Scalability Analysis of Partitioning Strategies for Finite Element Graphs.*

Grama Y. Ananth and Vipin Kumar
Department of Computer Science,
University of Minnesota
Minneapolis, MN 55455

April 3, 1992

Corresponding Author: Vipin Kumar
kumar@cs.umn.edu
Tel: 612-624-8023
Fax: 612-625-0572
Presenting Author: Ananth Y. Grama
ananth@cs.umn.edu

Abstract

Issues of partitioning Finite Element Graphs are central to parallel formulations of the Finite Element Method. Due to the nature of the problem, optimal partitioning schemes should conform to three basic criteria: equal load on all processors; locality of communication; and maximum computation to communication ratio associated with each partition. Many techniques have been presented in literature which achieve these to different degrees. This paper presents scalability analysis of three partitioning strategies, namely, striped partitioning, binary decomposition, and scattered decomposition. The analysis is performed using the Isoefficiency metric, which helps in predicting performance of these schemes on a range of processors and architectures.

*This work was supported by Army Research Office grant # 28408-MA-SDI to the University of Minnesota and by the Army High Performance Computing Research Center at the University of Minnesota.

1 Introduction

Availability of large scale commercial parallel computers has created a strong interest in parallel techniques for implementing the Finite Element Method (FEM). Parallel formulations of finite element techniques require a mapping of the elements onto processors, and the performance of the overall formulation is a very sensitive function of this mapping. Any mapping of elements to processors must try to satisfy the following criteria:

1. The ratio of computation to communication associated with each partition should be maximized. This translates to the condition that area to perimeter ratio of the partitions should be as large as possible.

2. Locality of communication should be preserved.

3. The computational load should be balanced to the extent possible.

4. The computational cost of partitioning the mesh should not be excessive.

These conditions represent the classical communication - load imbalance tradeoffs. Optimizing one of these criterion leads to a deterioration with respect to one or more of the other criteria. The mapping problem in its optimal form is known to be NP complete even for simple models of computation and communication costs [7]. Hence, a number of heuristic approaches have been presented to derive reasonable suboptimal partitions in a reasonable amount of time [1, 16, 19, 25, 18]. All of these try to balance the various tradeoffs mentioned. Most of these schemes have been evaluated only on specific parallel computers for certain problems, and from the available experimental results, it is difficult to ascertain relative merits of these schemes. The reason is that the performance of these schemes may be impacted quite differently by changes in hardware characteristics (such as the interconnection network, CPU speed, speed of communication channels etc.), number of processors and problem characteristics (number of elements, mesh geometry). Hence, any conclusions drawn on a set of experimental results may become invalid by changes in any one of the above parameters. Scalability analysis is very useful in extrapolating these conclusions [8, 20, 23, 24]. The isoefficiency metric has been found to be quite useful in characterizing the scalability of a number of algorithms [9, 14, 17, 21, 23, 24]. In particular, it has helped determine optimal load balancing techniques used in tree search based algorithms on various architectures [8, 11, 12, 15].

In this paper, we perform scalability analysis, using the Isoefficiency metric [13, 5, 14], of three partitioning algorithms, namely, striped partitioning, binary decomposition, and scattered decomposition. This helps us determine the relative performance of these schemes over a range of processors, and the effect of communication related parameters on the performance of these schemes. We also relate the performance of each of these schemes to the various problem characteristics such as mesh geometry and density. Our theoretical results are verified through simulations.

In Section 2 we review the parallel formulations of the FEM. Section 3 presents the various definitions and assumptions. Section 4 presents a brief overview of the Iso-Efficiency metric. Section 5 presents the scalability analysis of different partitioning schemes. Section 6 presents experimental results and conclusions are drawn in section 7.

2 Formulation of the Finite Element Method.

This section provides a brief overview of the finite element method. Section 2.1 reviews FEM and Section 2.2 provides a computational abstraction of FEM. Readers not conversant with FEM can skip section 2.1 and move to section 2.2.

2.1 Review of the Finite Element Method.

The finite element method for solving differential equations works by reducing the governing differential equation into a variational form over a discretized domain. The solution to the system is assumed to be represented by an n-dimensional space of basis functions where n is the number of grid points in the mesh [6]. Thus, if ϕ_i is the vector corresponding to each node, the overall approximate solution would be given by $\sum_{i=1}^{n} \phi_i \eta_i$, where η_i are the basis functions. (These have the property that $\eta_i = 0$ for all points except i). Applying the variational principle yields a system of equations $K\Phi = F$ (K is the stiffness matrix and the force vector F is said to contain the driving terms). This system can then be solved for the overall solution Φ. Computing the elements of the stiffness matrix involves the computation of an integral over the domain for the basis functions. The force vector is computed through assembly of element contributions and involves summing weighted contributions of potentials for all nodal points at which specified boundary conditions apply.

It must be noted that the computation of the stiffness matrix and the force vector is relatively inexpensive compared to the solution of this system of equations. Deriving the variational form is also fairly simple to parallelize since the stiffness matrix and force vector can both be computed over different elements without any communication overheads.

The solution to the variational form typically uses iterative solvers, the most popular of them are gradient method solvers. Gradient methods work by iteratively computing solutions in such a way as to minimize the error between iterated and actual solutions. A special class of gradient solvers, called the Conjugate Gradient, proceeds by choosing the descent direction vectors which are mutually orthogonal relative to the inner product weighted by the stiffness matrix [6].

2.2 Review of Parallel Formulations.

The use of iterative techniques (as mentioned above), requires computation of the product of the stiffness matrix with an n dimensional vector. This product does not require an assembly of the

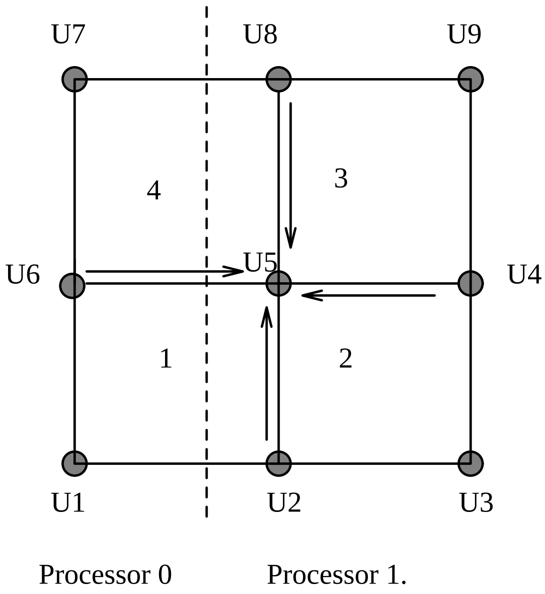

Figure 1: Illustration of computation of element vectors.

stiffness matrix since element contributions to this product can be computed independently and then summed. Consequently, each such iteration step requires computing a product at each node and taking a sum of the various element contributions. This iteration can be executed on a parallel machine by partitioning and assigning elements to various processors. For each element, a processor will need to do a fixed amount of computation followed by a fixed amount of communication with processors that are assigned adjoining elements.

We illustrate the computation involved in figure 1. The weight ($\phi_i s$) of the basis function for each node can be computed iteratively by solving the variational form of the differential equation (which is essentially a sparse linear system). In each iteration, the value of ϕ_i at any point is computed using its own value and weighted values of $\phi_i s$ at neighboring points. For instance, for a five point template, the value at node U5 can be computed from corresponding values at nodes U2, U4, U6 and U8. Now assume that the mesh was partitioned along the dotted line between processors 0 and 1. In computing the values for nodes U1, U6 and U7, processor 0 would need the weighted values of $\phi_i s$ at nodes U2, U5 and U8. Similarly, in computing $\phi_i s$ at nodes U2, U3, U4, U5, U8 and U9, processor 1 would need corresponding weighted values at nodes U1, U6 and U7. Thus, the number of nodes lying on the periphery of the partition correspond to the amount of communication required and the total number of nodes lying in a partition correspond to the amount of computation (calculating the value of ϕ_i for each node).

A number of schemes have been devised to derive reasonable partitions for finite element meshes. Next, we briefly review the three partitioning schemes that are analyzed later. Here, we refer to the four optimality criteria mentioned in the introduction to elucidate the working of these schemes.

Striped partitioning Striped partitioning technique maintains nearest neighbor communication and allocates equal number of elements to every processor. It partitions the mesh into "strips" such that the computation associated with any strip requires communication only with the two neighboring strips. It starts at one corner of the mesh and counts elements along the smaller dimension until it gathers the number of elements that should be assigned to one processor. When all the elements along a row (or column , whichever corresponds to the smaller dimension) have been selected, it continues the count on the next row. In this way the whole graph is partitioned so that equal elements are assigned to each processor. This technique requires that the number of processors be limited such that the number of elements assigned to one processor is large enough to cover a single strip across the whole mesh. This technique thus optimizes criteria 2 and 3 at the expense of 1.

Scattered Decomposition Scattered Decomposition (also referred to as modular mapping) is a technique which has been applied extensively to decompose highly irregular domains [6, 22, 18]. This method tries to balance load by partitioning the domain into a large number of rectangular clusters (r) such that $r >> P$ (number of processors). Each processor is now responsible for processing a disjoint set of r/P such clusters. It would be evident that for irregular problems, by increasing r and consequently decreasing the area of each partition, we can attain a better load balance. However, this is done at the cost of increased communication overheads in intercluster communication for adjoining clusters that do not lie on the same processor. Detailed discussion on this technique is presented in [22, 4, 18].

Binary Decomposition Berger and Bokhari [3] propose the use of binary decomposition to partition irregular domains. In this technique, the domain is partitioned into two parts recursively so that the number of elements in each part is identical. This technique ensures a very good load balance. However, the communication characteristics of the partitions are a function of the mesh geometry. These can be controlled to a certain extent by alternating the orientation of the partitioning cut. This technique optimizes criterion 3 and allows a tradeoff between criteria 1 and 2.

There have been several other techniques that have also been proposed in literature. Sadayappan et al. [19] propose a 2-D decomposition scheme with boundary refinement. This technique works in two steps, first the mesh is partitioned into pieces which might have unequal load in terms of nodes, but the communication involved is close to optimal. In the second phase, the boundaries are refined in such a way that the load on each processor is also equal, though in doing so, we might be causing extra communication overhead. Chung and Ranka [25] propose two other schemes - 2-way striping and Greedy assignment. 2-way striping also works in two phases where the first phase optimizes communication and the second phase tries to achieve load balance using specific load transfer heuristics. Greedy assignment uses information regarding assignment of neighboring nodes to assign a particular node to a processor.

3 Definitions and Assumptions.

Before analyzing the scalability of partitioning and mapping schemes described, we define the following variables :

N : Number of elements in the finite element mesh.

P : Number of processors.

k : Number of elements per processor. (ideally N/P, for perfect load balance.).

t_b : The time for transferring the number of bytes that are required to be communicated in one dimension by one grid point from one processor to another, at a unit distance from the source. (**Note**: the time for m items to be communicated over a store-and-forward network between any two processors of a hypercube is given by $t_s + m \times t_b \log P$. The analysis presented in this paper assumes an asymptotic case, and consequently, ignores the startup time t_s. Analysis without ignoring the startup time also follows similarly. We do consider t_s in our experiments and use it as a parameter in performance evaluation. A similar analysis can also be performed for networks with cut-through routing, where the message passing time for the above example is given by $t_s + t_h \log P + mt_b$ where t_h is the per-hop transfer time).

t_c : The time for processing one single element to derive the residual equations for the element.

T_p : The parallel execution time.

T_e : Sequential execution time. On a serial processor, each iteration of the algorithm takes time $K_1 \times b \times N$, where K_1 is a constant resulting from individual time for multiplication and summation, and b is the bandwidth of the matrix. Thus the sequential time for the algorithm is $T_e = K_1 \times b \times N$.

T_o : Total overhead incurred in parallel formulation. (This represents the sum of time spent by all processors in communication, idling due to load imbalance and synchronization).

S : Speedup (This is defined as the ratio of the sequential to the parallel execution time $S = T_e/T_p$).

E : Efficiency (This is defined as the ratio of speedup achieved to the number of processors used $E = S/P$).

All analysis presented in this paper is for a hypercube connected multicomputer unless otherwise mentioned. The finite element mesh under consideration is a 2-D mesh with quadrilateral elements. The meshes under consideration are static in time (non-adaptive meshes). For a given problem (system of equations and domain), the problem size is increased by making the mesh denser uniformly, *i.e.*, the number of elements in different parts of the mesh are all scaled up by the same factor. The analysis is performed for only one iteration. Since the mesh is static, relative scalability behavior of these schemes remains unchanged for any number of iterations.

4 The Isoefficiency Metric For Scalability Analysis.

If a parallel algorithm is used to solve a given problem instance of a fixed size (i.e., T_e), then the efficiency decreases as P increases. This property is true of all parallel algorithms. For a large class of parallel algorithms (e.g., parallel DFS [12], parallel shortest path algorithms the following additional property is also true:

- For any given number P of processors, the efficiency of the parallel algorithm goes up monotonically (i.e., it never goes down, and approaches a constant e, such that $0 < e \leq 1$) if it is used to solve problem instances of increasing size.

We call such algorithms *scalable* parallel algorithms. In these algorithms, efficiency can be maintained at a desired value (between 0 and e) for increasing number of processors provided that the problem size is also increased. Note that for a given parallel architecture and different algorithms, the problem size may have to increase at different rates (as a function of the number of processors) in order to maintain constant efficiency. The rate of growth of the problem size, with respect to the number of processors, that is required to keep the efficiency fixed essentially determines the degree of scalability of these parallel algorithms (for a specific architecture). For example, if the problem size is required to grow exponentially as a function of the number of processors, then the algorithm-architecture combination has a poor scalability, as it would be difficult to obtain good speedups for a large number of processors (unless the problem size being solved is enormously large). On the other hand, if the problem size needs to grow only linearly with respect to the number of processors then the parallel algorithm is highly scalable; i.e., it can deliver linearly increasing performance with increasing number of processors for reasonable problem sizes. If the problem size needs to grow as $f(P)$, where P is the number of processors, to maintain an efficiency E, then $f(P)$ is defined to be the **isoefficiency function** for efficiency E and the plot of $f(P)$ as a function of P is defined to be the *isoefficiency curve* for efficiency E. It is possible to have different isoefficiency functions for different efficiencies. In this paper, isoefficiency functions are the same for all values of E within some range $0 < E \leq e \leq 1$. Therefore, for the sake of convenience in the rest of the paper, we will refer to isoefficiency functions and isoefficiency curves without reference to a specific efficiency.

Since $E = 1/(1+\frac{T_o}{T_e})$, where T_o represents the total overhead, in order to maintain a constant efficiency, $T_e \propto T_o$ or $T_e = KT_o$ must be satisfied, where $K = E/(1-E)$ is a constant depending on the efficiency to be maintained. Once T_e and T_o are known as functions of n and p, the isoefficiency function can often be determined by simple algebraic manipulations.

5 Scalability analysis of selected techniques.

5.1 Striped Partitioning.

This scheme, as we have seen earlier, maintains good load balance, and preserves nearest neighbor communication. The biggest drawback of this partitioning strategy is that the area to perimeter ratio of the partitions can be very low and consequently the communication overheads can be very high. Also by forcing N/P (number of elements assigned to each processor) to be more than the smaller dimension of the mesh, we are essentially limiting the amount of parallelism that we can extract from the problem to $O(\sqrt{N})$. Hence, $P \sim O(\sqrt{N})$ and $N \sim O(P^2)$ due to concurrency. (The dimension of an irregular mesh is defined by the dimensions of the smallest rectangle that can contain the given mesh.) Let $q \times u$ be the dimensions of the mesh (in terms of number of elements), where q is the smaller dimension. Since we scale up the problem by making the mesh denser uniformly, we can make the following observations: a linear scaling of the problem leads to a linear scaling in the number of points in the mesh, if the points in the mesh are scaled up by a factor of g, the maximum points in each dimension are scaled up by a factor of $O(\sqrt{g})$, the ratio of the number of points in either dimension remains a constant irrespective of scaling factor. From these, we can see that $q/u = c$ is a constant, and since $q * u \sim N$, both dimensions can be expressed as $O(\sqrt{N})$. Thus, we can express the smaller dimension q as $q = \beta N$ where β is a constant for a given mesh.

For striping, the total amount of communication required by a partition in the worst case is given by $t_b \times 2q$.

$$T_o = 2t_b \times \sqrt{\beta \times N}$$

The parallel runtime is given by:

$$T_p = 2t_b \times \sqrt{\beta \times N} + K_1 \times b \times N/P + K_2 \log P.$$

Where K_1 and K_2 are constants and the last term results from overhead due to synchronization after each iteration. Assuming that $N/P >> \log P$ (a claim which will be validated later), we can ignore the third term in the expression for parallel time.

$$\text{Speedup} = T_e/T_p = \frac{K_1 \times b \times N}{2t_b \times \sqrt{\beta N} + K_1 \times b \times N/P}$$

$$= \frac{1}{1/P + (2t_b \times \sqrt{\beta N})/(K_1 \times b \times N)}$$

$$\text{Efficiency} = \text{Speedup/P} = \frac{1}{1 + P(2t_b \times \sqrt{\beta N}))/(K_1 \times b \times N)}$$

It can be seen from this expression that if P is fixed and N is increased, then the efficiency of the system goes up because the communication overhead becomes relatively smaller compared to useful work. If N is fixed and P is increased, the efficiency goes down. Consequently, if an increase in P accompanied by a compensatory increase in N, the efficiency may remain constant. This rate of increase of N to maintain a constant E is the isoefficiency of the system.

For Isoefficiency, at a constant efficiency E,

$$\frac{(P \times 2t_b \times \sqrt{\beta N})}{K_1 \times b \times N} = \frac{1-E}{E}.$$

Collecting the constants together, we get

$$K_1 \times b \times N = (\frac{E}{1-E})^2 \times \frac{4t_b^2 \times \beta}{K_1 \times b} \times P^2$$

or

$$T_e = K' \times P^2$$

(where K' is a constant. This expression also validates our claim that N/P would need to be higher than $\log P$ for constant efficiency) Hence, striped partitioning has an Isoefficiency of $O(P^2)$ due to both communication overheads and concurrency limitations.

5.2 Binary Decomposition.

The binary decomposition technique can have several variants based on the extent to which the communication requirements (volume of communication) and communication distances are optimized. We define two cases based on these criteria and analyze the performance in both cases.

5.2.1 Optimal communication volume, suboptimal communication distances.

The communication volume for binary decomposition is optimized when all the partitions become squares. This minimizes the periphery of each partition. Note that all the partitions must be squares since due to synchronization at each iteration, a single non-optimized partition could cause a bottleneck. This would typically result in the case of very regular finite element graphs. Further, we assume in this case that it is possible that some partition assigned to a processor would require communication with a partition assigned to another processor which can be $\log P$ steps away.

Clearly, the total amount of communication required per processor in this case is given by $4t_b \times \sqrt{N/P} \times \log P$ per iteration and parallel time would be given by

$$T_p = 4t_b \times \sqrt{N/P} \times \log P + K_1 \times \mathrm{b} \times N/P$$

$$\text{Speedup} = T_e/T_p$$

$$= \frac{K_1 \times b \times N}{4t_b \times \sqrt{N/P} \times \log P + K_1 \times b \times N/P}$$

$$= \frac{1}{1/P + (4t_b \times \sqrt{N/P} \times \log P)/(K_1 \times b \times N)}$$

$$\text{Efficiency} = \text{Speedup/P} = \frac{1}{1 + P \times (4t_b \times \sqrt{N/P} \times \log P)/(K_1 \times b \times N)}$$

For Isoefficiency, at a constant efficiency E,

$$\frac{P \times 4t_b \times \sqrt{N/P} \times \log P}{K_1 \times b \times N} = \frac{1-E}{E}.$$

Balancing the numerator term for isoefficiency and collecting the constants together, we get

$$K_1 \times \mathrm{b} \times \mathrm{N} = \left(\frac{E}{1-E}\right)^2 \times \frac{16t_b^2}{K_1 \times b} \times \mathrm{P} \times \log^2 P.$$

or

$$T_e = \mathrm{K} \times \mathrm{P} \times \log^2 P.$$

Therefore in the best case the isoefficiency of this scheme is O(P$\times\log^2 P$).

5.2.2 Non-optimal communication volumes and distances.

This represents the worst case situation for binary decomposition. Here, we assume that partitions may not be perfect squares and a processor may require communication with another processor $\log P$ steps away. While analyzing striped partitioning, we had shown that, given our scaling technique, we could express the dimensions of the graphs as $O(\sqrt{N})$. Thus, the larger dimension u of the mesh can be expressed as $\alpha\sqrt{N}$ where α is constant for a given mesh. If the orientation of partitioning is alternated, this dimension will get partitioned $\log P/2$ times. Thus the largest communication boundary would be $\alpha\sqrt{N} \times \frac{2}{\log P}$. Since each communication can be with a processor $\log P$ steps away, the total communication overhead is given by

$$T_o = 4t_b\alpha\sqrt{N} \times \frac{2}{\log P} \times \log P.$$

Equating this with essential computation, we get

$$\frac{t_c N}{P} \sim 4t_b\alpha\sqrt{N} \times \frac{2}{\log P} \times \log P$$

$$\sqrt{N} t_c \sim 8t_b\alpha P$$

$$N t_c \sim \frac{64t_b^2\alpha^2}{t_c} P^2$$

Thus, in the worst case, the isoefficiency function for binary decomposition is given by $O(P^2)$ with a inverse dependence on t_c and a square dependence on t_b.

5.3 Scattered Decomposition.

In the analysis for Scattered Decomposition, we use results from various related analyses. We briefly review these results first.

5.3.1 Related Analyses.

There have been various attempts to characterize the performance of scattered decomposition in the context of various problems. Nichol and Saltz [4] use a probabilistic model of workload in one dimension to formally explain the working of scattered decomposition. Assuming correlation for workload to be a convex function of distance, they show that increasing the degree of scattered decomposition does not increase the common processor workload variance. Also, if the workload process is stationary Gaussian and the correlation function decreases linearly with distance until it becomes zero and then stays there, increasing the degree of scattered decomposition reduces the expected maximum processor workload until a certain point, after which the expected maximum processor workload remains constant. Increasing the degree of scattered decomposition however comes at the expense of increased communication since the perimeter of the clusters, which is a measure of the required communication, increases with respect to the area of the clusters. Improved load balance is thus achieved at the expense of increased communication overheads.

Kruskal and Weiss [10] analyze the problem of allocating independent subtasks to processors. Though this work does not directly apply to the problem of analyzing scattered decomposition, we will show how this can be used to gain an estimate of the overhead incurred in balancing load using this technique. Under the assumption that the running times of all subtasks are independent, identically distributed increasing failure rate random variables, they derive the expected time of completion on P processors. They show that allocating an equal number of subtasks to each processor all at once yields good efficiency. We shall see that this result is of particular significance while analyzing the performance of scattered decomposition.

For a given finite element mesh, decomposing the domain does not yield independent subtasks. There will be a correlation between the workloads associated with each cluster. This correlation will be a function of the distance between the two clusters. The total processor workload is a sum of the cluster workloads over all the clusters that are assigned to a processor. For modular scattered decomposition, the correlation between clusters is used to balance load by assigning adjacent clusters (which have strongly correlated loads) to different processors. However, if we modify the assignment so as to assign clusters randomly to processors, we can approximate the workload on a processor by an independent random variable. Clearly, modular scattered decomposition would perform better than random scatter decomposition since the former uses the correlation among subtasks and the latter neglects it. Kruskal's analysis can be applied to random scattered decomposition and the expected timings can be used as an upper bound on the expected time for modular scattered decomposition. In fact, it turns out that Kruskal's analysis is able to predict the performance of modular scattered decomposition fairly accurately.

Kruskal's results state that if there were r independent subtasks with mean μ_r and standard deviation σ and if we were assigning them so that every processor gets r/P tasks at one time, the expected time of completion is given by

$$T_p \sim \tfrac{r}{P}\mu_r + \sigma\sqrt{2\tfrac{r}{P}\log P}$$

where r is large compared to $P \log P$. In this expression, the first term represents the actual work performed on P processors, and the second term represents overhead due to load imbalance. It is clear that as we increase r, the first term grows linearly, whereas the second term grows as $\sqrt{r}$. Thus on increasing r, essential computation grows faster than the overhead and consequently, the efficiency of the system increases.

5.3.2 Scalability Analysis of Scattered Decomposition

We can now use the above results to analyze the performance of scattered decomposition. We are given a domain of N points. The domain is divided into r equal clusters. Let $\mu_r t_c$ be the mean computation associated with each cluster and σ be its standard deviation. Thus, $r\mu_r = N$. Applying Kruskal and Weiss' result, we get,

$$T_p = \frac{t_c r \mu_r}{P} + \sigma\sqrt{2\frac{r}{P}\log P} \tag{1}$$

This term does not include the communication overhead incurred by this partitioning.

We can combine this result with the communication overheads of this partitioning technique and adapt it directly to our problem. For computing communication overhead, note that the shape of the partitions can be easily controlled for optimum area to perimeter ratio by changing the number of clusters along a particular dimension. In this case, all elements in a partition, which lie on the boundary, may require communication with a processor $\log P$ steps away. It can be verified that the communication overhead for each cluster is given by

$$T_1 = 4t_b\sqrt{\tfrac{N}{r}} \times \log P$$

Now, since the number of clusters per processor is given by r/P, total communication overhead will be given by

$$T_o = \tfrac{r}{P} \times 4t_b \times \sqrt{\tfrac{N}{r}}\log P.$$

The parallel time (T_p) is given by

$$T_p = \tfrac{t_c r \mu_r}{P} + \sigma\sqrt{2\tfrac{r}{P}\log P} + \tfrac{r}{P} \times 4t_b \times \sqrt{\tfrac{N}{r}}\log P.$$

Since $r\mu_r = N$,

$$T_p = \frac{t_c N}{P} + \sigma\sqrt{2\frac{r}{P}\log P} + \frac{r}{P} \times 4t_b \times \sqrt{\frac{N}{r}}\log P.$$

The first term represents the actual computation, the second term represents the overhead due to load imbalance and the third term represents overhead due to communication. Also, in the expression, r must grow at least as $O(P \log P)$.

For isoefficiency due to communication, we equate the communication overhead with the essential computation to get

$$\frac{t_c N}{P} \sim \frac{r}{P} \times 4t_b \times \sqrt{\frac{N}{r}}\log P.$$

$$\sqrt{N}t_c \sim 4t_b\sqrt{r}\log P.$$

$$Nt_c \sim 16\frac{t_b^2}{t_c}r \times \log^2 P.$$

Since r must grow at least as $O(P \log P)$,

$$Nt_c \sim O(P \log^3 P)$$

For isoefficiency due to load imbalance, we have

$$\frac{t_c N}{P} \sim \sigma\sqrt{2\frac{r}{P}\log P}$$

Now, $r = N/\mu_r$, therefore,

$$\frac{t_c N}{P} \sim \sigma\sqrt{2\frac{N}{\mu_r P}\log P}$$

$$t_c N \sim \frac{2\sigma^2}{\mu_r t_c} P \log P$$

Now, if σ^2/μ_r [1] goes up as $O(\log^2 P)$ or less, isoefficiency due to load imbalance goes up as $O(P \log^3 P)$ or less.

The overall isoefficiency of scattered decomposition is thus given by $O(P \log^3 P)$ with an inverse dependence on t_c and a square dependence on t_b.

6 Experimental Evaluation.

We present results from the simulations of various partitioning strategies. Figure 2 shows the structure of the meshes used in these simulations. This mesh is used in actual fluid flow computation [2]. This mesh was scaled uniformly to generate meshes with elements ranging from 2000 elements to 300,000 elements. Simulations were run for up to 1024 processors connected in a hypercube. All partitioning techniques were run for various values of unit computation(t_c), message startup (t_s) and per-word transfer times (t_b). To observe the effect of each of these parameters on performance,

[1] If we assume a Poisson distribution of elements, $\sigma^2 = \mu_r$ and consequently, σ^2/μ_r is a constant

P→		2	4	8	16	32	64	128	256	512	1024
	N ↓										
Striping	197968	98.99	97.34	95.73	90.08	81.85	69.06	55.24	50.81	38.36	21.32
	98984	98.64	96.94	94.57	88.58	79.25	65.09	50.98	39.17	33.10	19.81
Bin. Dec.	197968	99.40	98.32	98.02	97.43	93.82	91.95	83.06	77.72	65.31	51.15
	98984	99.39	98.22	97.90	97.18	93.28	90.98	81.15	73.51	60.60	42.96
Scat. Dec.	197968	99.13	98.50	96.30	91.56	83.47	79.29	68.80	58.06	47.79	40.53
	98984	98.97	98.51	96.27	91.01	82.20	77.71	65.75	54.08	44.24	35.67

Table 1: Comparison of efficiencies of various decomposition techniques for $t_c : t_s : t_b$ given by $1 : 5 : 1$.

three values of ratios of t_c,t_s and t_b were used. These are given by 1:0:1, 1:5:1 and 5:100:1. The first set corresponds to a SIMD machine, the second set to a future MIMD machine and the third set to a current MIMD machine.

Tables 1, 3 and 2 present results of various decomposition strategies applied to two different meshes containing 100K and 200K elements approximately. The tables correspond to t_c, t_s and t_b in ratio of 1:5:1, 1:0:1 and 5:100:1 respectively. Both binary decomposition and scattered decomposition techniques perform significantly better for lower values of startup times, whereas striping gains very little in terms of performance by reduced startup times. On the other hand, a decrease in the per-word transfer time causes a more significant improvement for striping than for either binary decomposition or scattered decomposition. The reason for this is as follows. Striping generates a smaller number of messages, whereas scattered decomposition generates a larger number of small messages, and binary decomposition lies between the two. Note that the efficiencies for all partitioning schemes is very good for small number of processors, but it diverges for larger number of processors. This behavior at larger number of processors is captured by scalability and is discussed below.

To derive isoefficiency plots for different efficiencies, each decomposition technique was run on the test cases, points of equal efficiency were identified and plotted. The isoefficiency plots presented here were all compiled for t_c, t_s and t_b in the ratio 1:5:1.

Figure 3 shows the variation of problem size (sequential execution time) with number of processors to maintain constant efficiency levels for striping. The theoretically derived isoefficiency for striping is given by $O(P^2)$. To validate this, we plot problem size against P^2 in Figure 4. The plots for various efficiencies are observed to be straight lines. For higher efficiencies, the slope is observed to be high. This is because the $(E/(1-E))^2$ term in the isoefficiency grows very fast as efficiency approaches 1. This is in agreement with the theoretical result of the isoefficiency of striping technique.

Figure 2: Sample finite element mesh used to evaluate decomposition techniques.

P→		2	4	8	16	32	64	128	256	512	1024
	N ↓										
Striping	197968	99.78	99.41	99.03	97.68	95.40	91.14	85.14	82.11	72.93	39.96
	98984	99.70	99.29	98.72	97.19	94.39	89.31	81.99	73.62	66.62	38.45
Bin. Dec.	197968	99.38	98.25	97.87	96.90	92.85	90.13	81.22	73.44	61.75	49.52
	98984	99.36	98.07	97.58	96.05	91.27	87.07	76.87	65.76	52.94	37.09
Scat. Dec.	197968	99.06	98.07	96.04	91.14	83.19	77.90	66.13	55.55	47.15	39.44
	98984	98.25	97.04	95.10	89.96	80.91	75.10	63.03	50.97	42.66	29.19

Table 2: Comparison of efficiencies of various decomposition techniques for $t_c : t_s : t_b$ given by $5 : 100 : 1$.

P→		2	4	8	16	32	64	128	256	512	1024
	N ↓										
Striping	197968	99.00	97.35	95.75	90.12	81.91	69.14	55.34	50.98	38.55	21.55
	98984	98.65	96.96	94.61	88.64	79.35	65.23	51.15	39.37	33.39	20.22
Bin. Dec.	197968	99.40	98.35	98.08	97.66	94.25	92.50	84.19	80.14	68.19	55.55
	98984	99.40	98.28	98.01	97.53	94.13	92.17	83.33	77.18	65.76	49.57
Scat. Dec.	197968	99.30	98.77	96.57	91.87	85.31	79.81	68.89	63.80	50.74	44.24
	98984	99.32	99.06	96.8	91.62	84.33	79.56	68.68	58.41	47.38	41.85

Table 3: Comparison of efficiencies of various decomposition techniques for $t_c : t_s : t_b$ given by $1 : 0 : 1$.

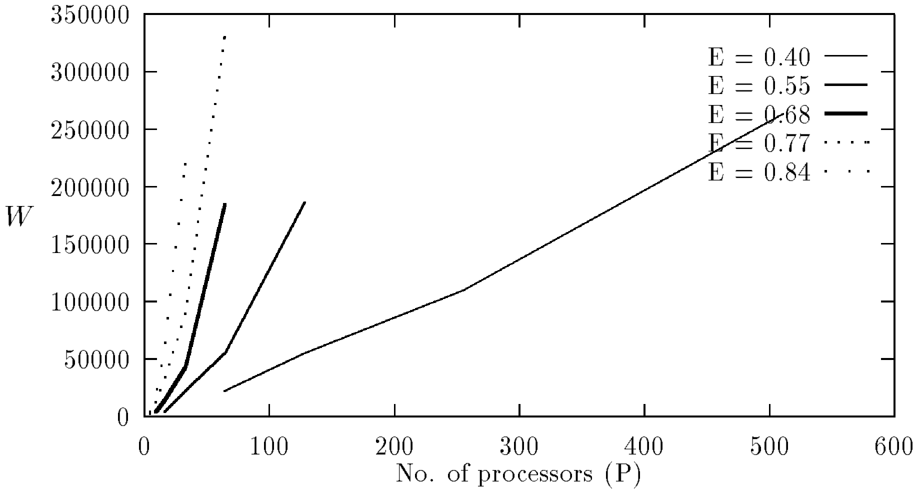

Figure 3: Number of processors vs. problem size for fixed efficiency for Striping.

Figure 5 shows the isoefficiency plots for binary decomposition. The theoretical isoefficiency of this technique is lower bounded by $O(P \log^2 P)$. Figure 6 plots the problem size against $P \log^2 P$. Clearly the rate of increase can be seen to be slightly higher than the lower bound of $P \log^2 P$. We would like to point out that we did not use any sophisticated scheme to optimize shapes of resulting partitions. We simply alternated the orientation of partitioning in successive partitioning phases. It is noteworthy that even such a simple implementation resulted in scalability close to the lower bound.

For scattered decomposition, our analysis indicated that for keeping the efficiency constant, the number of clusters must increase as $O(P \log P)$ and the cluster size should increase as $O(\log^2 P)$. In our experiments, as we scaled up P, we increased the number of elements as $O(P \log^3 P)$ and the number of clusters as $O(P \log P)$. This gave us theoretically predicted problem size for required efficiency. These problem sizes were then scaled up or down, as needed, to get equal efficiency points experimentally. Figure 7 shows the isoefficiency plots for scattered decomposition. The theoretically derived isoefficiency of $O(P \log^3 P)$ is validated in Figure 8 by plotting problem size against $P \log^3 P$. The plots are observed to be close to straight lines, which is in agreement with the theoretical isoefficiency function.

7 Conclusions.

In this study, we investigated the suitability of selected partitioning schemes for different problem characteristics using the isoefficiency criterion. We have shown that striping has an isoefficiency

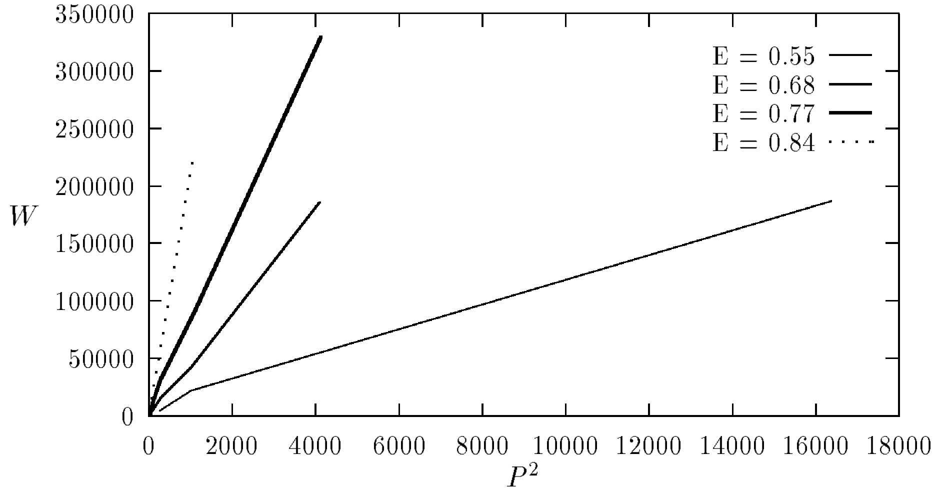

Figure 4: P^2 vs. problem size for fixed efficiency for Striping.

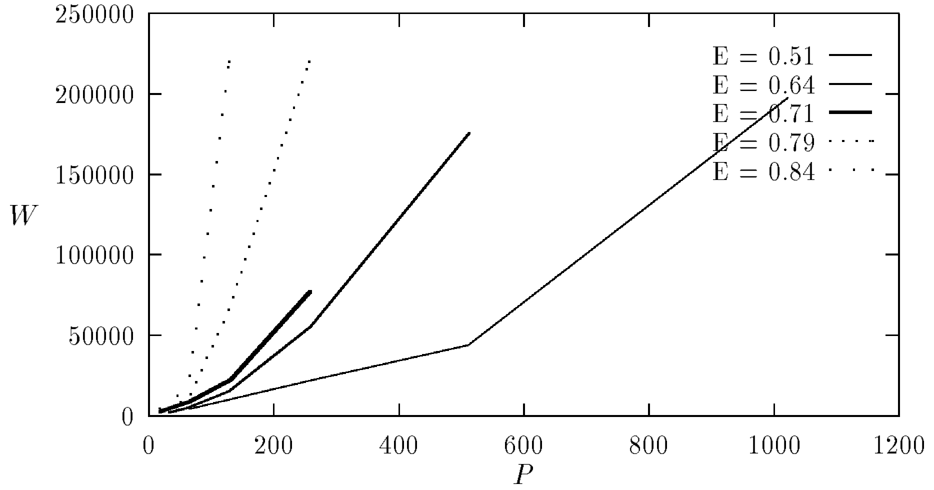

Figure 5: Number of processors vs. problem size for fixed efficiency for Binary Decomposition.

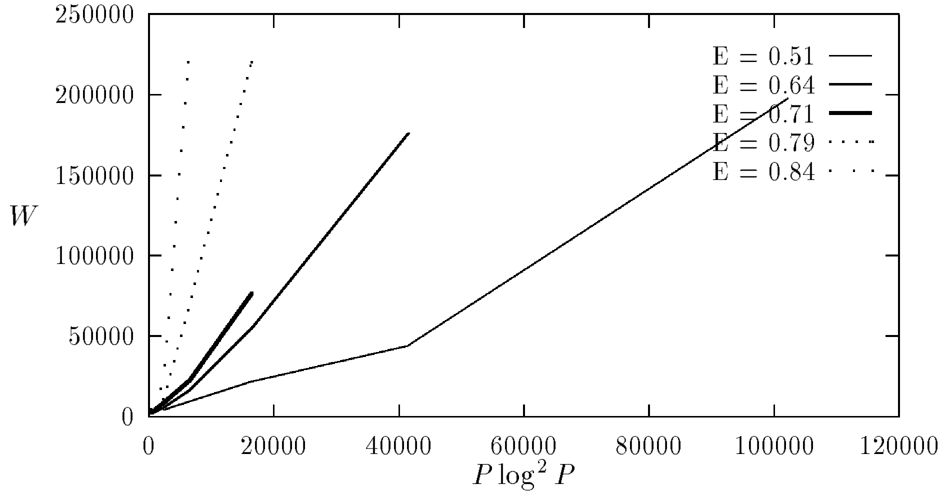

Figure 6: $P \log^2 P$ vs. problem size for fixed efficiency for Binary Decomposition.

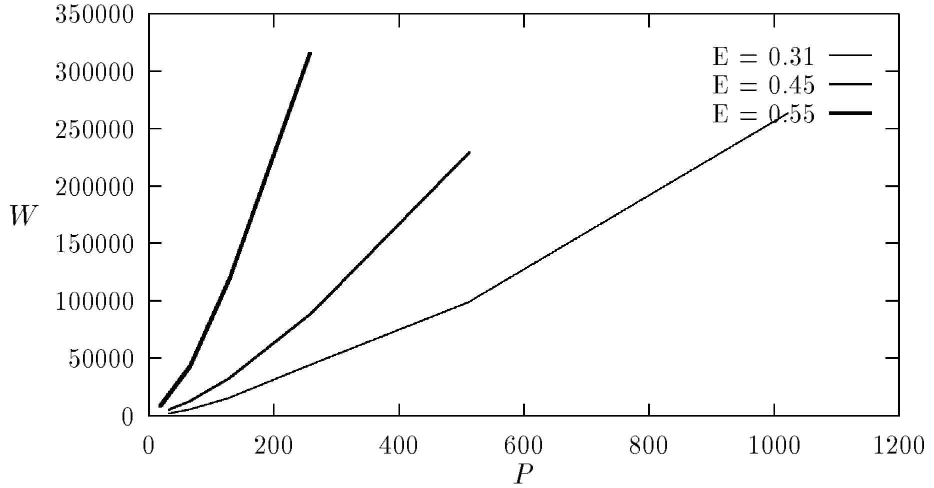

Figure 7: Number of processors vs. problem size for fixed efficiency for Modular Scattered Decomposition.

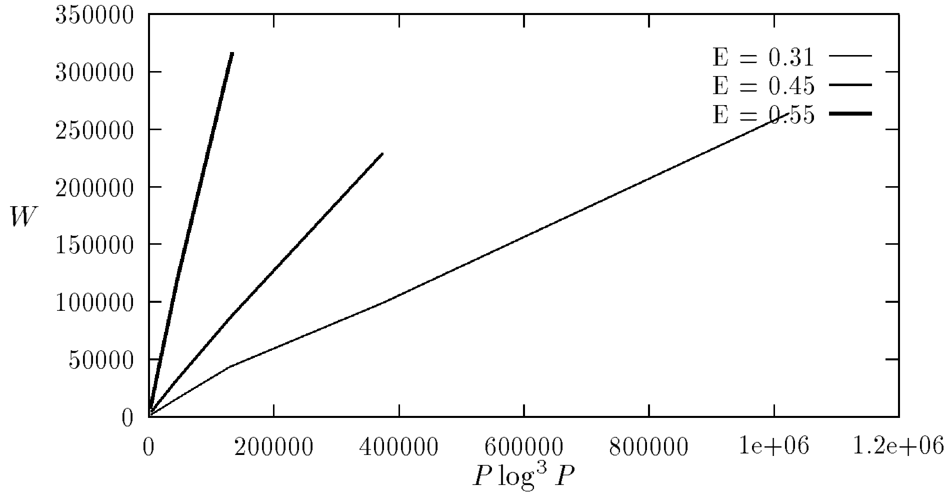

Figure 8: $P \log^3 P$ vs. problem size for fixed efficiency for Modular Scattered Decomposition.

of $O(P^2)$, scattered decomposition has an isoefficiency of $O(P \log^3 P)$ and binary decomposition has an isoefficiency which lies between $O(P \log^2 P)$ and $O(P^2)$ depending on the shapes of the partitions. Analysis for mesh connected architectures can be performed in a similar fashion as presented in this paper but is not detailed for sake of brevity. Table 4 presents isoefficiencies for hypercube and mesh connected architectures. Experimental results show that using very simple techniques to optimize communication volume, binary decomposition can achieve close to best case isoefficiencies. These results indicate that striping is not efficient for large number of processors. We also see that binary decomposition yields the best performance of the three on a range of processors. In general, binary Decomposition will perform better than Scattered Decomposition when the mesh is very irregular. This is because scattered decomposition would require a large number of clusters to be able to balance load.

Both binary decomposition and scattered decomposition techniques perform significantly better for lower values of startup times, whereas striping gains very little in terms of performance by reduced startup times. The performance of striped partitioning improves more significantly than others with decrease in the per-word transfer time, as the volume of communication is much higher in this scheme.

In a continuation of this study, we hope to cover a broader horizon of partitioning and mapping strategies. We also hope to study the applicability of various partitioning techniques to other techniques of solving the system of equations. Techniques like Clustered Element-By-Element (CEBE) provide a promise for significant performance improvements. New partitioning techniques which would be conducive to these solution techniques will have to be identified and validated.

Partition	Hypercube	Mesh	Setup time (order)
Stripes	P^2	P^2	N
Binary Decomposition	$P \log^2 P$	P^2	$N \log P$
Scattered Decomposition	$P \log^3 P$	$P^{1.5} \log^2 P$	N

Table 4: Iso-Efficiencies for different partitioning techniques on the hypercube and the mesh architectures (table shows best case results for binary decomposition).

References

[1] C. Aykanat, F. Ozguner, F. Ercal, and P. Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. In *IEEE Transactions on computers, Vol 37, pp.1554 - 1568*, December 1988.

[2] M. Behr, T.E. Tezduyar, and H. Higuchi. Wake interference behind two flat plates normal to the flow. Technical report, Tech Report 91-21, Army High Performance Computing Reseaarch Center, University of Minnesota, 1991.

[3] M.J. Berger and S.H. Bokhari. Partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, 1987.

[4] D.M.Nichol and J.H.Saltz. An analysis of scatter decomposition. *IEEE Transactions on Computers*, Nov 1990.

[5] Vipin Kumar et. al. Sacalability analysis of load-balancing schemes on parallel computers. Technical report, Computer Science Department, University of Minnesota, 1989 (working paper).

[6] et.al G. Fox. *Solving Problems on Concurrent Computers.* Prentice-Hall, 1988.

[7] M. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[8] Ananth Grama, Vipin Kumar, and V. Nageshwara Rao. Experimental evaluation of load balancing techniques for the hypercube. In *Proceedings of the Parallel Computing 91 Conference*, 1991.

[9] Anshul Gupta and Vipin Kumar. On the scalability of FFT on parallel computers. In *Proceedings of the Frontiers 90 Conference on Massively Parallel Computation*, October 1990. An extended version of the paper is available as a technical report from the Department of Computer Science and Army High Performance Computing Research Center, University of Minnesota.

[10] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, SE-11, NO. 10:1001–1016, Oct. 1985.

[11] Vipin Kumar, Ananth Grama, and V. Nageshwara Rao. Scalable load balancing techniques for parallel computers. Technical report, Tech Report 91-55, Computer Science Department, University of Minnesota, 1991.

[12] Vipin Kumar and V. Nageshwara Rao. Parallel depth-first search, part II: Analysis. *International Journal of Parallel Programming*, 16 (6):501–519, 1987.

[13] Vipin Kumar and V. Nageshwara Rao. Load balancing on the hypercube architecture. In *Proceedings of the 1989 Conference on Hypercubes, Concurrent Computers and Applications*, pages 603–608, 1989.

[14] Vipin Kumar and Vineet Singh. Scalability of Parallel Algorithms for the All-Pairs Shortest Path Problem: A Summary of Results. In *Proceedings of the International Conference on Parallel Processing*, 1990. Extended version available as a technical report from the department of computer science, University of Minnesota, Minneapolis, MN 55455 and as MCC TR ACT-OODS-058-90.

[15] V. Nageshwara Rao and Vipin Kumar. Parallel depth-first search, part I: Implementation. *International Journal of Parallel Programming*, 16 (6):479–499, 1987.

[16] J. W. Parker, T. Cwik, R. Ferraro, P. Liewer, P. Lyster, and J. Patterson. Helmholtz finite elements performance on mark iii and intel ipsc/860 hypercubes. In *Proceedings of 6th DMCC conference, Portland, Oregon*, 1991.

[17] S. Ranka and S. Sahni. *Hypercube Algorithms for Image Processing and Pattern Recognition.* Springer-Verlag, New York, 1990.

[18] R.Morrison and S.Otto. The scattered decomposition for finite elements. *Journal of Scientific Computing.*, 2. No.1, March 87.

[19] P. Sadayappan and F. Ercal. Mapping of finite element graphs onto processor meshes. *IEEE Transactions on Computers*, C-36:1408–1424, 1987.

[20] Vineet Singh, Vipin Kumar, Gul Agha, and Chris Tomlinson. Scalability of parallel sorting on mesh multicomputers. Technical Report ACT-SPA-298-90, Microelectronics and Computer Technology Corp., Austin,TX, 1990. Also available as a technical report (number TR 90-45) from the department of computer science, University of Minnesota, Minneapolis, MN 55455.

[21] Vineet Singh, Vipin Kumar, Gul Agha, and Chris Tomlinson. Scalability of parallel sorting on mesh multicomputers. In *Proceedings of the Fifth International Parallel Processing Symposium*,

March 1991. Extended version available as a technical report (number TR 90-45) from the department of computer science, University of Minnesota, Minneapolis, MN 554 55, and as TR ACT-SPA-298-90 from MCC, Austin, Texas.

[22] Winifred Williams. Load balancing on hypercubes : A preliminary look. In *Hypercubes 87.*, 1987.

[23] Jinwoon Woo and Sartaj Sahni. Computing biconnected components on a hypercube. Technical Report TR 89-7, Computer Science Department, University of Minnesota, Minneapolis, MN 55455, February 1989.

[24] Jinwoon Woo and Sartaj Sahni. Hypercube computing : connected components. Technical Report TR 88-50, Computer Science Department, University of Minnesota, Minneapolis, MN 55455, July 1988.

[25] Sanjay Ranka. Yeh-Ching Chung. Mapping finite element graphs on hypercubes. In *Proceedings of Frontiers of Massively Parallel Computations*, 1990.