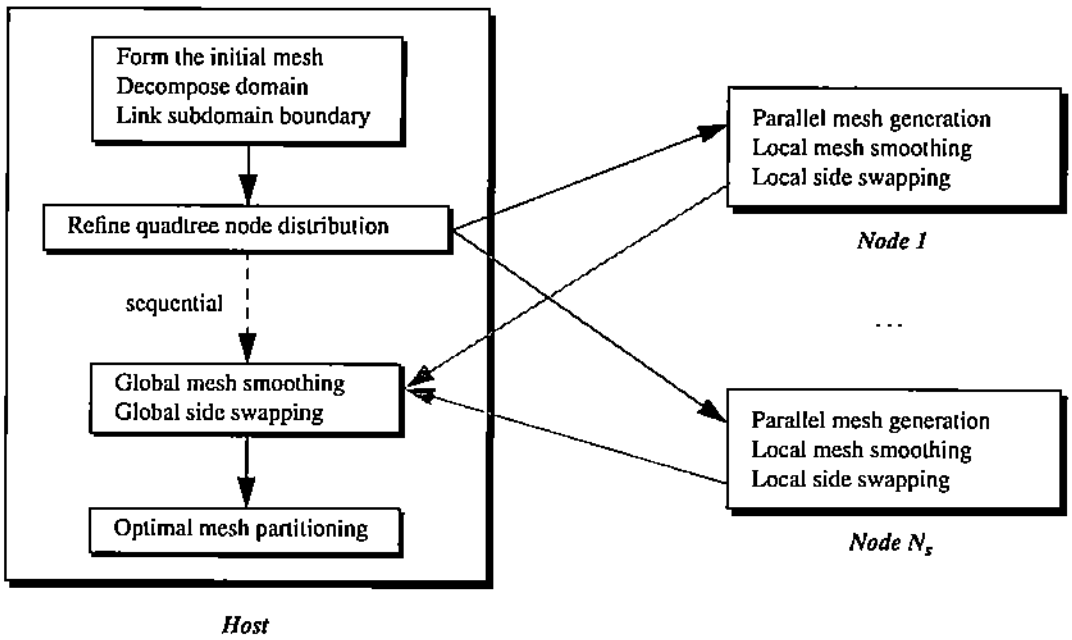


Fig. 6. A strategy for a parallel final mesh generation of domain Ω based on a given coarse background mesh of Ω and an initial decomposition.

7. Refinement of Initial Decomposition Phase

The refined parallel mesh obtained by the parallel phase described in Section 4 is decomposed according to the initial partitioning considered of the background mesh. In general this decomposition is unbalanced and the interface is not minimum. Thus, it must be appropriate refined.

Kernighan-Lin algorithm (KL): This scheme attempts to locate the best k exchange pairs of nodes (elements) among two subdomains using the sum of the first k best gains to search for the best set of exchange pairs of gains based on some cost function. This searching procedure may continue even though some local gains are negative. Therefore, it can locally identify a swapping sequence that will produce the maximum gain. In order to make the improvement as large as possible, this scheme might be forced to visit all possible pairs with considerable increase in the execution time [38]. From most of our experiments, the ratio $r_k = k / N_n$ is usually less than *0.05*. Thus, KL is allowed to make significant redundant work. In our implementation we restrict the checking below $n_k = f_k \times N_n$ where f_k is a user defined factor. For the test results presented in Section 8, the factor $f_k = 0.05$ was used. This gives a speedup more than *300* over the original KL.

Simulated annealing based algorithms: The simulated annealing (SA) algorithm [39] is an optimization approach that attempts to prevent a poor local optimum by allowing an occasional uphill move based on some probabilistic strategy. In general SA converges very slow and it is not applicable for large meshes. We have implemented a double loop modification suggested in ref. [40, 41] that usually converges faster. Another variation of SA is the so called *stochastic evolution* (SE) algorithm. This scheme allows the uphill move probability to be updated whenever it is necessary. To avoid SA searching procedure from cyclically stuck a list of predefined forbidden

moves has been introduced. This list is called Tabu list and the scheme Tabu search [42]. This list is constantly updated and its size effects the performance of the search. A size of *7* is suggested in ref. [42, 43].

Most of the above heuristics are hard to parallelize. To overcome this difficulty *a parallel search heuristic (MOB)* was introduced in ref. [44]. Its basic idea is to swap large number of nodes between two subdomains. All of the above algorithms are described in the appendix of ref. [8].

8. Performance of Parallel Mesh Generation and Decomposition

To test the performance of the proposed parallel mesh generation and decomposition approach, we have selected several geometric objects including the engine rod head (Fig. 1), engine cap (Fig. 7a), engine axis (Fig. 7b), and torque arm (Fig. 7c). The software realizing this approach was run on the nCUBE II and Sun SPARC 2 hardware platforms. The performance of the parallel mesh generator is measured in terms of *fixed speedup*, *processor utilization*, *communication overhead*, and *synchronization overhead* (idle processor time) [45-49]. The last three indicators were estimated using the ParaGraph tool. The performance of the parallel decomposition phase is measured in terms of the satisfiability of the *load balancing*, *interface length*, *subdomain connectivity*, and *bandwidth* of diagonal submatrices in the corresponding finite element matrix obtained using linear triangular elements. All the raw data obtained are reported in ref. [8]. In this paper we report the performance data obtained for the engine axis geometric object and make general observations supported by all data obtained so far.

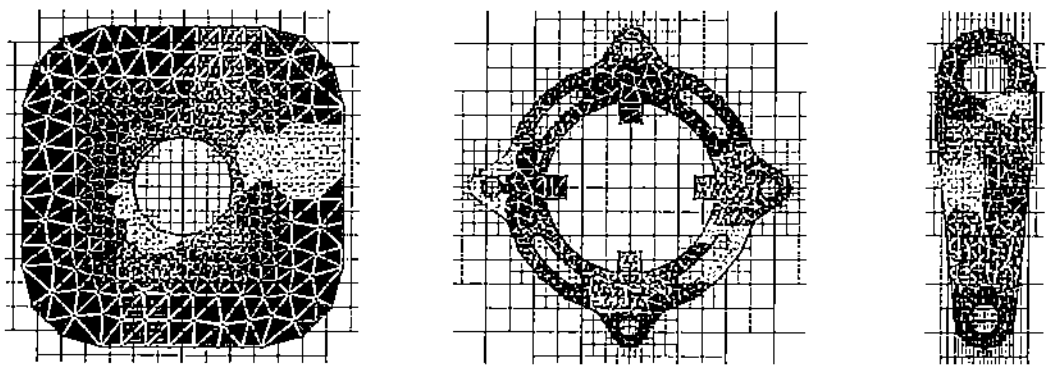
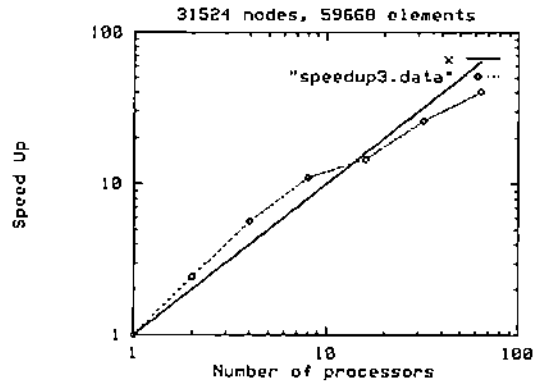


Fig. 7. The geometric objects considered in the performance evaluation of parallel mesh and decomposition approach include (a) engine cap, (b) engine axis, and (c) torque arm.

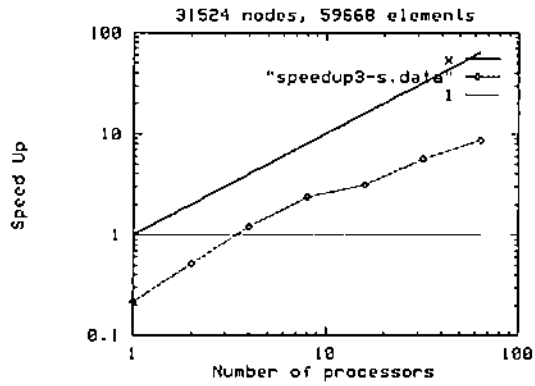
8.1. Performance of Parallel Mesh Generation

In Fig. 8a we display the speedup for the parallel mesh generator proposed with respect to its sequential time (T_1) on a single nCUNE II processor. These data indicate that the scalability of the parallel scheme is almost superlinear. Fig. 8b shows its speedup with respect to the execution time (T_1^s) of a sequential mesh generator measured on a Sun SPARC 2 and converted to the corresponding time on a single nCUBE II processor. A speedup of at least 10 has been observed. Fig. 8c and 8d suggest that the communication overhead is negligeable while the synchronization overhead varies from 5 to 30% with an average of about 15%. In our opinion, this is a noticeable

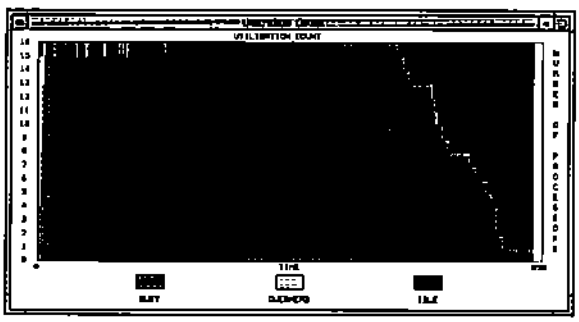
speedup of the mesh generation phase which combined with the speedup obtained from the parallel decomposition phase can reduce significant the cost of the preprocessing stages for parallel FEM computations.



(a) *SpeedUp - parallel : T_1 / T_p*

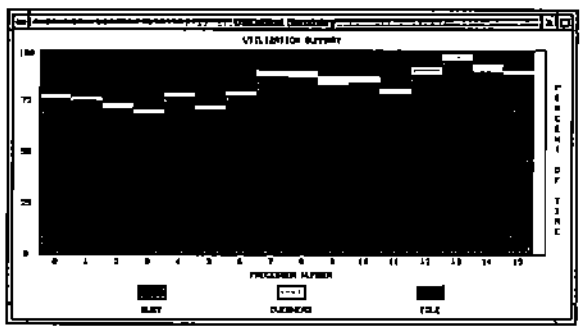


(b) *SpeedUp - sequential : T_1^s / T_p*



(c) *Utilization Count*

states of idle, overhead, and busy as function of time.



(d) *Utilization Summary*

overall cumulative percentage of time in idle, overhead, and busy states.

Fig. 8. The performance of the proposed parallel mesh generation scheme for the engine axis geometric domain. (a) displays the speedup with respect to T_1 and (b) the speedup with respect to T_1^s. (c) shows the processor utilization and (d) the cumulative processor utilization.

8.2. Performance of Constructive Mesh Splitting Algorithm

The objectives of all mesh splitting algorithms considered in this paper include *load balancing*, *minimum interface length*, *minimum number of interpartioning boundary vertices (IBV)*, and *minimum subdomain connectivity*. Several formulations of these partitioning optimality criteria have been proposed. We have adopted the most common way of measuring these partitioning indicators where a) **load balancing** is measured by

$$\left(\max_{i,j} | N_i - N_j | \right) / N_n \qquad i = 1, \ldots, N_s \quad j = 1, \ldots, N_s \quad i \neq j \qquad (4)$$

where N_i and N_j denote the number of nodes in subdomains i and j.

b) **the maximum interface length** is computed by

$$max \sum_{i=1}^{N_s} C_{i,j} \qquad j = 1, \ldots, N_s \quad i \neq j \qquad (5)$$

$$max\ C_{i,j} \qquad i = 1, \ldots, N_s \quad j = 1, \ldots, N_s \quad i \neq j \qquad (6)$$

where $C_{i,j}$ denotes the number of common edges or nodes between subdomains i and j.

and c) **the subdomain connectivity** is computed by

$$max\left(N_{b,j} \cdot \sum_{i=1}^{N_s} C_{i,j} \right) \qquad j = 1, \ldots, N_s \quad i \neq j \qquad (7)$$

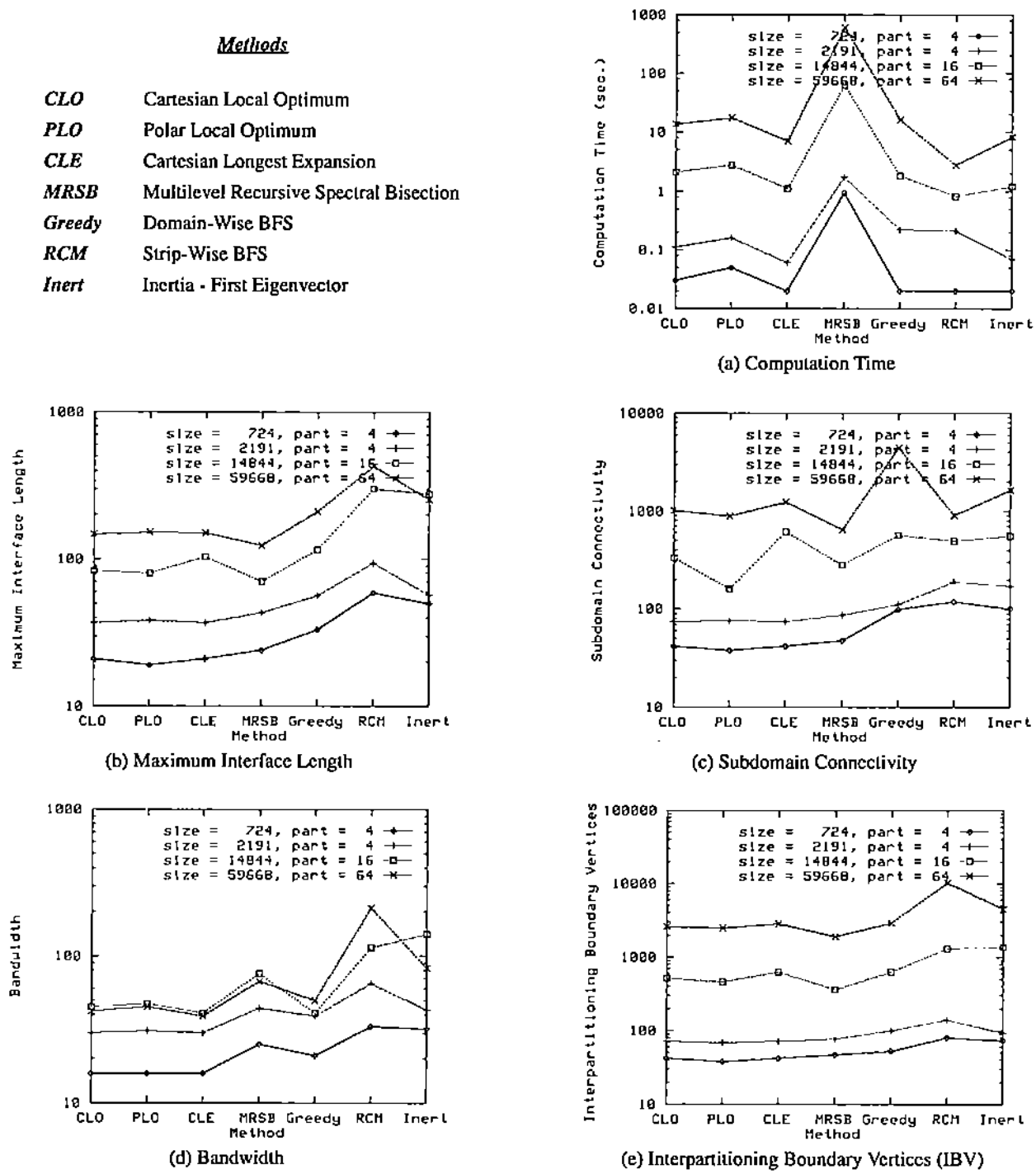
where $N_{b,j}$ denotes the number of neighboring subdomains of subdomain j.

Also, the effect of these optimality partitioning criteria can been seen at the structure of the corresponding finite element matrix $\boldsymbol{K}$. For example load balancing assures a uniform row partitioning of matrix $\boldsymbol{K}$, minimum interface length implies that the interface unknown vector is of minimum length, the minimum subdomain connectivity guarantees minimum number of non-diagonal submatrices, and the ordering of the elements in each subdomain determines the bandwidth of each diagonal submatrix of $\boldsymbol{K}$ which we considered as the fourth partitioning criterion. In Fig. 9 we display the values of the objective functions corresponding to the first three criteria for a fine mesh of the engine axis geometric object.

From Fig. 9a and the rest of the data in ref. [8], we conclude that *MRSB* is the most expensive (CPU time requirement) while the cost of the rest varies within a relative small time interval independently of the mesh size. Fig. 9b suggests that *CLO*, *PLO*, *CLE*, and *MRSB* produce equivalent partitionings with respect to interface length criterion. The rest of the heuristic partitionings have higher interface length with *RCM* having the highest. *Greedy* and *Inert* partitionings usually have higher subdomain connectivity than the rest. The *RCM* and *Inert* partitionings have higher IBV than the rest, specially for fine meshes. With respect to bandwidth criterion *RCM* and *Inert* schemes are the worst. Fig. 10 depicts various partitionings of the engine axis mesh and displays the structure of the corresponding FEM matrices for linear triangular elements.

9. Conclusion

In this paper we have proposed a parallel methodology for handling the mesh generation and decomposition preprocessing phases required by domain decomposition based parallel FEM computations. A number of options were considered for the implementation of the decomposition phase. Application specific geometric objects were considered for its evaluation. The preliminary performance data obtained so far suggest that the proposed methodology is capable of speeding up significantly these computations on message passing hardware platforms.



(a) Computation Time

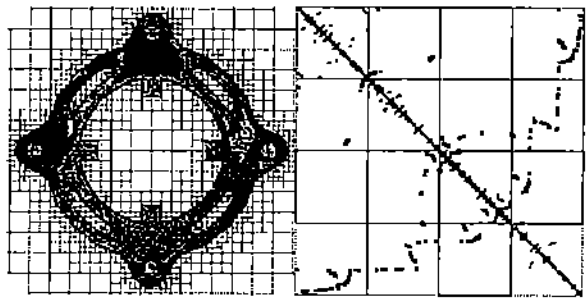
(b) Maximum Interface Length

(c) Subdomain Connectivity

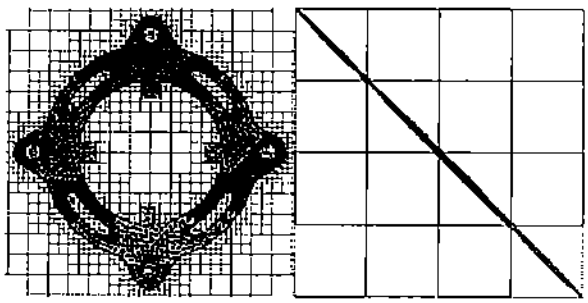
(d) Bandwidth

(e) Interpartitioning Boundary Vertices (IBV)

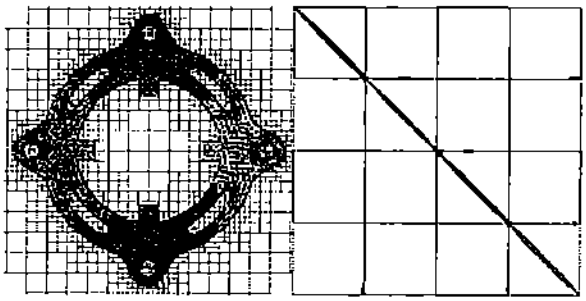
Fig. 9. The performance of seven mesh decomposition heuristics for an engine axis mesh with respect to (a) their execution time requirement and partitioning characteristics such as (b) maximum interface length, (c) subdomain connectivity, (d) bandwidth, and (e) IBV. The data displayed are for the following heuristics: Cartesian (*CLO*) and polar (*PLO*) axis splittings with local optimization, Cartesian with longest expansion (*CLE*), multilevel recursive spectral bisection (*MRSB*), domain-wise BFS (*Greedy*), strip-wise BFS (*RCM*) and inertia axis splitting using first eigenvector (*Inert*).



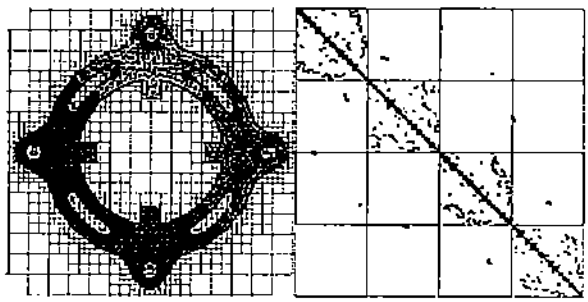
DFS - basic
Communication: 286 / 143 Bandwidth: 39 / 818
Connectivity: 858 IBV: 412



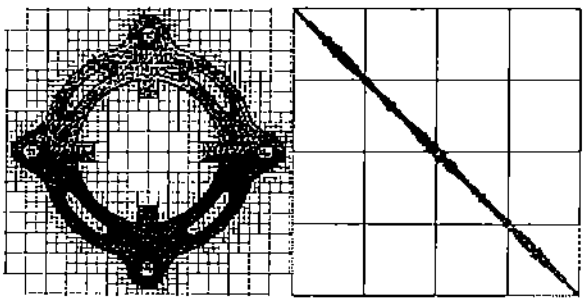
BFS - strip-wise
Communication: 62 / 43 Bandwidth: 55 / 101
Connectivity: 124 IBV: 80



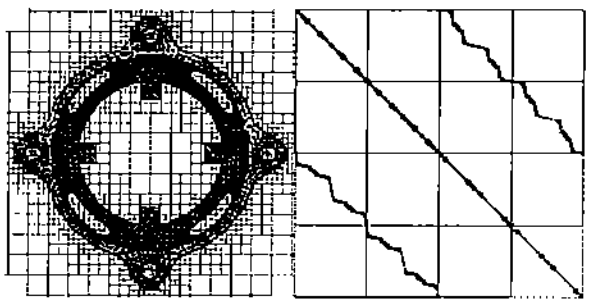
BFS - domain-wise
Communication: 18 / 10 Bandwidth: 34 / 73
Connectivity: 36 IBV: 36



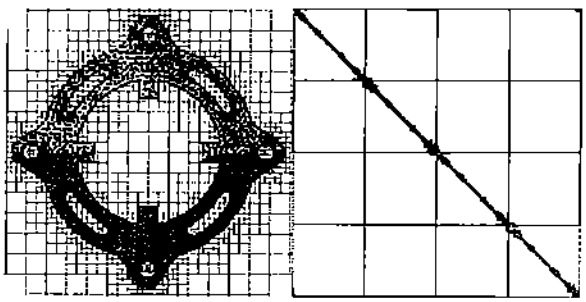
Eigenvector Spectral
Communication: 20 / 10 Bandwidth: 111 / 861
Connectivity: 40 IBV: 36



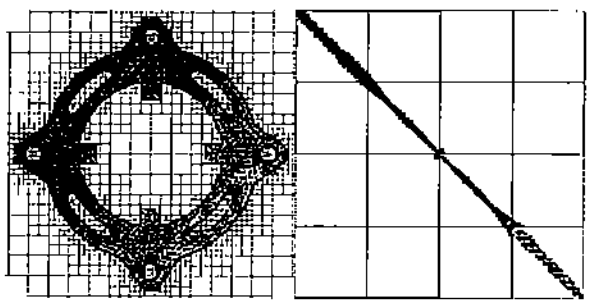
Cartesian - local optimum
Communication: 67 / 48 Bandwidth: 47 / 163
Connectivity: 134 IBV: 85



Polar - recursive bisection
Communication: 131 / 104 Bandwidth: 13 / 57
Connectivity: 387 IBV: 251



Polar - local optimum
Communication: 51 / 27 Bandwidth: 22 / 135
Connectivity: 102 IBV: 100



Inertia - first eigenvector
Communication: 65 / 49 Bandwidth: 54 / 219
Connectivity: 130 IBV: 114

Fig. 10. The 4-way partitionings of engine axis mesh using various splitting schemes and the structure of the corresponding FEM matrix. For each partitioning, the values of the objectives functions of the four partitioning are displayed.

Acknowledgments

This work was supported by NSF grants 9123502-CDA and 9202536-CCR, AFOSR F49620-92-J-0069 and ARPA grant DAAH04-94-G-0010.

References

1. Chrisochoides, N. P.; Houstis, E. N.; Rice, J. R. (1994) Mapping algorithms and software environments for data parallel PDE iterative solvers. Journal of Distributed and Parallel Computing, 21, 75-95

2. Chrisochoides, N. P.; Houstis, E. N.; Houstis, C. E. (1991) Geometry based mapping strategies for PDE computations. Proceedings of the International Conference on Supercomputing, Cologne-Germany, 128-135

3. Houstis, E. N.; Houstis, C. E.; Rice, J. R.,; Weerawarana, S. (1994) PYTHIA: A computationally intelligent paradigm to support smart problem solving environments for PDE based applications. to appear

4. Houstis, E. N.; Rice, J. R. (1992) Parallel ELLPACK: A development and problem solving environment for high performance computing machines. Programming Environments for High-Level Scientific Problem Solving (Gaffney, P. W.; Houstis, E. N., Eds), North-Holland, 229-241

5. Kim, S.; Houstis, E. N.; Rice, J. R. (1994) Parallel stationary iterative methods and their performance. Proceedings of the INTEL supercomputer users group conference (Marinescu, D.; Frost, R., Eds), San Diego, to appear

6. Farhat, C.; Simon, H. D. (1993) TOP/DOMDEC - a software tool for mesh partitioning and parallel processing. Technical Report, NASA Ames Research Center, RNR-93-011, 1-28

7. Tragoudas, S. (1994) Min-cut partitioning on underlying tree and graph structures. to appear

8. Wu, P.; Houstis, E. N. (1993) Parallel dynamic mesh generation and domain decomposition. Technical Report, Purdue University, Department of Computer Sciences, CSD-TR-93-075, 1-49

9. Wu, P.; Houstis, E. N. (1993) Parallel electronic prototyping of physical objects. Technical Report, Purdue University, Department of Computer Sciences, CSD-TR-93-026, 1-48

10. Bykat, A. (1976) Automatic generation of triangular grid: I-Subdivision of a general polygon into convex subregions. II-Triangulation of convex polygons. International Journal for Numerical Methods in Engineering, 10, 1329-1342

11. Khan, A. I.; Topping, B. H. V. (1991) Parallel adaptive mesh generation. Computing Systems in Engineering, 2(1), 75-101

12. Lo, S. H. (1991) Automatic mesh generation and adaptation by using contours. International Journal for Numerical Methods in Engineering, 31, 689-707

13. Sadek, E. A. (1980) A scheme for the automatic generation of triangular finite elements. International Journal for Numerical Methods in Engineering, 15, 1813-1822

14. Cheng, F.; Jaromczyk, J. W.; Lin, J.; Chang, S.; Lu, J. (1989) A parallel mesh generation algorithm based on the vertex label assignment scheme. International Journal for Numerical Methods in Engineering, 28, 1429-1448

15. Samet, H. (1984) The quadtree and related hierarchical data structures. Computing Surveys, 16(2), 187-260

16. Mitchell, W. F. (1987) A comparison of adaptive refinement techniques for elliptic problems. Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science, UIUCDCS-R-87-1375, 1-14

17. PATRAN (1992) A Division of PDA Engineering - PATRAN Plus User Manual, Vol. 1 & 2, PDA Engineering, PATRAN Division

18. Deljouie-Rakhshandeh, K. (1990) An approach to the generation of triangular grids possessing few obtuse triangles. International Journal for Numerical Methods in Engineering, 29, 1299-1321

19. Baase, S. (1988) Graphs and Digraphs, Computer Algorithms: Introduction to Design and Analysis, Reading, MA, Addison-Wesley, Ch.4, pp. 145-207

20. Farhat, C. (1988) A simple and efficient automatic FEM domain decomposer. Computers & Structures, 28(5), 579-602

21. Farhat, C.; Lesoinne, M. (1993) Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. International Journal for Numerical Methods in Engineering, 36, 745-764

22. George, A.; Liu, J. W. H. (1979) An implementation of a pseudoperipheral node finder. ACM Transactions on Mathematical Software, 5(3), 284-295

23. Gibbs, N. E.; Poole, J. W. G.; Stockmeyer, P. K. (1976) An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM J. Numer. Anal., 13(2), 236-250

24. Malone, J. G. (1988) Automated mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computers. Computer Methods in Applied Mechanics and Engineering, 70, 27-58

25. Pissanetsky, S. (1984) Ordering for Gauss elimination: Symmetry matrices, Sparse Matrix Technology, Orlando, Academic Press, Ch.4, pp. 94-158

26. Chan, W. M.; George, A. (1980) A linear time implementation of the reverse Cuthill-McKee algorithm. BIT, 20, 8-14

27. George, A.; Liu, J. W. H. (1978) Algorithms for matrix partitioning and the numerical solution of finite element systems. SIAM J. Numer. Anal., 15(2), 297-327

28. Barnard, S. T.; Simon, H. D. (1993) A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 711-718

29. Fiedler, M. (1975) A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. Czechoslovak Mathematical Journal, 25(100), 619-633

30. Fiedler, M. (1975) Eigenvectors of acyclic matrices. Czechoslovak Mathematical Journal, 25(100), 607-618

31. Fiedler, M. (1973) Algebraic connectivity of graphs. Czechoslovak Mathematical Journal, 23(98), 298-305

32. Hendrickson, B.; Leland, R. (1993) An improved spectral load balancing method. Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 953-961

33. Simon, H. D. (1991) Partitioning of unstructured problems for parallel processing. Computing Systems in Engineering, 2(2/3), 135-148

34. Venkatakrishnan, V.; Simon, H. D. (1992) A MIMD implementation of a parallel Euler solver for unstructured grids. The Journal of Supercomputing, 6, 117-137

35. Loriot, M.; Fezoui, L. (1988) Mesh-splitting preprocessor. unpublished manuscripts

36. Williams, R. (1992) Parallel meshes for complex geometry. PML, 302-312

37. Lohner, R.; Camberos, J.; Merriam, M. (1992) Parallel unstructured grid generation. Computer Methods in Applied Mechanics and Engineering, 95, 343-357

38. Kernighan, B. W.; Lin, S. (1970) An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 49, 291-307

39. Kirkpatrick, S.; Gelatt, J. C. D.; Vecchi, M. P. (1983) Optimization by simulated annealing. Science, 220(4598), 671-680

40. Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; Schevon, C. (1989) Optimization by simulated annealing: An experimental evaluation; Part I, Graph partitioning. Operations Research, 37(6), 865-892

41. Saab, Y. G.; Rao, V. B. (1991) Combinatorial optimization by stochastic evolution. IEEE Transactions on Computer-Aided Design, 10(4), 525-535

42. Hertz, A.; Werra, D. (1987) Using Tabu search techniques for graph coloring. Computing, 39, 345-351

43. Glover, F.; McMillan, C. (1985/6) Interactive decision software and computer graphics for architectural and space planning. Annals of Operations Research, 5, 557-573

44. Savage, J. E.; Wloka, M. G. (1991) Parallelism in graph-partitioning. Journal of Parallel and Distributed Computing, 13, 257-272

45. Geist, G. A.; Heath, M. T.; Peyton, B. W.; Worley, P. H. (1992) A users' guide to PICL, a portable instrumented communication library. Technical Report, Oak Ridge National Laboratory, Oak Ridge, TN, ORNL/TM-11616, 1-22

46. Geist, G. A.; Heath, M. T.; Peyton, B. W.; Worley, P. H. (1990) PICL: a portable instrumented communication library, C reference manual. Technical Report, Oak Ridge National Laboratory, Oak Ridge, TN, ORNL/TM-11130, 1-146

47. Heath, M. T.; Finger, J. E. (1993) ParaGraph: A tool for visualizing performance of parallel programs, Oak Ridge National Laboratory, Oak Ridge, TN, 1-50

48. Heath, M. T. (1993) Recent developments and case studies in performance visualization using ParaGraph. Performance Measurement and Visualization of Parallel Systems (Haring, G.; Kotsis, G., Eds.), Amsterdam, The Netherlands, Elsevier Science Publishers, 175-200

49. Heath, M. T.; Finger, J. E. (1991) Visualizing the performance of parallel programs. IEEE Software, 8(5), 29-39

50. Al-Nasra, M.; Nguyen, D. T. (1991) An algorithm for domain decomposition in finite element analysis. Computers & Structures, 39(3/4), 277-289