Parallel Performance of a Preconditioned CG Solver for Unstructured Finite Element Applications[1]

John N. Shadid[2], Scott A. Hutchinson[2] and Harry K. Moffat[3]

Sandia National Laboratories
Albuquerque, New Mexico 87185

Appears in the Proceedings of the Colorado Conference on Iterative Methods, Breckenridge, CO, April 5 - 9, 1994

---

1. This work was partially funded by the Applied Mathematical Sciences Program, US Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories operated for the US Department of Energy under contract no. DE-ACO4-76DP00789.

2. Parallel Computational Sciences Department

3. Chemical Processing Sciences Department

Parallel Performance of a Preconditioned CG Solver for Unstructured Finite Element Applications[2]

John N. Shadid[2], Scott A. Hutchinson[2] and Harry K. Moffat[3]
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

A parallel unstructured finite element (FE) implementation designed for message passing MIMD machines is described. This implementation employs automated problem partitioning algorithms for load balancing unstructured grids, a distributed sparse matrix representation of the global finite element equations and a parallel conjugate gradient (CG) solver. In this paper a number of issues related to the efficient implementation of parallel unstructured mesh applications are presented. These include the differences between structured and unstructured mesh parallel applications, major communication kernels for unstructured CG solvers, automatic mesh partitioning algorithms, and the influence of mesh partitioning metrics on parallel performance. Initial results are presented for example finite element (FE) heat transfer analysis applications on a 1024 processor nCUBE 2 hypercube. Results indicate over 95% scaled efficiencies are obtained for some large problems despite the required unstructured data communication.

1. Introduction.

In recent years substantial progress has been made in the development of scientific applications on large-scale multiple instruction - multiple data (MIMD) machines. These machines utilize many independent processors, each with local memory, to provide a substantial performance increase over traditional vector supercomputers. Unfortunately, this performance increase is often obtained with considerable cost in programing complexity. In particular, users must now partition computational tasks into subtasks suitable for individual processors, map these tasks to the parallel architecture and then use message passing to achieve synchronization and communication. In general, algorithm development and performance analysis for the parallel solution of PDE systems has been focused on regular meshes [7,10,11]. For this case the static partitioning problem, of dividing the computational task into subtasks, is trivial. Due to the data locality inherent in these finite difference (FD) and finite element (FE) approximation procedures the task partitoning reduces to simple heuristics designed to minimize the perimeter-to-area (or surface -to-volume) ratio for subdomains of regular meshes. Once subdivided these tasks can be easily mapped onto hypercubes and mesh architectures so that neighboring subdomains in the computational domain are mapped to neighboring processors in the parallel machine. Thus, only structured - nearest neighbor communication is needed and contention for the communication channels is avoided.

In contrast, the partitoning and mapping problem for irregular and unstructured meshes can be much more difficult. Indeed, the determination of a partition that actually minimizes communication between balanced sets is know to be an NP-hard problem [3], so it is very unlikely that a general computationally efficient algorithm exists. In addition, the mapping problem for unstructured

2. This work was partially funded by the Applied Mathematical Sciences Program, US Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories operated for the US Department of Energy under contract no. DE-ACO4-76DP00789.

2. Parallel Computational Sciences Department

3. Chemical Processing Sciences Department

meshes is also difficult since a general unstructured mesh cannot be mapped with only nearest neighbor communication on a hypercube or mesh architecture. As a result the required unstructured communication can produce contention for communication channels between processors. However, the central importance of the partitoning and mapping problem for unstructured mesh computations has motivated the use of a wide range of heuristic algorithms [4,6,8,12,13]. These algorithms produce a variety of partitions for which the quality of the partition is roughly correlated with computational cost. It is these essential differences between regular mesh and unstructured mesh parallel computations which make the efficient parallel implementation of unstructured FE codes challenging on distributed memory MIMD machines.

2. Parallel Implementation Overview

In this section the overall parallel implementation framework for a representative parallel unstructured mesh FE application and CG solver is briefly discussed. This FE application is capable of solving for steady and transient conduction with local volumetric heat generation in complex 2D/3D geometries. Considering a typical complex unstructured grid as in Figure 1 the following comments about the typical structure of a parallel FE application can be made. It is evident from Figure 1 that a general automated method for subdividing an unstructured computational mesh and mapping it to the parallel MIMD machine is necessary. An ad-hoc or by-hand method would prove to be unusable for a large number of meshes and the resulting parallel communication efficiency would be difficult to predict, assess and control. In our implementation we have used a general graph and mesh partitioning utility, CHACO[5], developed at Sandia. Using this utility the basic overall parallel MIMD implementation on P processors is as follows. Consistent with the choice of a nodal based FE scheme, it is necessary to load balance the work associated with the matrix setup and solution by partitioning the N nodes among the P processors[3]. The load balancing algorithm partitions the N nodes into P sets of $\lfloor N/P \rfloor$ or $(\lfloor N/P \rfloor + 1)$ FE nodes[4] and then maps these sets to the P processors of the parallel machine such that the overall interprocessor communication is minimized (see Section 4). After the required load balance and mapping the FE application can then set up the distributed FE coefficient matrix. On each processor this distributed sparse matrix corresponds to a rectangular submatrix, $\mathbf{A}_p$, of the global overall coefficient matrix, $\mathbf{A}$. Thus each processor, in parallel, performs the necessary element integrations and sets up a local set of equations for each of the FE nodes which it has been assigned. In this implementation, the equations are fully summed and actually correspond to a complete row entry from the corresponding global coefficient matrix. The union of these distributed rectangular matrices is therefore equivalent to a serial global coefficient matrix.

3. Parallel CG Solver

The parallel CG algorithm is essentially the same as the structured grid Krylov methods discussed in [10,11] with the exception of the unstructured matrix-vector product. The main kernels are the matrix-vector product, DAXPY type operations and vector inner products. As in the structured mesh case, the key to performance in these solvers is typically the efficiency of the matrix-vector multiply kernel and, within this, the communication it requires. For the parallel CG kernel to operate as efficiently as possible, the time for the interprocessor communication during the required matrix-vector multiplies must be minimized. In turn, key to this minimization is the data structures in which both the distributed sparse matrix and the distributed vectors are stored. Logically and as will be described in the next section, each processor is given a set of nodes for which it is responsible. Thus, in the formation of the sparse matrix and vectors, each processor will have a set of rows in both the sparse matrix and any associated vectors, each corresponding to

3. Other choices such as element based schemes are possible. The relative performance of such schemes should not vary greatly for reasonable implementations of either the nodal or elemental schemes.
4. $\lfloor R \rfloor$ is the floor function which returns the largest integer, m, such that $m < R$.

unknowns located at the nodes which have been assigned to this processor by the partitioning algorithm. Formally, let $v_l \in \mathbf{R}^{n_l}$ be the vector of unknowns which have been assigned to processor l where n_l is the number of these unknowns. The edges cut in the connectivity graph represent the data dependencies between subpartitions (processors). Thus, in order to complete the matrix-vector product, processor l will need additional values of $\mathbf{v}$ which reside on other processors. These values are required to complete the interactions between processor *l's* "border" unknowns and it's "external" unknowns. That is, if $\mathbf{v}_{lb} \subset \mathbf{v}_l$ are the border unknowns of processor l, these unknowns interact with border unknowns on neighboring processors via the connectivity of the FEM mesh. These border unknowns assigned to neighboring processors are referred to as processor l's external unknowns. Figure 2 gives an illustration of this partitioning.

Given a specific partition and the assignment of the unknowns (internal, border and external) described above, a distributed sparse matrix storage scheme[10] is used. On each processor, the node numbers are reordered such that the first n_{in} nodes are the internal nodes of the processor, the next n_b are the border nodes and the last n_e are the external nodes. Therefore, n_l is defined as $n_l = n_{in} + n_b$. Thus, the last n_e members of the vector must be obtained by communicating with the processors which have been assigned these nodes. Once this portion of the vector is filled, the matrix-vector product may be computed. Note that the number of messages is equal to the number of neighboring processors and that the size of each message is directly proportional to the number of edges cut in the connectivity graph by interprocessor boundaries. In addition, if the processor is not a physical neighbor, the message may have to traverse several intermediate communication channels in order to arrive at its destination. Thus two processor based local metrics, the maximum number of messages and the maximum size of these messages along with two global network metrics, the total number of data items transmitted (termed cuts) and the total number of data items transmitted weighted by the distance they travel (termed hops) are useful to describe the effectiveness of partitioning algorithms. These factors are critical in determining the speed of the matrix-vector products. All of these factors are influenced to some degree by the partitioning of the problem. The next section briefly describes the partitioning methods used in the results presented in Section 5. For a more complete description of the partitioning methods available through the utility CHACO used in this study, see [5].

4. Partitioning Algorithms

The central importance of the partitoning problem to unstructured mesh computations has motivated the use of a wide range of heuristic algorithms [4,6,8,10,13]. These algorithms produce a wide spectrum of partitions for which the quality of the partition is roughly correlated with computational cost. The partitioning utility CHACO[5] implements a wide range of methods which can be used to produce load balanced partitions for distributed memory MIMD machines. These methods may be divided into two classifications geometric and graph based. Geometric methods use only information about the relative geometric distribution of vertices (or elements) of a mesh that is being partitioned. Graph based methods require only information on the inter-connectivity of the vertices (or elements) to produce a partition of the graph. Graph and geometric partitioners can further be subdivided into local and global methods to describe the scope of influence. Global methods use information from the entire graph (or geometric domain) to select the partition, whereas local methods consider only a small neighborhood of the vertex under consideration to produce the next candidate partition. In this study the following methods have been used. The *linear* and *scattered* partitioners use a simple assignment of the vertices (FE node in this study) to locally balance and map the vertices. The *linear* partitioner assigns groups of $\lfloor N/P \rfloor$ or $(\lfloor N/P \rfloor + 1)$ FE nodes to each processor in a sequential manner in accord with the numbering of the original graph. The *scattered* method assigns each vertex sequentially to a processor, after P vertices are assigned the method repeats cyclically until all vertices are assigned. The run time of the simple schemes is very small and the effectiveness of the partition depends on the original numbering of the graph. The *Inertial* method is a global geometric method which considers the vertices to be point masses with mass equal to a vertex weight. The principle axis of the distributed point masses is determined and a cut is made perpendicular to this direction to bisect the domain into two sets of equal mass. The *inertial* method is fast and can produce a very reasonable partition for a wide range of geometrically based problems. The global inertial method can be combined with a local graph based heuristic, Kernighan-Lin (KL), a greedy local optimization scheme, to produce a hybrid method. The KL local refinement scheme can significantly improve inertial method partitions, it does however increase the cost of this method substantially. The *multilevel spectral bisection* scheme (denoted by ML) is a global graph based heuristic. The ML method of Hendrickson and Leland[4,5] produces a coarse representation of the original graph, to which spectral partitioning is applied on the coarsest level, this partition is projected to the next finer level on which a refinement by KL is used to optimize the partition locally. This local refinement is done recursively up to the finest level. This algorithm is more expensive but usually produces the best partition for large problems. All of these methods are used as a preprocessing step in the parallel FEM implementation so that the individual cost of any partitioning method will be amortized over the number of times a specific parallel partition of a FE mesh is used.

5. Results

As described above there are a number of issues related to the performance of the parallel unstructured finite element application and in this section a number of these are briefly discussed. Figure 3 shows the fixed-size problem speedup for a small 3D unstructured FE problem with about 6000 FE nodes. This plot illustrates the transition from a computation to a communication bound application as the number of processors is increased. Clearly the matrix setup phase of the application scales well with increasing number of processors. The small deviation from linear speedup is due to a small amount of work that is redundantly done in parallel to obtain elemental Jacobians. The total time to setup and solve the problem is also shown in Figure 3. The point at which communication begins to overwhelm the computational cost is at about 256 processors. A transition point at 128 to 256 processors for a small problem indicates that the application scales reasonable well as the number of processors is increased despite the unstructured communication overhead. The effects of the various partitioning metrics for a small problem is shown in Table 1. Here it is evident that the local processor-based metrics of “maximum neighbors” and “maximum message size” are important in the overall performance of the unstructured solvers. The partitioner that produces the best run time is associated with a minimum in the “maximum neighbors” metric (Inertial). When two partitions tie with this metric (ML and Inertial + KL) the “maximum message size” metric becomes appropriate. Clearly the network based “cuts” and “hops” metrics have very little bearing due to the small size of the problem and the small number of processors.

However, distributed memory machines are most productive solving large problems. Tables 2 and 3 indicate the preliminary performance of the parallel application and solvers on two large test problems. In Table 3 the scaled parallel efficiency of the parallel unstructured solver is indicated. The scaled speedup (and thus efficiency) is estimated by using the single node maximum computational rate (Mflops) and the measured multiprocessor rate on two large FE problems. For these large problems it is evident that the network based metric of “hops” is an appropriate measure of expected performance. In terms of the time for the unstructured communication it is clear that having a good quality partition can reduce the required communication time substantially. Scaled efficiencies of over 95% indicate that the parallel solvers scale well for large problems on a large number of processors. Table 4 indicates performance for a 1,000,000 FE node problem run on 512 and 1024 processors. As indicated in the table the ML partitioner produces the highest performing partition. Again in this case the “hops” metric is a good predictor of the expected performance. The minimum time necessary to setup and solve the 1,000,000 node problem, 197 sec., is impressive. The overall performance can also be measure by the solver Mflop rate of 840 Mflops. This compares well to a 1,000,000 unknown finite difference calculation on the nCUBE2 that obtains about 1.4 Gflops. These preliminary results are encouraging.

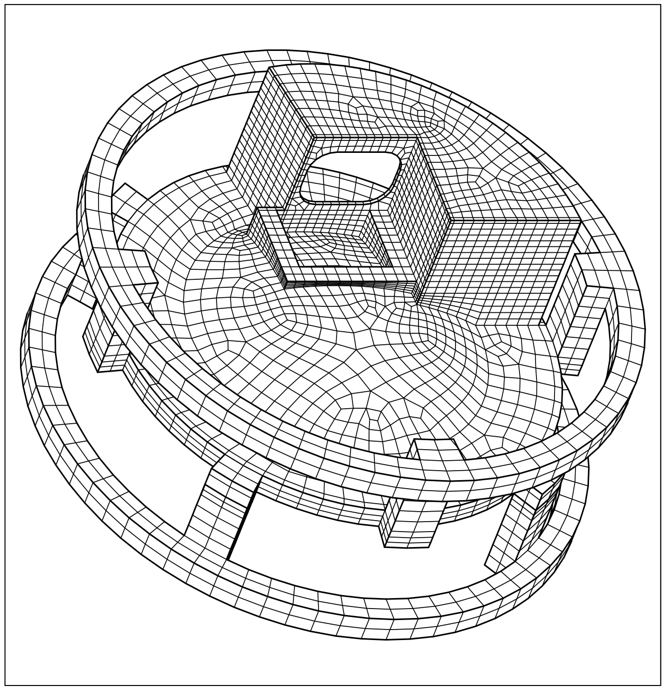

Figure 1: Coarse Investment Casting Mesh

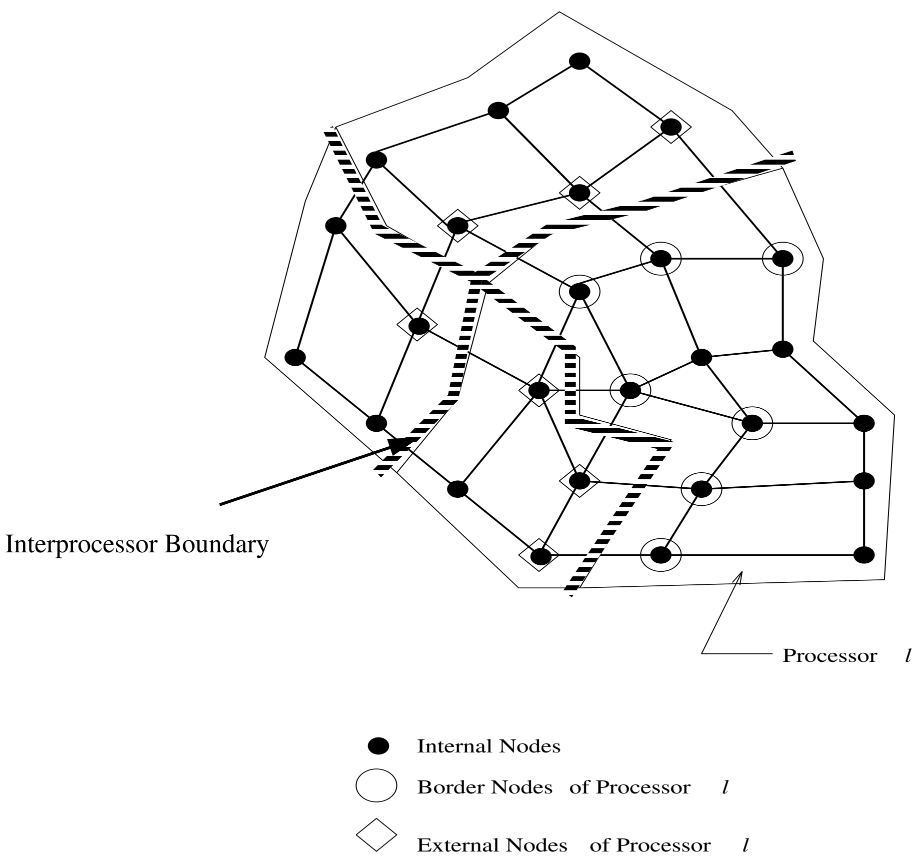

Figure 2: Unstructured Mesh Partitioning Illustrating the

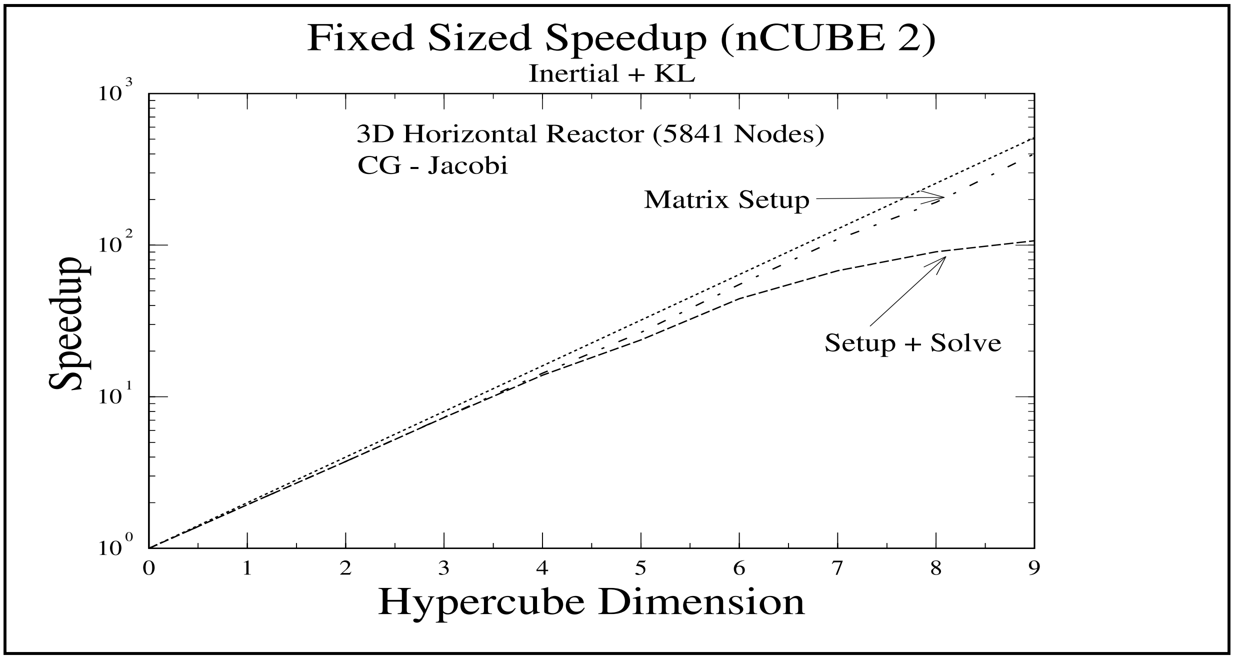

Figure 3: Fixed Sized Speedup for Coarse 3D Horizontal CVD Reactor Mesh

Algorithm	Cuts (10^3)	Hops (10^3)	Max. Neighbors	Max. Mesg. Length (Kbytes)	Partition Time (sec.)	Solve Time (sec.)
Scattered	51	88	7	6.6	0.0	26.9
Linear	7.3	13	7	2.3	0.0	21.2
Inertial	4.7	5.6	5	1.3	0.1	17.9
Inertial +KL	3.0	4.2	6	.46	2.1	18.1
ML	2.6	3.6	6	.70	3.0	19.7

Table 1: Comparison of Partition Metrics for Investment Casting Mesh -8 processors (6,673 FE Nodes)

	64 Processors (186,381 FE Nodes)[1]				256 Processors (1,088,019 FE Nodes			
Algorithm	Hops (10^6)	t_P	t_{UC}	Solver Efficiency	Hops (10^6)	t_P	t_{UC}	Solv Efficie
Linear	3.2	0.0	24.3	.69	--	--	---	--
Inertial	0.62	7.0	7.7	.87	7.6	58	101	.80
Inertial + KL	0.47	84.1	4.7	.90	4.7	688	54	.88
ML	0.27	373.1	2.2	.96	1.3	~3100	13	.97

Table 2: Scaled Parallel Efficiency for 3D Horizontal CVD Reactor Mesh

s is the largest problem per processor (2912 FE Nodes/Proc) that can run with the Linear partition
ıable to run due to increased message buffer size
Partitioning Time (sec.) on a SGI ONYX with R4000 100/50 MHz MIPS cpu and 256 Mb memory
· Unstructured communication time (sec.)

	512 Processors				1024 Processors			
Algorithm	Hops (10^6)	t_{UC}	t_S	Mflops	Hops (10^6)	t_{UC}	t_S	Mflops
Linear	-	-	--	--	41.1	145	397	371
Inertial	10.5	71	460	356	13.9	48	261	615
Inertial + KL	6.3	34.2	403	408	7.4	26	218	740
ML		12	365	451		11	197	840

Table 3: Parallel Performance for 3D Horizontal CVD Reactor Mesh (1,088,019 FE Nodes)

Jnable to run due to increased message buffer size
- Unstructured communication time (sec.)
Total solution time (matrix setup + CG Jacobi solver) (sec.)

References:

1. S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, “A taxonomy of Conjugate gradient methods”, SIAM J. Numer. Anal, Vol. 27, No 6, 1542-1568, (1990)
2. R. W. Freund, G. H. Golub, N. M. Nachtuigal, “Iterative solution of linear systems”, Acta Numerica, 57-100, (1991)
3. M. Garey, D. Johnson, and L. Stockmeyer, “Some simplified NP-complete graph problems”, Theoretical Computer Science, 1, 619-633, (1976)
4. B. Hendrickson and R. Leland, “An improved Spectral Graph partitioning algorithm for mapping parallel computations”, Technical Report SAND92-1460, Sandia National Laboratories, Albuquerque, NM, (1992)
5. B. Hendrickson and R. Leland, “A user’s guide to the graph partitioning code Chaco”, Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque, NM, (1993)
6. S. Hammond, “Mapping unstructured grid computations to massively parallel computers”, Ph. D. thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY, (1992)
7. M. T. Jones and P. E. Plassmann, “The efficient parallel iterative solution of large sparse linear systems”, Technical Report, Argonne National Laboratory, June 1992
8. B. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs”, Bell System Technical Journal, 29, 291-307, (1970)
9. Y. Saad, “Krylov subspace methods on supercomputers”, SIAM J. Sci. Stat. Comput., Vol 10, No. 6, 1200-1232, (1989)
10. J. N. Shadid and R. S. Tuminaro, “Sparse iterative algorithm software for large-scale MIMD machines: an initial discussion and implementation”, Concurrency: Practice and Experience, 4(6), 481-497, (1992)
11. J. N. Shadid and R. S. Tuminaro, “A comparison of preconditioned nonsymmetric Krylov methods on a large-scale MIMD machine”, SIAM J. Sci. Comput., Vol. 15, No. 2, pp 440-459, March 1994
12. H. Simon, “Partitioning of unstructured problems for parallel processing”, in Proc. Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergammon Press, (1991)
13. R. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency, 3, 457-481, (1991)