The Optimal Property of Polynomial Based Diffusion-Like Algorithms in Dynamic Load Balancing

Y. F. Hu and **R. J. Blake** [1]

Abstract. Diffusion type algorithms [1, 3, 13] are some of the most popular algorithms for scheduling in dynamic load balancing. In the paper it is proved that all diffusion-like algorithm has an interesting minimum norm property in the sense that the weighted Euclidean norm of the amount of load migration generated by the algorithm is minimized.

1 INTRODUCTION

One of the most important issue in the efficient use of parallel computers is that of load balancing. For many applications the load changes during the course of computation, making it necessary to balance the load dynamicly, and in parallel.

Most of the existing parallel dynamic load balancing algorithms [14, 4, 12] involve two steps:

- "flow" calculation: finding out the amount of load to be migrated between neighboring processors, such that a uniform load distribution will be achieved when the migration is carried out to satisfy the "flow".
- migration: deciding which particular tasks are to be migrated, and migrating these tasks to the appropriate neighboring processors.

This paper is concerned with algorithms for the first step (but see [14, 4, 12] for possible strategies of the second step). Diffusion type algorithms [1, 3, 13] are some of the most popular ones for flow calculations, although there are a number of other algorithms [9, 3, 15, 16, 8].

[1] Daresbury Laboratory, Daresbury, Warrington WA4 4AD, United Kingdom, Tel. +44 (0)1925 603362, Fax +44 (0)1925 603634, e-mail: Y.F.Hu@dl.ac.uk

© 1998 Y. F. Hu and R. J. Blake
ECCOMAS 98.
Published in 1998 by John Wiley & Sons, Ltd.

To illustrate the "flow" calculation step, Figure 1 (a) shows a processor graph, the load on one of the processor is 32, twice higher than that of the others. Figure 1 (d) shows the "flow" along each edge of the processor graph (rounded to integers in brackets), calculated using 3 iterations of a diffusion algorithm. Figure 1 (e) show the load of each processor after the flow is satisfied.

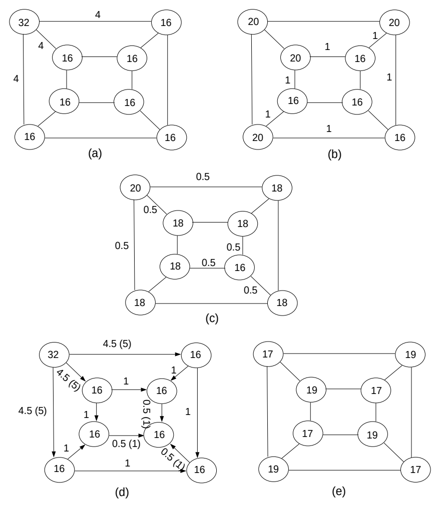

Figure 1. (a) the first iteration of the diffusion algorithm on a processor graph with load imbalance; (b) the second iteration of the diffusion algorithm; (c) the third iteration of the diffusion algorithm; (d) the "flow" along each edge after 3 iterations (figures in the brackets are rounded to integer); (e) the load of each processor after the "flow" is satisfied

The "flow" calculation problem usually has many solutions. To minimize the communication cost, it is important to choose a solution that involves as little load migration as possible. In [9], an algorithm based on minimizing the Euclidean norm of the amount of load migration was introduced. It was demonstrated to converge much faster than the diffusion algorithm of [1, 3]. However, numerical experiment [9] shown, some what surprisingly, that the Euclidean norm of the "flow" produced by the diffusion algorithm were very close to those of the new algorithm. It was believed that the diffusion algorithm may also satisfy some optimal property. In this paper this conjecture is proved for a general class of diffusion algorithm based on the polynomial of the weighted Laplacian matrix.

The conventional diffusion algorithm [1, 3] and a more efficient new algorithm [9] are introduced. The generalise diffusion-like algorithm based on the polynomial of the weighted Laplacian matrix is then presented. Finally the optimal property of the diffusion-like algorithm is proved.

1 SOME "FLOW" ALGORITHMS

Notations

Let $\mathbf{p}$ be the number of processors. Denote the processor graph by the graph $(\mathbf{V}, \mathbf{E})$, with $|\mathbf{V}| = \mathbf{p}$ vertices and $|\mathbf{E}|$ edges. Each of the vertices $\mathbf{V} = (1,2,\ldots,\mathbf{p})$ represents a processor, and $\mathbf{E}$ is the set of edges. The graph is assumed to be connected. Denoting $\mathbf{i} \leftrightarrow \mathbf{j}$ if vertices $\mathbf{i}$ and $\mathbf{j}$ form an edge of the processor graph. Associated with each processor $\mathbf{i}$ is a scaler $\mathbf{l_i}$ representing the load on the processor. The average load per processor is

$$\bar{\mathbf{l}} = \frac{\sum_{\mathbf{i}=1}^{\mathbf{p}} \mathbf{l_i}}{\mathbf{p}}.$$

Each edge has a direction associated, say from the vertex with smaller index to the vertex with larger index. The former will be called the *head* and the latter the *tail* of the edge. The amount of load to be transferred on the $\mathbf{i}$-th edge (along the direction of the edge) is denoted as $\delta_{\mathbf{i}}$.

Denote

$$\mathbf{b} = (\mathbf{l}_1 - \bar{\mathbf{l}}, \mathbf{l}_2 - \bar{\mathbf{l}}, \ldots, \mathbf{l_p} - \bar{\mathbf{l}})^{\mathbf{T}} \tag{1}$$

the vector of load imbalance, and

$$\mathbf{x} = (\delta_1, \delta_2, \ldots, \delta_{|\mathbf{E}|})^{\mathbf{T}}$$

the vector of "flow" along each of the edges.

The vertex-edge incident matrix $\mathbf{A}$ [11], which is of the dimension $|\mathbf{V}| \times |\mathbf{E}|$, is defined as

$$(\mathbf{A})_{\mathbf{i},\mathbf{j}} = \begin{cases} 1, & \text{if vertex } \mathbf{i} \text{ is the head of edge } \mathbf{j}, \\ -1, & \text{if vertex } \mathbf{i} \text{ is the tail of edge } \mathbf{j}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

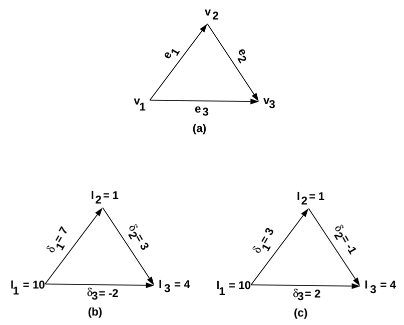

Figure 2. (a) a graph; (b) a load migration scheme; (c) another scheme

For example, for the graph in Figure 2 (a), the vertex-edge incident matrix is

$$\begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & -1 \end{pmatrix}. \tag{3}$$

Definition 1 *The "flow" calculation problem for dynamic load balancing problem is that of finding a schedule which gives the amount of load $\delta_{\mathbf{k}}$ to be transferred along any edge $\mathbf{k}$ (also called the "flow" along the edge $\mathbf{k}$), such that after the transfer, the load of each processor will be the same. In other words,*

$$\mathbf{l_i} - \sum_{\mathbf{i} \leftrightarrow \mathbf{j}} \delta_{\mathbf{ij}} = \bar{\mathbf{l}}_{\mathbf{i}}, \quad \mathbf{i}, \mathbf{j} \in \mathbf{V},$$

or

$$\mathbf{Ax} = \mathbf{b}. \tag{4}$$

Here $\delta_{\mathbf{ij}}$ denotes the "flow" from vertex $\mathbf{i}$ to vertex $\mathbf{j}$.

For example, for the graph in Figure 2 (a), the matrix $\mathbf{A}$ is given by (3), thus equation (4) becomes

$$\begin{aligned} \delta_1 + \delta_3 &= \mathbf{l}_1 - \bar{\mathbf{l}}, \\ -\delta_1 + \delta_2 &= \mathbf{l}_2 - \bar{\mathbf{l}}, \\ -\delta_1 - \delta_3 &= \mathbf{l}_3 - \bar{\mathbf{l}}. \end{aligned}$$

The vertex-edge incident matrix $\mathbf{A}$ is of the dimension $|\mathbf{V}| \times |\mathbf{E}|$. Because there are usually more edges than vertices in a graph, the linear system (4) is likely to have many solutions. For instance consider the graph in Figure 2, with the load as specified in the Figure. The schedules given by both Figure 2 (b) and Figure 2 (c) are valid. But schedule (c) is preferred as it involves less data movement.

The diffusion algorithm

The diffusion algorithm, as presented in [1, 3], is introduced as follows. Each iteration, a vertex sends load to its neighbors, the amount of which being proportional to the difference of its load and the neighbors' load.

Notice that this load migration is not actually carried out. Rather, it is recorded and the accumulated load migration along each edge is used as the "flow" on convergence of the diffusion algorithm. This is because the diffusion algorithm may take many iterations to converge and to carry out an actual migration of load each iteration would incur a large amount of communication and book keeping cost. Separating the "flow" calculation from the actual migration of load would be more economical.

At iteration $\mathbf{k}+1$ of the diffusion algorithm, the new load $\mathbf{l}_i^{(k+1)}$ of a vertex $\mathbf{i}$ is given by the combination of its load at the iteration $\mathbf{k}$, and those of its neighboring vertices, namely

$$\mathbf{l}_i^{(k+1)} = \mathbf{l}_i^{(k)} - \sum_{i \leftrightarrow j} \mathbf{c}_{ij}(\mathbf{l}_i^{(k)} - \mathbf{l}_j^{(k)}), \quad \mathbf{i},\mathbf{j} \in \mathbf{V}. \quad (5)$$

Here $\mathbf{c}_{ij}$ is a weight associated with the edge $(\mathbf{i},\mathbf{j})$.

Initially the load for vertex $\mathbf{i} \in \mathbf{V}$ is $\mathbf{l}_i^{(1)} = \mathbf{l}_i$. One advantage of the diffusion algorithm is that every processor only needs to know the load of its neighboring processor. No global communication is necessary.

For a choice of the weights in (5), Boillat [1] suggested

$$\mathbf{c}_{ij} = \frac{1}{\max\{\mathbf{deg}(\mathbf{i}),\mathbf{deg}(\mathbf{j})\}+1}, \quad \mathbf{i} \leftrightarrow \mathbf{j}, \quad \mathbf{i},\mathbf{j} \in \mathbf{V},$$

where $\mathbf{deg}(\mathbf{i})$ is the degree of vertex $\mathbf{i}$, defined as the number of edges connected to the vertex.

For example, because each vertex of the graph in Figure 1 (a) is of degree 3, the weight of each edge is $1/(1+3) = 0.25$. At the first iteration, the processor with the load of 32 will send to each of its neighbors a load of $0.25 * (32 - 16) = 4$. Figure 1 (b) shows the processor graph at the second iteration, and Figure 1 (c) the third iteration. The accumulated "flow" of these three iterations are shown in Figure 1 (d), where the "flow" rounded to the nearest integer is shown in brackets. The resulting load of each processor, after this integer "flow" is satisfied, is shown in Figure 1 (e).

Defining the *induced graph* as the original graph but with edges $(\mathbf{i},\mathbf{j})$ removed if the weight $\mathbf{c}_{ij} = 0$, it can be proved [3] that the above diffusion algorithm will converge to the uniform distribution if the following assumption holds.

Assumption 2 *The diffusion algorithm (5) is assumed to satisfy the following:*

- $\mathbf{c}_{ij} \geq 0$, for $\mathbf{i} \leftrightarrow \mathbf{j}$;
- $\sum_{\mathbf{j}:\, \mathbf{i} \leftrightarrow \mathbf{j}} \mathbf{c}_{ij} < 1$, for $\mathbf{i} \in \mathbf{V}$;
- $\mathbf{c}_{ij} = \mathbf{c}_{ji}$;
- *the induced graph is connected.*

The matrix form of the diffusion algorithm

Let

$$\mathbf{y}^{(k)} = (\mathbf{l}_1^{(k)}, \mathbf{l}_2^{(k)}, \ldots, \mathbf{l}_{|V|}^{(k)})^T, \quad \mathbf{k} = 1,2,\ldots$$

be the vector of load at iteration $\mathbf{k}$. The diffusion algorithm (5) can be stated in matrix form as

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} - \mathbf{L}\mathbf{y}^{(k)} = (1 - \mathbf{L})\,\mathbf{y}^{(k)}, \quad (6)$$

where $\mathbf{L}$ is the (weighted) Laplacian matrix, a $|\mathbf{V}| \times |\mathbf{V}|$ symmetric matrix given by

$$(\mathbf{L})_{ij} = \begin{cases} -\mathbf{c}_{ij}, & \text{if } \mathbf{i} \neq \mathbf{j}, \mathbf{i} \leftrightarrow \mathbf{j}, \\ \sum_{i \leftrightarrow k} \mathbf{c}_{ik}, & \text{if } \mathbf{i} = \mathbf{j}, \\ 0, & \text{otherwise}. \end{cases}$$

Now defining the *edge-weight matrix* $\mathbf{W}$ as the diagonal matrix of dimension $|\mathbf{E}| \times |\mathbf{E}|$, with the $\mathbf{i}$-th diagonal element equals to the weight $\mathbf{c}$ associated with the $\mathbf{i}$-th edge. Then it can be confirmed that the Laplacian matrix $\mathbf{L}$ can be expressed as the product of vertex-edge incident matrix and the edge-weight matrix, that is,

$$\mathbf{L} = \mathbf{A}\,\mathbf{W}\,\mathbf{A}^T. \quad (7)$$

For example, for the graph of Figure 2, one has

$$\begin{aligned} \mathbf{A}\,\mathbf{W}\,\mathbf{A}^T &= \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} \mathbf{c}_1 & 0 & 0 \\ 0 & \mathbf{c}_2 & 0 \\ 0 & 0 & \mathbf{c}_3 \end{pmatrix} \\ & \quad \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{c}_1+\mathbf{c}_3 & -\mathbf{c}_1 & -\mathbf{c}_3 \\ -\mathbf{c}_1 & \mathbf{c}_1+\mathbf{c}_2 & -\mathbf{c}_2 \\ -\mathbf{c}_3 & -\mathbf{c}_2 & \mathbf{c}_1+\mathbf{c}_2 \end{pmatrix} \\ &= \mathbf{L}. \end{aligned}$$

A more efficient algorithm [9]

Assuming that the Euclidean norm of the data movement is used as a metric, then to minimise the data movement, the following problem need to be solved

$$\text{Minimise } \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{W}^{-1}\mathbf{x}, \quad \text{subject to } \quad \mathbf{Ax} = \mathbf{b}.$$

Here $\mathbf{W}$ is a positive diagnol weight matrix.

Applying the necessary condition for the constrained optimization problem (see [5]) gives

$$\mathbf{x} = \mathbf{WA}^{\mathbf{T}}\mathbf{d}, \tag{8}$$

where $\mathbf{d}$ is the vector of Lagrange multipliers. Substituting (8) into $\mathbf{Ax} = \mathbf{b}$ gives

$$\mathbf{L}\,\mathbf{d} \;=\; \mathbf{b}, \tag{9}$$

with $\mathbf{L} = \mathbf{AWA}^{\mathbf{T}}$ the weight Laplacian matrix.

Thus the problem of finding an optimal load balancing schedule is transformed to that of solving the linear equation (9). Once the Lagrange multipliers are found, then the load transfer vector is $\mathbf{x} = \mathbf{A}^{\mathbf{T}}\mathbf{d}$. For any graph, each row $\mathbf{i}$ of the matrix $\mathbf{A}^{\mathbf{T}}$ only has two nonzero, $\mathbf{1}$ and $-\mathbf{1}$, corresponding to the head and tail vertices of the edge $\mathbf{e_i}$. Therefore the amount of load to be transferred from processor $\mathbf{j_1}$ to processor $\mathbf{j_2}$ (assuming $\mathbf{j_1}$ is the head and $\mathbf{j_2}$ the tail), along the edge $\mathbf{e_i} = (\mathbf{j_1},\mathbf{j_2})$, is simply

$$\delta_{\mathbf{i}} = \mathbf{d_{j_1}} - \mathbf{d_{j_2}},$$

where $\mathbf{d_{j_1}}$ and $\mathbf{d_{j_2}}$ are the Lagrange multipliers associated with vertices $\mathbf{j_1}$ and $\mathbf{j_2}$ respectively.

Therefore the new load balancing algorithm is:

- a) find the average work load, and the load imbalance vector $\mathbf{b}$.
- b) solve linear equation (9) for $\mathbf{d}$, with $\mathbf{L}$ the Laplacian matrix;
- c) the "flow" between processor $\mathbf{j_1}$ and $\mathbf{j_2}$ is $\mathbf{d_{j_1}} - \mathbf{d_{j_2}}$.

If we call the vector $\mathbf{d}$ the vector of potential, then Step c) of the above algorithm says that the "flow" between two processors are the difference of their potential!

The linear system (9) can be solved by many standard numerical algorithms. The conjugate gradient algorithm [6] was used in [9]. The algorithm is simple, easy to parallelise and converges fast. For preconditioning the diagonal of the Laplacian can be used.

It is well known (see, e.g., [2]) that the Laplacian matrix is positive semi-definite. It has an eigenvalue of zero associated with the eigenvector of all ones, and if the graph is connected, the rest of the eigenvalues are all positive. The conjugate gradient algorithm will converge in less than $\mathbf{k}$ iterations, where $\mathbf{k}$ is the number of distinctive positive eigenvalues of matrix $\mathbf{L}$ (see [5] for the theory of conjugate gradient algorithm). Clearly $\mathbf{k} \leq \mathbf{p} - \mathbf{1}.$

In fact $\mathbf{p} - \mathbf{1}$ is a pessimistic estimate of the iteration numbers needed for the new algorithm (combined with the conjugate gradient) to converge. For many graph with rich connectivity the algorithm converges a lot faster. Some analysis of the algorithm on many regular graphs (meshes, hypercubes etc.) can be found in [9].

The following table compares this algorithm with the diffusion algorithm, where "D" stands for the diffusion algorithm and "P" stands for the above algorithm. They were tested on 5 random graphs of 256 vertices, with average degree ranging from 2 to 9. The result from a Cray T3D clearly indicated that the diffusion algorithm was not efficient for graphs of low connectivity (small degree), while the new algorithm performed well on all graphs. The is also interesting to noted from Table 1 that although the new algorithm was design to minimize the norm of the "flow", in reality the Euclidean norm of both algorithm are close to each other, indicating that the diffusion algorithm may also has some optimal property. This conjecture will be proved in the next section in a more general form. In fact from Table 1, in some case the diffusion algorithm even has a smaller norm, this abnormality is due to the fact that the norm in this table is the sum of the squares of the flow along each edge after they are rounded to integer. When the norm of the floating point "flow" is considered, the new algorithm does always give a smaller or equal norm.

Table 1 Comparing the new algorithm with the diffusion algorithm on the 256 processors of a Cray T3D

method	diameter	degree	iterations	time(ms)	norm
P	152.70	2.00	222	218.4	106867
D	152.70	2.00	40044	11884.4	105003
P	12.40	3.11	33	33.3	23190
D	12.40	3.11	201	73.9	23274
P	7.30	5.01	17	20.6	15085
D	7.30	5.01	75	34.0	15201
P	5.50	7.00	15	18.7	11506
D	5.50	7.00	42	23.8	11594
P	4.30	9.00	12	16.3	9749
D	4.30	9.00	33	20.7	9812

3 AN OPTIMAL PROPERTY OF THE DIFFUSION-LIKE ALGORITHM

It is seen in Section 1 that the "flow" calculation problem usually has more than one solutions. From the point of view of minimizing the communication cost, a solution that involves small amount of load migration (such as that of Figure 2 (b)) is clearly preferable to one that involves a lot of load migration (such as that of Figure 2 (c)).

As defined in Section 1, let $\mathbf{x}$ be the vector of load migrations. To minimize the amount of load migration and achieve load balance at the same time, it is necessary to find a "flow" solution $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{b}$ and the norm of $\mathbf{x}$, as a measure of total load migration, is minimized. It shall be prove in this section that the "flow" given by the diffusion algorithm in fact minimizes the weighted Euclidean norm of the load migration, $\mathbf{x^T W^{-1} x}$, with $\mathbf{W}$ the edge-weight matrix defined in Section 2.

General diffusion-like algorithms

From (6), using the diffusion algorithm, each iteration the load can be expressed as

$$\mathbf{y^{(k+1)}} = \mathbf{(I - L)^k y^{(1)}}, \quad \mathbf{k = 1,2,\ldots} \tag{10}$$

At iteration $\mathbf{k{+}1}$ of the diffusion algorithm, the amount of load transferred, from vertex $\mathbf{j_1}$ to vertex $\mathbf{j_2}$, equals to the load difference between the two vertices, scaled by the edge weight. That is,

$$\delta_{\mathbf{i}}^{\mathbf{(k)}} = \mathbf{c_{j_1 j_2}} * (\mathbf{l_{j_1}^{(k)}} - \mathbf{l_{j_2}^{(k)}}).$$

In matrix form this is

$$\mathbf{x^{(k)}} = \mathbf{W\ A^T\ y^{(k)}}, \tag{11}$$

where

$$\mathbf{x^{(k)}} = (\delta_1^{\mathbf{(k)}}, \delta_2^{\mathbf{(k)}}, \ldots, \delta_{|\mathbf{E}|}^{\mathbf{(k)}})^{\mathbf{T}}$$

is the vector of migrating load along all the edges at iteration $\mathbf{k}$.

In the easy to confirm, using (11) and (7), that the difference of the load at two subsequent iteration equals to the vector of migrating load multiplied by the vertex-edge incident matrix:

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(k)}} = \mathbf{Ax^{(k)}} \tag{12}$$

We define the diffusion-like algorithm to be

$$\mathbf{y^{(k+1)}} = \mathbf{p_k(L) y^{(1)}}, \quad \mathbf{k = 1,2,\ldots} \tag{13}$$

where $\mathbf{p_k(x)}$ is the $\mathbf{k}$-th order polynomial, and $\mathbf{p_k(0) = 1}$. This is clearly a generalisation of (10), and there is a good reason in this generalisation. By utilisaing this general formula, it is possible to construct diffusion-like algorithm which converges faster than the orginal diffusion algorithm, yet retains the nice property of the diffusion algorithm in that only nearest neighbor communication is needed [10].

Similar to (12), the vector of load migration, $\mathbf{x^{(k)}}$, for the generalised diffusion algorithm (13) can be derived as follows:

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(k)}} = (\mathbf{p_k(L)} - \mathbf{p_{k-1}(L)})\mathbf{p_k(L) y^{(1)}}$$

Because the polynomials satisfies $\mathbf{p_k(1) = 1}$, the difference between them can be expressed as

$$\mathbf{p_k(L)} - \mathbf{p_{k-1}(L)} = \mathbf{L q_{k-1}(L)}$$

where $\mathbf{q_{k-1}}$ is a polynomial of order $\mathbf{k-1}$. Define the vector of load migration to be

$$\mathbf{x^{(k)}} = \mathbf{W A^T q_{k-1}(L) y^{(1)}},$$

then equation (12) is again satisfied.

The optimal property of the diffusion-like algorithms

The accumulated load migration (the "flow") given by the diffusion-like algorithm is

$$\sum_{\mathbf{k=1}}^{\infty} \mathbf{x^{(k)}}$$

It was proved [3] that under assumption (2), the diffusion algorithm will converge to the uniform load.

If we assume that the diffusion-like algorithm (13) also converges ($\mathbf{y^{(k)}} \to \mathbf{y}$), then it is possible to prove that the "flow" given by this general diffusion-like algorithm is optimal in the following sense:

Theorem 3 *The load migration scheme generated by the diffusion-like algorithm is the solution of the following minimization problem*

$$\begin{array}{ll} \text{Minimise} & \frac{1}{2}\mathbf{x^T W^{-1} x}, \\ \text{subject to} & \mathbf{Ax = b}. \end{array} \tag{14}$$

Here it is assumed that the weight associated with each edge is positive so that $\mathbf{W^{-1}}$ exists. The matrix $\mathbf{A}$ is the vertex-edge incident matrix defined in (2) and the vector $\mathbf{b}$ is the vector of load imbalance (1).

Proof

We first prove that the accumulated amount of load transfer.

Because

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(k)}} = \mathbf{L q_{k-1}(L) y^{(1)}}$$

Define $\mathbf{d^{(k)}} = \sum_{i=1}^{k} \mathbf{q_{i-1}(L) y^{(1)}}$ as the *potential vector*, then

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(1)}} = \mathbf{L d^{(k)}}$$

Let $\mathbf{\bar{d}^{(k)}} = \mathbf{d^{(k)}} - (\mathbf{e^T d^{(k)}})\mathbf{e}$ be the normalized potential vector (with a sum of zero), where $\mathbf{e}$ is the vector of all ones, then

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(1)}} = \mathbf{L \bar{d}^{(k)}} \tag{15}$$

Expand $\mathbf{\bar{d}^{(k)}}$ using the normalized eigenvectors of $\mathbf{L}$, because $\mathbf{\bar{d}^{(k)}}$ is orthogonal to the eigenvector $\mathbf{u_1} = \mathbf{e}$,

$$\mathbf{\bar{d}^{(k)}} = \sum_{i=2}^{|V|} \mathbf{a_i^{(k)} u_i}.$$

Substituting to (15) gives

$$\mathbf{y^{(k+1)}} - \mathbf{y^{(1)}} = \sum_{i=2}^{|V|} \lambda_i \mathbf{a_i^{(k)} u_i}.$$

Thus

$$||\mathbf{\bar{d}^{(k)}}|| = \sum_{i=2}^{|V|} \left(\mathbf{a_i^{(k)}}\right)^2 \le ||\mathbf{y^{(k+1)}} - \mathbf{y^{(1)}}||/\lambda_2.$$

Since $\mathbf{y^{(k+1)}}$ converged to the uniform load, it follows that $||\mathbf{\bar{d}^{(k)}}||$ is bounded. So there exists a subsequence $\{\mathbf{k_i}\}$ such that

$$\mathbf{\bar{d}^{(k_i)}} \to \mathbf{\bar{d}}.$$

Because of (15), $\mathbf{L\bar{d}} = \mathbf{y^{(0)}} - \mathbf{\bar{l}e}$, so

$$\mathbf{L}\left(\mathbf{\bar{d}^{(k)}} - \mathbf{\bar{d}}\right) = \mathbf{y^{(k)}} - \mathbf{\bar{l}e} \to 0.$$

It can now be proved that $\mathbf{\bar{d}^{(k)}}$ converges to $\mathbf{\bar{d}}$, as follows. The vector $\mathbf{\bar{d}^{(k)}} - \mathbf{\bar{d}}$ is orthogonal to the eigenvector $\mathbf{u_1} = \mathbf{e}$. Expanding $\mathbf{\bar{d}^{(k)}} - \mathbf{\bar{d}}$ using the rest of the eigenvectors of $\mathbf{L}$ gives

$$\mathbf{\bar{d}^{(k)}} - \mathbf{\bar{d}} = \sum_{i=2}^{|V|} \mathbf{b_i^{(k)} u_i}.$$

Multiplying both side with $\mathbf{L}$, it follows that

$$\sum_{i=2}^{|V|} \mathbf{b_i^{(k)}} \lambda_i \mathbf{u_i} = \mathbf{L}\left(\mathbf{\bar{d}^{(k)}} - \mathbf{\bar{d}}\right) \to 0.$$

Therefore

$$\sum_{i=2}^{|V|} \left(\mathbf{b_i^{(k)}}\right)^2 \to 0,$$

or $\mathbf{\bar{d}^{(k)}} \to \mathbf{\bar{d}}$

Since the vector of load migration is

$$\mathbf{x^{(k)}} = \mathbf{W A^T q_{k-1}(L) y^{(1)}},$$

the accumulated load migration converges:

$$\begin{aligned} \sum_{i=1}^{k} \mathbf{x^{(i)}} &= \mathbf{W A^T}\left(\sum_{i=1}^{k} \mathbf{q_{k-1}(L) y^{(1)}}\right) \\ &= \mathbf{W A^T d^{(k)}} \\ &= \mathbf{W A^T \bar{d}^{(k)}} \\ &\to \mathbf{W A^T \bar{d}} \end{aligned}$$

Let $\mathbf{x} = \lim_{k\to\infty} \mathbf{x^{(k)}} = \mathbf{W A^T \bar{d}}$ be the accumulated load, by (15),

$$\mathbf{Ax} = \mathbf{AWA^T\bar{d}} = \mathbf{L\bar{d}} == \mathbf{y^{(\infty)}} - \mathbf{y^{1)}} = \mathbf{b} \tag{16}$$

Equation (16) is the necessary and suffcient condition [5] for $\mathbf{x}$ to be the minimum of the linearly constrained quadratic optimization problem (14)

□

Theorem 3 substantiates the conjecture at the beginning of this paper, that because the Euclidean norms of the "flow" of the diffusion algorithm were quite close to those of the new load balancing algorithm in numerical experiment [9], it may also satisfy some optimal property.

Because the miminum of the optimization problem (14) is unique under assumption (2), from the above theorem we can also conclude that the flow calculated but any diffusion algorithm of the form (13) is equivalent!

Discussions

In this paper it was proved that the "flow" calculated using any diffusion-like algorithms are equivalent, they all minimizing the Euclidean norm of the "flow".

Even though diffusion algorithms have this optimal property, they are not always efficient. The diffusion algorithm, in its orginal form, was known to suffer from poor convergence on graphs that is not rich in connectivity [1, 3, 9]. Effort has be made to improve the orginal diffusion algorithm while retaining its advantage of nearest neighbor communication [7, 10]. The result of this paper reassures that, any diffusion-like algorithm which are based on polynimials of the Laplacian will possess the same optimal property.

REFERENCES

[1] J. E. Boillat, Load balancing and Poisson equation in a graph. *Concurrency: Practice and Experience* **2** (1990) 289-313.

[2] N. Briggs, *Algebraic Graph Theory*. Cambridge University Press, Cambridge, 1974.

[3] G. Cybenko, Dynamic load balancing for distributed memory multi-processors. *J. Parallel Distrib. Comput.* **7** (1989) 279-301.

[4] R. Diekmann, D. Meyer, B. Monien, Parallel decomposition of unstructured FEM-meshes *Concurrency: Practice and Experience* **10** (1998) 53-72.

[5] R. Fletcher, *Practical Methods of Optimization* (John Wiley and Sons, Chichester, 1987).

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, 1981).

[7] A. Heirich and S. Tayler, A parabolic load balancing method, International Conference on Parallel Processing, 1995.

[8] G. Horton, A multi-level diffusion method for dynamic load balancing, *Parallel Computing* **9** (1993) 209-218.

[9] Y. F. Hu and R. J. Blake, An optimal migration algorithm for dynamic load balancing, Preprint DL-P-95-011, Daresbury Laboratory, Warrington WA4 4AD, UK, 1995, to appear in *Concurrency: Practice and Experience*.

[10] Y. F. Hu and R. J. Blake, Improving the diffusion algorithms for dynamic load balancing, submitted to *Parallel Computing*, 1998.

[11] A. Pothen, D. H. Simon and K. P. Liou, Partitioning sparse matrices with eigenvectors of graphs, *SIAM J. Matrix Anal. Appl.* **11** (1990) 430-452.

[12] K. Schloegel, G. Karypis and V. Kumar, Parallel multilevel diffusion schemes for repartitioning of adaptive meshes, http://www-users.cs.umn.edu/ karypis/metis/parmetis/main.html.

[13] J. Song, A partially asynchronous and iterative algorithm for distributed load balancing, *Parallel Computing* **20** (1994) 853-868.

[14] C. Walshaw, M. Cross and M. Everett, Dynamic load balancing for parallel adaptive unstructured meshes, in: M. Head eds, *Parallel Processing for Scientifi c Computing* (SIAM, 1997).

[15] C. Z. Xu, and F. C. M. Lau, Analysis of the generalized dimension exchange method for dynamic load balancing, *J. Parallel Distrib. Comput.* **16** (1992) 385-393.

[16] C. Z. Xu, F. C. M. Lau, B. Monien and R. Lüling, Nearest-neighbor algorithms for load balancing in parallel computers, *Concurrency: Practice and Experience* **7** (1995) 707-736.