International Journal of Foundations of Computer Science Vol. 8 No. 3 (1997) 329–346

A SCALABLE DIFFUSION ALGORITHM FOR DYNAMIC MAPPING AND LOAD BALANCING ON NETWORKS OF ARBITRARY TOPOLOGY

A. HEIRICH

Center for Advanced Computing Research & Department of Computer Science
California Institute of Technology, Pasadena, California 91125 USA

Received 1 November 1995
Revised 6 August 1996
Communicated by D. F. Hsu

ABSTRACT

The problems of mapping and load balancing applications on arbitrary networks are considered. A novel diffusion algorithm is presented to solve the mapping problem. It complements the well known diffusion algorithms for load balancing which have enjoyed success on massively parallel computers (MPPs). Mapping is more difficult on interconnection networks than on MPPs because of the variations which occur in network topology. Popular mapping algorithms for MPPs which depend on recursive topologies are not applicable to irregular networks. The most celebrated of these MPP algorithms use information from the Laplacian matrix of a graph of communicating processes. The diffusion algorithm presented in this paper is also derived from this Laplacian matrix. The diffusion algorithm works on arbitrary network topologies and is dramatically faster than the celebrated MPP algorithms. It is delay and fault tolerant. Time to convergence depends on initial conditions and is insensitive to problem scale. This excellent scalability, among other features, makes the diffusion algorithm a viable candidate for dynamically mapping and load balancing not only existing MPP systems but also large distributed systems like the Internet, small cluster computers, and networks of workstations.

Keywords: mapping, load balancing, embedding, diffusion, parallel, fault tolerant.

1. Introduction

Commodity interconnection networks have made qualitative breakthroughs in performance and price in recent years. These breakthroughs have reduced effective latency and increased effective bandwidth to the point where the performance of commodity components equals or surpasses that of MPP components.[1,2] At the same time costs have declined to the point where they are effectively redefining the term "parallel computer". Considerable discussion has been generated about building inexpensive and scalable parallel systems from commodity components and of constructing distributed systems from existing workstations.[3,4,5] Several MPP vendors and at least one major PC manufacturer offer or are developing inexpensive

clustered systems based on these technologies. These systems will have performance attributes equal to or better than the best current MPP systems at costs that are at least an order of magnitude below current levels.

The hardware components of these computer systems are developing rapidly. The same statement could not be made about software. One might conjecture that a software infrastructure can be adapted from the existing infrastructure for MPPs. This conjecture would be at least partially wrong. There are fundamental differences between networks and MPPs which have impacts on algorithm design and software implementation. Networks have highly variable topologies while most MPPs are based on a few well studied topologies. Networks have faults and failures while MPPs are designed to provide high levels of error free operation.

In order to execute any application on one of these systems it is necessary to solve the mapping problem.[6] In the mapping problem an application, consisting of a (rather large) set of communicating processes, is embedded in a (smaller) set of communicating computers. In practice the processes are often replaced by a data structure, such as a PDE grid, which must be distributed among the computers. An algorithm which solves the mapping problem seeks an embedding which lets the application run in the shortest possible time. In practice this means it seeks an embedding in which the workload is balanced and the time spent in communication is minimal. Experience with applications on existing MPPs has shown that, with an appropriate software model, maximizing locality and overlapping communication with computation can often prevent any time from being lost in communication.[7,8,9]

The problems of mapping and load balancing have been discussed since the earliest days of parallel computing.[6,10,11] For problems on MPPs the scientific community has converged on a consensus that favors recursive bisection methods. These methods exploit the regular structure of the MPP in order to simultaneously map and load balance an application. For the most difficult problems the most powerful of these methods find high quality solutions at correspondingly high cost.[12] For easy problems like PDE grids the fastest of these methods achieve the optimal parallel execution time of $\mathcal{O}(\log p)$.[13] Recursive bisection methods depend on the recursive nature of the interconnection topology. When the networks have irregular structure both load balancing and mapping are more difficult and recursive bisection methods fail.

Efficient diffusion algorithms to solve the load balancing problem have been considered for many years.[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30] All of these algorithms work by diffusing quantities of work between pairs of computers until achieving an equilibrium. This paper presents an efficient diffusion algorithm to solve the mapping problem. The algorithm imposes no regularity assumption on the interconnection network and therefor it can succeed where recursive bisection methods fail. The algorithm is derived from the Laplacian matrix of a graph of communicating processes. Vertices of this graph are diffused in a Euclidian space which represents the computer system. The result of this diffusion is an embedding of the components into the computers. Under this embedding locality is maximal and workload is balanced.

The rest of this paper is organized as follows. It first presents the combined problems of mapping and load balancing in the form of a minimization problem. It discusses the relationship between the topology of the interconnection network and the Euclidian space in which the diffusion occurs. The paper next presents a model problem instance which will be used in the subsequent analysis of the algorithm. It presents the algorithm and proves its correctness. Through analysis the paper demonstrates that the algorithm has excellent scaling properties. The desirable properties of delay and fault tolerance follow directly. The paper demonstrates a two phase mapping and load balancing procedure which reduces the time to convergence, presents simulations, and concludes with a summary and discussion.

2. A Minimization Problem

The object of load balancing and mapping is to place an application (represented by a set of processes) onto a set of computers (connected by a network) so that the application can run in the shortest possible time. This description implies the existence of an objective function,

$$Cost(C, D, \vec{w}, \mathcal{F}) = \max_p \sum_i \delta_{\mathcal{F}(i),p} w_i + \sum_{i,j,p,p'} \delta_{\mathcal{F}(i),p} \delta_{\mathcal{F}(j),p'} c_{i,j} d_{p,p'} \qquad (1)$$

This function (1) represents the amount of time required for an application to terminate as a result of workload imbalance and communication overhead. The quantities of interest are as follows,

- i, j are indices of processes in the application.
- p, p' are indices of computers connected by a network.
- w_i is the workload associated with process i.
- $c_{i,j}$ is the communication bandwidth required between processes i and j.
- $d_{p,p'}$ is the routing distance between computers p and p' through the network.
- $\mathcal{F}$ is a mapping function from tasks to computers.
- $\delta_{\mathcal{F}(i),p}$ is a truth predicate with the value one if $\mathcal{F}(i) = p$ and zero otherwise.

The first term describes the time due to load imbalance and is minimal when the workloads on all computers are the same. The second term describes the time spent in communication in the form of bandwidth $c_{i,j}$ multiplied by routing distance $d_{p,p'}$ among pairs of communicating processes. For a given C the second term is small if $d_{p,p'}$ is small for nonzero δ. In informal terms the minimization problem can be stated as follows: given sets of processes and computers, $\vec{w}$, and matrices C and D, find a mapping $\mathcal{F}$ which balances workload and minimizes the routing distance between pairs of communicating processes.

Formulated in this way the problem possesses an exponential number of possible solutions and thus it should not be surprising to find that it is NP-complete. One proof of this fact is by reduction from the number partition problem:[31] given an arbitrarily large set of integers drawn from an unknown distribution, does there exist a partition of the integers into two subsets whose components have the same sum? An algorithm which finds a minimal solution of the first term of (1) could solve the number partition problem. Since number partition is known to be NP-complete, so too is problem (1). With respect to the second term of (1) it has long been known to be equivalent to a number of related NP-complete problems including graph isomorphism and matrix bandwidth reduction.[10]

This formulation of the problem is consistent with other discussions which avoid explicit consideration of network contention.[32] Although contention is not explicitly treated it enters indirectly by observing that in most interconnection technologies the likelihood of contention is reduced when routing distances are decreased. Thus minimizing the second term of (1) results in minimal network contention.

3. The Setting of the Problem

An instance of the problem is defined by two graphs which describe the interconnection network and the application. The network is described by a "host" graph $\mathcal{H}$ in which edges in $E(\mathcal{H})$ represent network links and vertices in $V(\mathcal{H})$ represent computers. The application is described by a "guest" graph $\mathcal{G}$ in which edges in $E(\mathcal{G})$ represent software communication channels and vertices in $V(\mathcal{G})$ represent processes. Given instances of $\mathcal{G}$ and $\mathcal{H}$ the problem is to construct a map $\mathcal{F}$ which places processes onto computers. The quality of any candidate solution $\mathcal{F}$ is evaluated by predicting the time required to execute the application under $\mathcal{F}$ according to (1).

In order to make this problem tractable the routing distance D in (1) will be approximated by a metric $\mathcal{D}$ on a Euclidian space Ω in R^m derived from $\mathcal{H}$. The space Ω is partitioned into regions which represent vertices of $\mathcal{H}$. If two regions are adjacent in Ω they correspond to two vertices which share an edge in $\mathcal{H}$. As a result $\mathcal{D}$, which is a metric on Ω, is also a metric on $\mathcal{G}$ under $\mathcal{F}$.

This paper will take m to be 2 and Ω to be the unit square. This space is sufficient to represent any planar interconnection network. It should be emphasized that these assumptions do not restrict the generality of the algorithm. Both m and $\mathcal{D}$ can be generalized according to the requirements of a particular problem instance. Thus while Ω might be two dimensional when $\mathcal{H}$ is planar (for example in the case of a mesh or irregular network) it could have three or more dimensions for a more complex topology (for example, a three dimensional torus). The only requirement is that if two vertices are connected by an edge in $\mathcal{H}$ then the corresponding subregions of Ω must share an interface.

Toroidal topologies of $\mathcal{H}$ present no problems for this algorithm, as it is trivial to modify the diffusion process to allow vertices of $\mathcal{G}$ to diffuse as though Ω were a torus. While it might at first appear that recursive topologies such as hypercubes and star networks require large m this is not the case. A network of this sort can

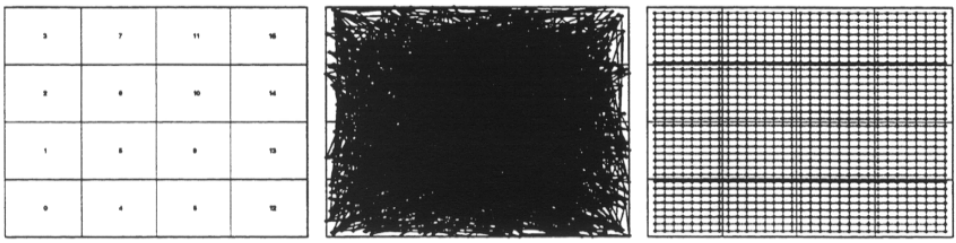

Figure 1: The model problem instance used for analysis in this paper. Left, a partitioning of the unit square Ω into equitable regions corresponding to $\mathcal{H}$ a 4×4 mesh of computers. Middle, a bad solution $\mathcal{F}$ for $\mathcal{G}$ a regular lattice. Right, a perfect solution for the same $\mathcal{G}$ in which locality is maximal and workload is balanced.

be mapped into a torus by selecting an initial node and then traversing the network in a radially expanding pattern until all nodes have been visited. This is similar to standard algorithms to perform broadcast operations in these topologies.

4. A Model Problem Instance

This paper will consider a simple model problem instance that can be generalized to any problem instance. The guest graph $\mathcal{G}$ will be a regular lattice. An example of such a lattice would be a uniform grid for a PDE problem in two dimensions. This simple structure lends itself to the analysis of parallel complexity in later sections of this paper. The conclusions of the analysis will generalize to arbitrary problems. The host graph $\mathcal{H}$ for the model problem can represent any planar network. For the sake of simplicity $\mathcal{H}$ will be also be assumed to be a regular lattice, although much smaller than $\mathcal{G}$. This assumption is convenient but does not restrict the generality of the subsequent analysis.

A virtue of the model problem is that its correct solution is known. Figure 1 shows the partitioning of Ω according to $\mathcal{H}$ and two candidate embeddings of $\mathcal{G}$. From this figure it is apparent that a perfect solution of this model problem is one in which the edges of $\mathcal{G}$ under $\mathcal{F}$ have equal lengths. When the extremal vertices of $\mathcal{G}$ are properly constrained this solution is unique. This observation suggests an algorithm to find an optimal $\mathcal{F}$ for any planar $\mathcal{G}$. This algorithm constrains the extremal vertices of $\mathcal{G}$, then searches for a placement of the vertices of $\mathcal{G}$ in the plane under which the edges have equal length. A solution of this sort exists for every planar $\mathcal{G}$ but not for general $\mathcal{G}$.

5. An Efficient Dynamic Mapping Algorithm

A general algorithm to solve the mapping problem would converge to the perfect solution whenever $\mathcal{G}$ is planar and to nearby solutions for nonplanar graphs which are similar to $\mathcal{G}$. One way to construct such a general algorithm is to define an algorithm with a discrete postcondition which is correct for planar $\mathcal{G}$ and then

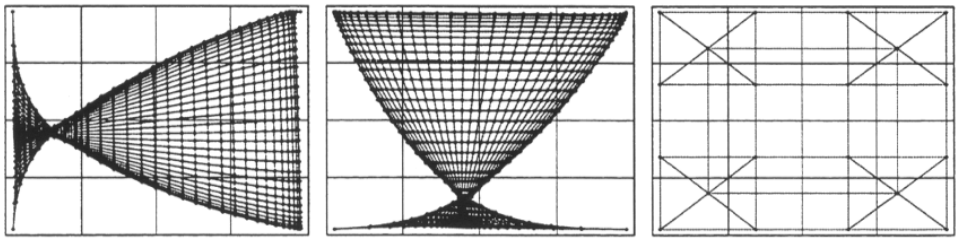

Figure 2: The correctness of the solution depends on properly constraining the extremal vertices of $\mathcal{G}$. Left and middle, solutions $\mathcal{F}(\mathcal{G})$ resulting from incorrectly ordered constraints. Right, a nonplanar instance of $\mathcal{G}$ corresponding to a two level structured multigrid. There exists no nontrivial $\mathcal{F}$ under which all edges of this instance have equal length.

implement the algorithm using a continuous postcondition which can be satisfied to within a tolerance for general $\mathcal{G}$. Figure 3 presents an informal algorithm which solves the mapping problem for planar $\mathcal{G}$. Each vertex $v \in \mathrm{V}(\mathcal{G})$ is assumed to occupy a position in Ω. The algorithm is a simple iteration which repeatedly repositions vertices of $\mathcal{G}$ in Ω. It terminates when all vertices of $\mathcal{G}$ have edges of equal length. This discrete postcondition can be satisfied only for planar $\mathcal{G}$. Figure 4 presents an implementation of an algorithm with a continuous postcondition which can be satisfied by any $\mathcal{G}$, planar or nonplanar. The notation and proof style are influenced by classical expositions.[33,34,35]

From the discrete postcondition it is clear that the informal algorithm (figure 3) solves the mapping problem in this setting if it terminates. The discrete postcondition requires the edges of $\mathcal{G}$ under $\mathcal{F}$ to have equal lengths when the extremal vertices are properly constrained. Figures 1 and 2 illustrate that this postcondition satisfies the requirements of the mapping problem for planar $\mathcal{G}$. Figure 2 also illustrates that this discrete postcondition cannot be satisfied when $\mathcal{G}$ is nonplanar.

When $\mathcal{G}$ is planar the implementation (figure 4) converges to the same solution as the informal algorithm. As a result the implementation is correct for planar $\mathcal{G}$. To see this expand the continuous postcondition ($norm \leq \epsilon$) with $\epsilon = 0$. Consider the progress of the computation after ν updates to every vertex v_i, with $v_i.x^{(\nu)}$ and $v_i.y^{(\nu)}$ denoting the values of the coordinates of vertex i after ν steps.

$$
\begin{aligned}
0 &\geq norm \\
&\geq \|\vec{\Delta}\|_2 \\
&\geq \left[\sum_i \left(dx^2 + dy^2\right)^2\right]^{1/2} \\
&\geq \left[\sum_i \left(\left(v_i.x^{(\nu+1)} - v_i.x^{(\nu)}\right)^2 + \left(v_i.y^{(\nu+1)} - v_i.y^{(\nu)}\right)^2\right)^2\right]^{1/2} \qquad (2)
\end{aligned}
$$

{ V_{ext} constrained }
do
($\exists v \in (\mathrm{V}(\mathcal{G}) \backslash V_{ext})$ with unequal edges) $\rightarrow$
{ (V_{ext} constrained) $\wedge$ (v has unequal edges) }
Equalize(v)
{ V_{ext} constrained }
od
{ (V_{ext} constrained) $\wedge$ ($\nexists v \in (\mathrm{V}(\mathcal{G}) \backslash V_{ext})$ with unequal edges) }

Figure 3: An informal algorithm to compute $\mathcal{F}(\mathcal{G})$ under which $\mathcal{G}$ has edges of equal lengths. $V_{ext} \subset \mathrm{V}(\mathcal{G})$ is the set of extremal vertices of $\mathcal{G}$. The statement Equalize(v) repositions $v \in \mathrm{V}(\mathcal{G}) \backslash V_{ext}$ to equalize the lengths of edges incident on v. As a result of the discrete postcondition this algorithm can only terminate for planar $\mathcal{G}$.

{ (V_{ext} constrained) $\wedge$ ($norm = \infty$) }
$c := c_0$;
do while ($norm > \epsilon$)
$i := \mathrm{Random}(1, |V(\mathcal{G}) \backslash V_{ext}|)$;
$v := v_i \in V(\mathcal{G}) \backslash V_{ext}$;
$dx := \sum_{(i,j) \in E(\mathcal{G})} v_j.x/d(v) - v.x$;
$dy := \sum_{(i,j) \in E(\mathcal{G})} v_j.y/d(v) - v.y$;
$\Delta_i := dx^2 + dy^2$;
($\Delta_i > 0$) $\rightarrow$
{ (V_{ext} constrained) $\wedge$ (v has unequal edges) }
($v.x := v.x + dx; v.y := v.y + dy$)
{ V_{ext} constrained }
$c := c - 1$;
($c = 0$) $\rightarrow$
($norm := \|\vec{\Delta}\|_2$; Renormalize; $c := c_0$)
od
{ (V_{ext} constrained) $\wedge$ ($norm \leq \epsilon$) }

Figure 4: An implementation of the informal algorithm. In this implementation the discrete postcondition is replaced by convergence of a continuous norm to within a prescribed tolerance. As a result the postcondition will be satisfied for any $\mathcal{G}$, planar or nonplanar, after a sufficient number of iterates, for appropriate ϵ. The existential quantifier $\exists v$ is implemented by randomly selecting and testing vertices of $\mathcal{G}$ which are located in Ω. The guard "unequal edges" is implemented by the condition $\Delta_i > 0$. The degree of v is $d(v)$. Evaluation of $\|\vec{\Delta}\|_2$ requires a global operation which can be handled in various ways. For the sake of simplicity it is presented here as occurring at fixed intervals $c_0 \gg |V(\mathcal{G})|$. The Renormalize statement normalizes the coordinates within the bounds of Ω. This operation has been found to accelerate convergence.

Since *norm* is nonnegative (2) can only be satisfied for $norm = 0$. This is true only for the unique fixed point at which all $v_i.x^{(\nu+1)} = v_i.x^{(\nu)}$ and similarly for y. From the definitions of dx and dy in the implementation it is apparent that at this fixed point the edges of $\mathcal{G}$ under $\mathcal{F}$ have equal length.

$$\begin{aligned} v_i.x^{(\nu+1)} &= v_i.x^{(\nu)} + \sum_{(i,j)\in E(\mathcal{G})} v_j.x^{(\nu)}/d(v_i) - v_i.x^{(\nu)} \\ &= \sum_{(i,j)\in E(\mathcal{G})} v_j.x^{(\nu)}/d(v_i) \qquad (3) \\ &= \sum_{(i,j)\in E(\mathcal{G})} v_j.x^{(\nu+1)}/d(v_i) \qquad (4) \end{aligned}$$

These arguments demonstrate that a unique fixed point exists for both the informal algorithm and the implementation whenever $\mathcal{G}$ is planar. In order to show that the implementation terminates it is necessary to show that the fixed point is attracting from any initial state. This can be shown by the fact that $\|\vec{\Delta}\|_2$ decreases monotonically under successive iterates of the implementation. Group the unknowns $\vec{x}, \vec{y}$ into a single vector $\vec{u}$. The progress of the unknowns can be modeled by a discrete dynamical system $\vec{u}^{(\nu+1)} := \mathcal{M}\left(\vec{u}^{(\nu)}\right)$ which operates on $\vec{u}$. This is a linear dynamical system and can be represented as a stationary linear iteration.

$$\vec{u}^{(\nu+1)} = M\vec{u}^{(\nu)} \qquad (5)$$

M is a matrix operator and has full rank. The diagonal entries of M are zero and the row sums are equal to one. By Gersgorins theorem[36] these conditions suffice to show the spectral radius of M is strictly less than one and therefore (5) converges to a fixed point. Since M has full rank this fixed point is unique. The analysis in the following section will demonstrate that this convergence is rapid in most instances. Convergence persists when the order in which the vertices are updated is allowed to vary randomly.[37] It follows trivially from this fact that the randomized iteration of figure 4 converges in the presence of delays and nondeterministic execution order.

This convergence implies that $\|\vec{u}\|_2$ decreases monotonically along trajectories of the dynamical system (5). In order to show termination of the implementation it is necessary to show a corresponding decrease in $\|\vec{\Delta}\|_2$. The dynamical system converges to a fixed point $\vec{u}^* = (M^\nu)\vec{u}^{(0)}$ after sufficient iterates $\nu < \infty$. The fixed point $\vec{u}^*$ satisfies the discrete postcondition of the informal algorithm when $\mathcal{G}$ is planar. In order to show that the implementation converges to $\vec{u}^*$ for any $\mathcal{G}$ it is necessary to show that $\|\vec{\Delta}\|_2$ is a bound function which decreases along any trajectory of the dynamical system. This is easily accomplished. The difference between successive iterates of (5) is $\|\vec{u}^{(\nu+1)} - \vec{u}^{(\nu)}\|_2$ which converges to zero as $\nu \to \infty$. This implies that $\|\vec{dx}\|_2 \to 0$ and $\|\vec{dy}\|_2 \to 0$ individually, and thus $\|\vec{dx^2 + dy^2}\|_2 \equiv \|\vec{\Delta}\|_2 \to 0$. This proves that the fixed point $\vec{u}^*$ is attracting from any initial state and thus the implementation terminates for any $\mathcal{G}$. Given that the continuous postcondition with appropriate ϵ meets the requirement of locality

maximization for any $\mathcal{G}$ we can conclude that the implementation is a correct general solution to the mapping problem.

6. Analysis of Parallel Complexity

The rate of convergence of the implementation is determined by the structure of $\mathcal{G}$ and the initial $\mathcal{F}$. The relationship between $\mathcal{G}$ and $\mathcal{F}$ can be quantified by studying the eigenstructure of M. Since M has full rank it is equipped with a full set of orthonormal eigenvectors $\vec{X}_{i,j}$ and eigenvalues $\lambda_{i,j}$. In the case of the model problem the λ and $\vec{X}$ are well known.[37]

$$\lambda_{i,j} = \frac{1}{4}\left(\cos\pi\frac{i}{n} + \cos\pi\frac{j}{n}\right)^2 \tag{6}$$

$$(X_{i,j})_{x,y} = k_{i,j}\cos\pi\frac{ix}{n}\cos\pi\frac{jy}{n} \tag{7}$$

The convergence of the implementation for any $\mathcal{G}$ can be analyzed by considering the action of each $\lambda_{i,j}$ on $\vec{X}_{i,j}$. From the eigenvalue identity $M\vec{X}_{i,j} = \lambda_{i,j}\vec{X}_{i,j}$ it follows that the best and worst cases of convergence occur for the smallest and largest λ, λ_{min} and λ_{max}. After ν iterates of (5) the magnitude of each $\vec{X}_{i,j}$ is diminished by $(\lambda_{i,j})^\nu$. In order to reduce the magnitude of a particular $\vec{X}_{i,j}$ by a factor ϵ it is necessary that $(\lambda_{i,j})^\nu < \epsilon$, or

$$\nu = \left\lceil \frac{\ln\epsilon}{\ln\lambda_{i,j}} \right\rceil \tag{8}$$

Applying this formula to

$$\lambda_{min} = \lambda_{31,0} = 5.79671 \times 10^{-6}, \lambda_{max} = \lambda_{1,0} = 0.995191 \tag{9}$$

with $\epsilon = 1/10$ reveals that the best and worst cases diminish by an order of magnitude after $\nu_{min} = 1$ and $\nu_{max} = 478$ respectively. The convergence from arbitrary initial conditions depends on the initial magnitudes of the $\vec{X}_{i,j}$. Any such initial condition can be expanded in an eigenvector basis.

$$\vec{u}^{(0)} = \sum_{i,j} a_{i,j}^{(0)}\vec{X}_{i,j} \tag{10}$$

The $a_{i,j}^{(0)}$ represent initial amplitudes of the $\vec{X}_{i,j}$. From (8) it follows that the time dependent configuration after ν iterates is

$$\vec{u}^{(\nu)} = \sum_{i,j} a_{i,j}^{(0)}(\lambda_{i,j})^\nu\vec{X}_{i,j} \tag{11}$$

Consider an atomic problem instance in which all vertices are optimally placed with the exception of a single vertex displaced by one unit. Assume this is the "first"

vertex under an arbitrary vertex ordering, that is, $u^{(0)}_{0,0} = 1$, $u^{(0)}_{i,j} = 0, 0 < i, j < n$. Then

$$\begin{aligned}
\vec{X}_{i,j} \cdot \vec{u}^{(0)} &= \vec{X}_{i,j} \cdot \sum_{k,l} a^{(0)}_{k,l} \vec{X}_{k,l} \\
&= \sum_{k,l} \left(\vec{X}_{i,j} \cdot \vec{X}_{k,l} \right) a^{(0)}_{k,l} \\
&= \sum_{k,l} a^{(0)}_{k,l} \delta_{i,k} \delta_{j,l} \\
&= a^{(0)}_{i,j}
\end{aligned}$$

From this and the assumption on $\vec{u}^{(0)}$ it follows that $(\vec{X}_{i,j})_{0,0} = a^{(0)}_{i,j}$ for all i, j. From (7) it follows that $(\vec{X}_{i,j})_{0,0} = k_{i,j} = a^{(0)}_{i,j}$ and thus

$$\begin{aligned}
u^{(0)}_{0,0} &= \sum_{i,j} k_{i,j} (\vec{X}_{i,j})_{0,0} \\
&= \sum_{i,j} k_{i,j} a^{(0)}_{i,j} \\
&= \sum_{i,j} k^2_{i,j}
\end{aligned}$$

The assumption on $\vec{u}^{(0)}$ and the normality of the $\vec{X}_{i,j}$ makes it possible to deduce an expression for the $k_{i,j}$.

$$\begin{aligned}
1 &= \vec{X}_{i,j} \cdot \vec{X}_{i,j} \\
&= \sum_{x,y} \left(k_{i,j} \cos \pi \frac{ix}{n} \cos \pi \frac{jy}{n} \right)^2 \\
&= k^2_{i,j} \frac{1}{4} \sum_{x,y} \left(1 + \cos 2\pi \frac{ix}{n} \right) \left(1 + \cos 2\pi \frac{jy}{n} \right) \\
&= k^2_{i,j} \frac{1}{8} \sum_{x,y} \left(1 + \cos 2\pi \frac{jy}{n} \right) \\
&= k^2_{i,j} \frac{1}{4} \sum_{x,y} \cos^2 2\pi \frac{jy}{n} \\
&= k^2_{i,j} n^2
\end{aligned}$$

Which reveals that $k_{i,j} = 1/n$. This quantity depends on n but not on the particular choice of i, j. This implies that $a^{(0)}_{i,j}$ is identical for all i, j.

$$a^{(0)}_{i,j} = 1/n \tag{12}$$

From (11) and the preceding one can write a precise expression for the magnitude of $\|\vec{\Delta}\|_2$ for the atomic problem instance after ν iterates.

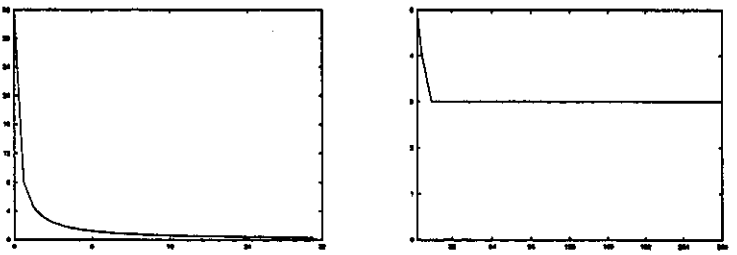

Figure 5: Time dependent behavior of (13) and the consequent implication of infinite scalability. Left, decay of $\|\vec{\Delta}\|_2$ on successive iterates of (5) for a fixed problem size. Vertical axis shows $\|\vec{\Delta}\|_2$ with number of iterates on the horizontal axis. Right, a demonstration of the infinite scalability of the atomic case. Vertical axis shows the smallest ν which satisfies (13) $\leq \vec{u}^{\nu}_{0,0}/10$, horizontal axis shows $n = |V(\mathcal{G})|^{1/2}$.

$$\begin{aligned} u^{(\nu)}_{0,0} &= \sum_{i,j} a^{(0)}_{i,j} (\lambda_{i,j})^{\nu} \\ &= \sum_{i,j} \frac{1}{4^{\nu} n} \left(\cos \pi \frac{i}{n} + \cos \pi \frac{j}{n} \right)^{2\nu} \end{aligned} \tag{13}$$

Figure 5 offers insight into the time dependent behavior of (5) for the atomic case. The time to converge (13) to within a prescribed ϵ can be bounded by a constant regardless of the size of $V(\mathcal{G})$. Indeed this is a generic feature of diffusion algorithms. The rate of convergence to an equilibrium is primarily determined by the distance of the initial state from the fixed point. This observation suggests that diffusion is an attractive paradigm for concurrent algorithms since it scales efficiently to problems of any size.

This analysis was derived for a specific problem instance. The excellent scaling is a result of the scale invariance of the Laplacian spectrum (6,13). The magnitudes of the $\lambda_{i,j}$ will vary when $\mathcal{G}$ varies from the model problem instance. This variation is smooth and will never allow the iteration to diverge (by Gersgorins theorem). Therefore the conclusions of the analysis of the model problem instance will be qualitatively similar to the conclusions for any instance. The smooth variation in the eigenvalues of M follows from the observation that M is a linear transformation of the *Laplacian matrix* of $\mathcal{G}$ and the eigenvalues of this Laplacian matrix vary smoothly as $\mathcal{G}$ varies from the model problem.

Theorem 3.2[38]

Call $\vec{\lambda}$ the eigenvalues of the Laplacian matrix $Q(G)$ of a graph G. Construct a new graph G' by adding an edge between any two vertices of G. Then if $\vec{\lambda'}$ are the eigenvalues of $Q(G')$,

$$0 = \lambda_0 = \lambda'_0 \leq \lambda_1 \leq \lambda'_1 \leq \lambda_2 \leq \lambda'_2 \cdots \leq \lambda_{max} \leq \lambda'_{max}$$

This theorem can be used to generalize the results of the preceding analysis for the model problem instance to problem instances for any $\mathcal{G}$. The argument is by induction on the number of edges by which G and G' differ.

7. Delay and Fault Tolerance

Commodity interconnection networks differ from the interconnection fabrics used in existing MPPs in terms of performance and reliability. The issue of reliability imposes qualitatively new requirements upon software designers since it forces software to be fault tolerant. This requirement of fault tolerance in turn makes it highly desirable to construct delay tolerant algorithms so that if an individual process fails the remaining processes are not forced to wait for it to recover. Delay tolerance in software design is desirable for other reasons as well since it minimizes the need for synchronization and facilitates latency hiding.

In a distributed implementation of this algorithm the coordinates of a vertex v_i must be transferred between computers if any edge which is incident on v_i is incident on another vertex v_j which is located on a different computer from v_i. This leads to message traffic on the interconnection network in a quantity proportional to the number of edges of $\mathcal{F}(\mathcal{G})$ which cross the boundaries of processor regions in Ω. If the implementation migrates vertices of $\mathcal{G}$ simultaneously with the evolution of the vertex coordinates then the amount of this traffic will reduce as locality increases among the vertices. Message traffic will be minimal in dynamic problem instances in which the algorithm is modifying an existing mapping in response to changes in $\mathcal{G}$ or $\mathcal{H}$.

This message traffic among computers provides a natural mechanism for fault tolerance since data will be replicated on computers which need it. If a computer or network component fails the only consequence will be that replicated data on functioning computers will not be updated as quickly as usual. Computers which have not failed can continue to execute the algorithm using stale replicated information. When the failed computer or network component returns to operation the stale information will be refreshed and the computation will proceed as before. The convergence of the algorithm will degrade gracefully under these conditions and will never fail catastrophically.

8. Simulations

Figures 6, 7 and 8 show simulations of the algorithm on the model problem.

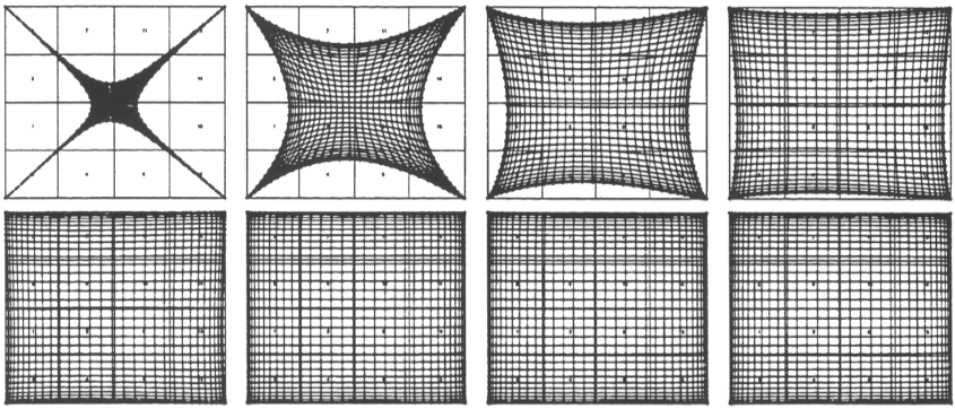

Figure 6: Evolution of $\mathcal{F}(\mathcal{G})$ for the model problem instance after 50, 100, 150, 200, 250, 300, 350 and 400 iterates of (5). All $v \in \mathrm{V}(\mathcal{G}) \backslash V_{ext}$ were initially placed at the center of Ω.

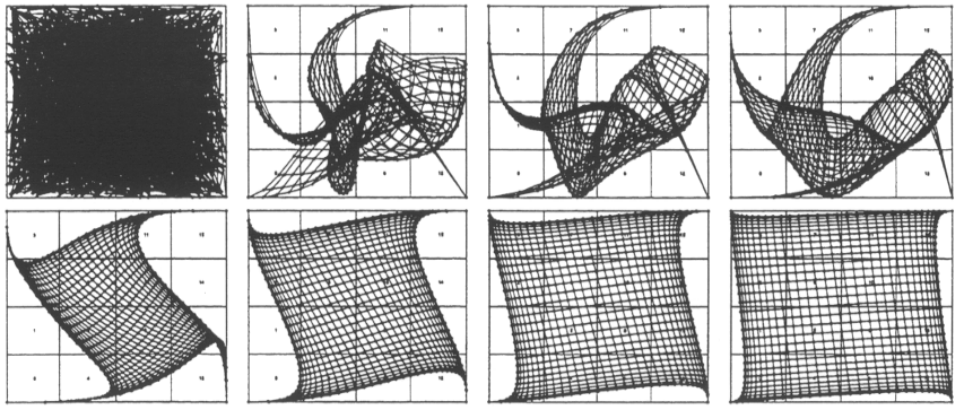

Figure 7: Evolution of $\mathcal{F}(\mathcal{G})$ for the model problem instance with random $a_{i,j}^{(0)}$ after 0, 25, 50, 75, 250, 500, 750, and 1000 iterates of (5).

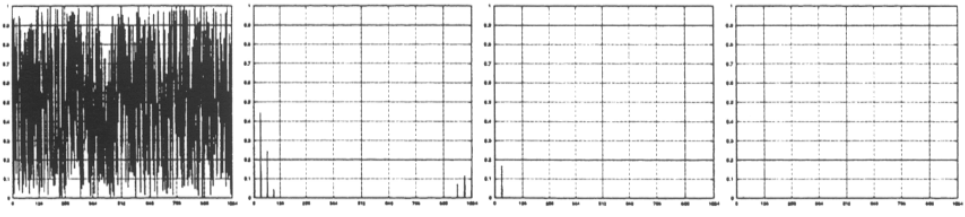

Figure 8: Spectral distribution $a_{i,j}$ after 0, 50, 250, and 1000 iterates.

$\{\forall v \in V(\mathcal{H}), \text{Load}(v) > 0\}$
do
$i := \text{Random}(1, |E(\mathcal{G})|);$
$e := e_i \in E(\mathcal{G});$
$v_1 := v_{e.left} \in V(\mathcal{G}); v_2 := v_{e.right} \in V(\mathcal{G});$
$r_1 := \text{Region}(v_1); r_2 := \text{Region}(v_2);$
$((r_1 \neq r_2) \wedge (\text{Load}(r_1) > \text{Load}(r_2))) \rightarrow \text{Remap}(v_1, r_1, r_2);$
$((r_1 \neq r_2) \wedge (\text{Load}(r_1) < \text{Load}(r_2))) \rightarrow \text{Remap}(v_1, r_2, r_1);$
od
$\{\forall v \in V(\mathcal{H}), \text{Load}(v) > 0 \wedge \text{Load}(v) = |V(\mathcal{G})|/|V(\mathcal{H})|\}$

Figure 9: A simple load balancing algorithm which preserves the locality of $\mathcal{G}$.

Figure 6 represents a case in which a set of processes are mapped *ab initio* onto an unloaded computer system. All of the vertices which are not in V_{ext} are initially assigned to locations in the center of Ω from which they rapidly diffuse until they fill Ω in a locality maximizing way. The partition of Ω into a 4×4 lattice is shown in the background. Any other partition of Ω to represent any other planar interconnection network would show the same rate of convergence as this example.

Figure 7 shows a malicious example in which the initial vertex locations are randomized. Although this example takes longer to converge it eventually approaches the same locality maximizing fixed point as the *ab initio* example does. Figure 8 shows that the linear iteration (5) rapidly annihilates components $\vec{X}_{i,j}$ until only a small number of low frequency components remain.

9. Efficient Mapping and Load Balancing

The continuous postcondition of the implementation guarantees that locality among vertices of $\mathcal{G}$ is maximal under $\mathcal{F}$. When $\mathcal{G}$ is planar and $\|\vec{\Delta}\|_2 = 0$ vertices of $\mathcal{G}$ are evenly distributed in Ω. If Ω is partitioned into regions of equal area an equal number of vertices will be assigned to each region and the workload will be balanced among the computers. If $\mathcal{G}$ is nonplanar this is not generally possible. Even when $\mathcal{G}$ is planar the algorithm is not the most efficient way to solve the combined problems of mapping and load balancing since the final convergence to the fixed point involves successively finer vertex movements (see figures 6 and 7).

The most efficient way to solve the combined problems of mapping and load balancing is to first solve the mapping problem to maximize locality and then separately solve the load balancing problem to equalize workload. If the mapping problem is solved first then it is trivial to adjust the workload by algorithms which only exchange vertices among neighboring regions of Ω. Since these algorithms use only neighbor exchanges they can transfer work without disrupting the locality maximizing ordering which represents the solution to the mapping problem. Workload diffusion algorithms[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30] are suitable for this purpose as is the even simpler algorithm of figure 9. This simple algorithm

was applied to the solution shown in the last frame of figure 6. After $5|E(\mathcal{G})|$ iterates the number of vertices was balanced to within one unit.

This algorithm finds edges of $\mathcal{G}$ which are mapped under $\mathcal{F}$ across adjacent regions of Ω. It moves vertices incident upon those edges across the region boundaries in order to equalize the number of vertices in each region. A workload is associated with each region in Ω corresponding to a vertex of $\mathcal{H}$. This workload is accessed by the function Load(v). The function Region(v) returns the index of the region association with $v \in V(\mathcal{G})$. The function Remap(v, r_1, r_2) moves a vertex from region r_1 to region r_2.

10. Summary and Discussion

This paper has presented a diffusion algorithm to solve the dynamic mapping and load balancing problems. Diffusion has previously produced efficient and infinitely scalable solutions to the load balancing problem. It has now done so for the mapping problem. In the authors experience this is the first time the dynamic mapping problem has been addressed and the first time a diffusive solution has been proposed to the mapping problem. The properties of the algorithm have been argued by analysis and demonstrated in simulation. The method has been shown to be correct in the presence of delays and nondeterministic execution order and to degrade gracefully in the presence of faults and failures. A simple load balancing algorithm has been presented which makes the combined algorithm for mapping and load balancing efficient. An application of these concepts is currently in progress.[30]

Simple extensions of this diffusion algorithm allow it to solve problems in which both the computers and the processes are weighted nonuniformly. Nonuniform weights can be assigned to computers simply by partitioning Ω into regions of nonuniform area. Weights can be assigned to processes by representing vertices as circles with radius proportional to the process weight. If edge lengths are measured from the perimeter of these circles then the algorithm will take these nonuniformly distributed weights into consideration in a natural way. A closely related alternative algorithm can be constructed by averaging the angles of incident edges at each vertex rather than coordinates in Ω. This has an advantage in that it does not require constraints on the extremal vertices.

The diffusion algorithm adapts naturally to an environment in which computers appear and disappear during the course of a computation. To introduce a new computer into the system it is only necessary to allocate a new region within the existing Ω and assign to the new computer the vertices in this region. To remove an existing computer it is only necessary to merge an existing region with its neighbors.

It is worth noting that this diffusion algorithm bears a fundamental relationship to the celebrated spectral bisection algorithms which are currently popular on MPPs. Spectral bisection algorithms have gained respect because they are rooted in the spectral characteristics of the Laplacian matrix $Q(\mathcal{G})$. They work by recursively bisecting $\mathcal{G}$ according to the components of the eigenvector of $Q(\mathcal{G})$ which corresponds to the second smallest eigenvalue. Although these bisections produce high quality solutions the cost of obtaining this eigenvector can be very high.

The diffusion algorithm presented in this paper is based on spectral characteristics of $Q(\mathcal{G})$ but involves a considerably less expensive computation. The algorithm seeks a placement of vertices of $\mathcal{G}$ which yields a uniform distribution of edge lengths. This uniform distribution is the eigenvector of $Q(\mathcal{G})$ which corresponds to the smallest eigenvalue. It is obtained by solving a simple Laplace system $\nabla^2 u = 0$. In this respect the insight behind this diffusion algorithm has been anticipated in a classical algorithm of Tutte.[39]

The topic of diffusion in concurrent computation was first addressed in the paradigm of *diffusing computations*.[11,40,41,42] While this paradigm is not precisely the same as the notion of *diffusion algorithm* described in this paper there are striking similarities. Some of the same problems addressed by diffusing computations are addressed by diffusion algorithms, such as mapping and load balancing.[11] Diffusion algorithms can address a number of problems of a quadratic nature including the shortest paths problem, which has also been addressed by diffusing computations.[42]

In a prescient statement Cybenko[16] describes his own work as "the first rigorous treatment of dynamic load balancing strategies [which can] serve as a basis for future development and research into the subject". It is the authors hope that this paper, inspired by work[26,27] which was anticipated by Cybenko, may serve as the next step in exploring the space of diffusion algorithms. All such algorithms inherit the properties of robustness, dynamic responsiveness, scalability, delay and fault tolerance which have been described in this paper. As a result it is worth exploring the myriad ways in which this algorithmic paradigm can be exploited.

11. Acknowledgements

This paper has benefitted from discussions with Eric van de Velde, Joel Franklin, and Herb Keller, and from a careful reading by Harm Peter Hofstee. The author appreciates the encouragement to pursue this work given by George Cybenko, Jan van de Snepscheut, and Mani Chandy. The work was carried out while the author was enjoying the hospitality of Caltech's Center for Advanced Computing Research. The author was supported by the C.A.C.R. and by the N.S.F. Center for Research in Parallel Computation.

References

1. N. Aboulenen, S. Gjessing, J. Goodman and P. Woest, "Hardware support for synchronization in the Scalable Coherent Interface (SCI)", *Proc. 8th I.E.E.E. Inter. Par. Proc. Symp.* (1994).
2. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and W. Su, "Myrinet: a gigabit per second local area network", *I.E.E.E. Micro* 15 (1995) 29–36.
3. T. Sterling, D. J. Becker, J. E. Dorband, D. Savarese, U. A. Ranawake and C. V. Packer, "Beowulf: a parallel workstation for scientific computation", *Proc. 24th Intern. Conf. Par. Proc.*, I (1995) pp. 11–14.
4. T. E. Anderson, D. E. Culler and D. A. Patterson, "A Case for NOW (Networks of Workstations)", *I.E.E.E. Micro* 15 (1995) 54–64.

5. T. Sterling, "The scientific workstation of the future may be a pile-of-pcs", To appear in *Comm. ACM* (1996).
6. H. T. Kung and D. Stevenson, "A software technique for reducing the routing time on a parallel computer with a fixed interconnection network", In *High Speed Computer and Algorithm Organization*, eds. D. J. Kuck, D. H. Lawrie and A. H. Sameh (1977) pp. 423–433.
7. J. A. Ortega, *Introduction to parallel and vector solution of linear systems* (Plenum, New York, 1988).
8. D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings and O. Zone, "A retrofit methodology for the fast generation and optimization of large-scale mesh partitions: beyond the minimum interface size criterion", Technical report CU-CAS-94-18, Center for Aerospace Structures, University of Colorado, Boulder, 1994.
9. D. Vanderstraeten, R. Keunings and C. Farhat, "Beyond conventional mesh partitioning algorithms and the minimum edge cut criterion: impact on realistic applications", *Proc. 7th SIAM Conf. Par. Proc. Sci. Comp.* (1995) pp. 611–614.
10. S. H. Bokhari, "On the Mapping Problem", *I.E.E.E. Trans. Comput.* **C-30** (1981) 550–557.
11. A. Martin, "A distributed implementation method for parallel programming", *Inf. Proc.* **80** (1980) 309–314.
12. S. T. Barnard and H. Simon, "A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes", *Proc. 7th SIAM Conf. Par. Proc. Sci. Comp.* (1995) pp. 627–632.
13. R. D. Williams, "Performance of dynamic load balancing algorithms for unstructured mesh calculations", *Concurrency: Pract. Exp.* **3** (1991) 457–481.
14. F. C. H. Lin and R. M. Keller, "The gradient model load balancing method", *I.E.E.E. Trans. Soft. Eng.* **SE-13** (1987) 32–38.
15. J. Hong, X. Tan and M. Chen, "From local to global: an analysis of nearest neighbor balancing on hypercube", *Proc. ACM Sigmetric Conf. on Measurement and Modeling of Comp. Sys.* (1988) pp. 73–82.
16. G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors", *J. Par. Distrib. Comp.* **7** (1989) 279–301.
17. J. E. Boillat, "Load balancing and Poisson equation in a graph", *Concurrency: Pract. Exp.* **2** (1990) 289–313.
18. F. Brugé and S. L. Fornili, "A distributed dynamic load balancer and it's implementation on multi-transputer systems for molecular dynamics simulation", *Comp. Phys. Comm.* **60** (1990) 39–45.
19. S. H. Hosseini, B. Litow, M. Malwaki, S. Nadella and K. Vairavan, "Analysis of a graph coloring based distributed load balancing algorithm", *J. Par. Dist. Comp.* **10** (1990) 160–166.
20. J. E. Boillat, F. Brugé and P. G. Kropf, "A dynamic load balancing algorithm for molecular dynamics simulation on multiprocessor systems", *J. Comp. Phys.* **96** (1991) 1–14.
21. S. B. Baden, "Programming abstractions for dynamically partitioning and coordinating localized scientific calculations running on multiprocessors", *SIAM J. Sci. Stat. Comp.* **12** (1991) 145–157.
22. C. Z. Xu and F. C. M. Lau, "Analysis of the generalized dimension exchange method for dynamic load balancing", *J. Par. Dist. Comp.* **16** (1992) 385–393.

23. A. J. Conley, "Using a transfer function to describe the load balancing problem", Argonne National Laboratory report ANL-93/40 (1993).
24. G. Horton, "A multi-level diffusion method for dynamic load balancing", *Par. Comp.* **19** (1993) 209–218.
25. M. Willabeek-LeMair and A. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers", I.E.E.E. Tr. Par. Dist. Sys. **4** (1993) 979–993.
26. A. Heirich, "Scalable load balancing by diffusion", Technical report CS-TR-94-04, Department of Computer Science, California Institute of Technology (1994).
27. A. Heirich and S. Taylor, "A parabolic load balancing method", *Proc. 24th Intern. Conf. Par. Proc.* **III** (1995) pp. 192–202.
28. C. Z. Xu and F. C. M. Lau, "The generalized dimension exchange method for load balancing in k-ary n-cubes and variants", *J. Par. Dist. Comp.* **24** (1995) 72–85.
29. F. J. Muniz and E. J. Zaluska, "Parallel load balancing: an extension to the gradient model", *Par. Comp.* **21** (1995) 287–301.
30. A. Heirich and J. Arvo, "An approach to scalable photorealistic rendering", To appear in *Proc. First Eurographics Workshop on Parallel Graphics and Visualization* (1996).
31. M. R. Garey and D. S. Johnson, *Computers and intractibility: a guide to the theory of NP-completeness* (Freeman, San Francisco, 1980).
32. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken, "LogP: towards a realistic model of parallel computation", *SIGPLAN notices* **28** (1993) 1-12.
33. C. A. R. Hoare, "Communicating sequential processes", *Comm. ACM* **21** (1978) 666-677.
34. E. W. Dijkstra, *A discipline of programming* (Prentice-Hall, Englewood, NJ, 1976).
35. K. M. Chandy and J. Misra, *Parallel program design: a foundation* (Addison-Wesley, Reading, MA, 1988).
36. R. A. Horn and C. R. Johnson, *Matrix Analysis* (Cambridge University Press, New York, 1991).
37. D. M. Young, *Iterative solution of large linear systems* (Academic Press, New York, 1971).
38. B. Mohar, "The Laplacian spectrum of graphs", In *Graph theory: combinatorics and applications*, eds. Y. Alavi et al (Wiley, New York, 1988) pp. 871-898.
39. W. T. Tutte, "How to draw a graph", *Proc. London Math. Soc.* **13** (1963) 743–768.
40. E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations", *Inf. Proc. Lett.* **11** (1980) 1-14.
41. K. M. Chandy and J. Misra, "Termination detection of diffusing computations in communicating sequential processes", *ACM Tr. Prog. Lang. and Sys.* **4** (1982).
42. K. M. Chandy and J. Misra, "Distributed Computation on Graphs: Shortest Path Algorithms", *Comm. ACM* **25** (1982) 833-837.