A New Metric for Dynamic Load Balancing

M. Berzins

School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK

Abstract

The issue of the dynamic load balancing of unstructured adaptive meshes is discussed. Experimental results are used to suggest the need for a new metric for dynamic load balancing. The metric is derived by extending the constrained minimisation approach of Hu and Blake [3] to an unconstrained minimisation problem. Simple examples are used to demonstrate the effectiveness of this approach for both the determination of the amount of communications traffic between processors and for the more detailed graph partitioning of a mesh.

1 Introduction

This work is concerned with the dynamic load-balancing problem which arises in the adaptive solution of time-dependent partial differential equations (PDEs) using parallel adaptive algorithms based upon hierarchical mesh refinement such as that described by [8]. Calculations using such codes start by distributing the initial mesh (or generating it in parallel) and then taking as many time steps as is necessary until the mesh needs to be refined or coarsened. At this point the new mesh needs to be redistributed across the processors together with solution values and other data associated with the mesh. In the cases when irregular meshes based on triangles or tetrahedra are used or when regular meshes are used with patches of mesh refinement it is non-trivial to decide how to redistribute the mesh so that each processor has the same load.

A key step in load balancing such calculations has been to make use of graph based techniques following on from the work of Simon [9]. The computations associated with the unstructured mesh are represented by the graph (or possibly the dual graph) of the mesh. The nodes of the graph correspond to the amount of work on each mesh cell (or possibly cluster of cells) and the interconnections between the nodes represent the communications required between

[1] Corresponding author. E-mail: martin @scs.leeds.ac.uk.

them. Graph partitioning techniques are then used to distribute the mesh so that the communications between the processors are minimised and that the load is equally distributed.

As a simple example consider the following simple 8 node mesh taken from Hu and Blake [3] and representing a small-scale but realistic irregular triangular mesh of around 4720 elements around an airfoil. The graph is given by Figure 1.

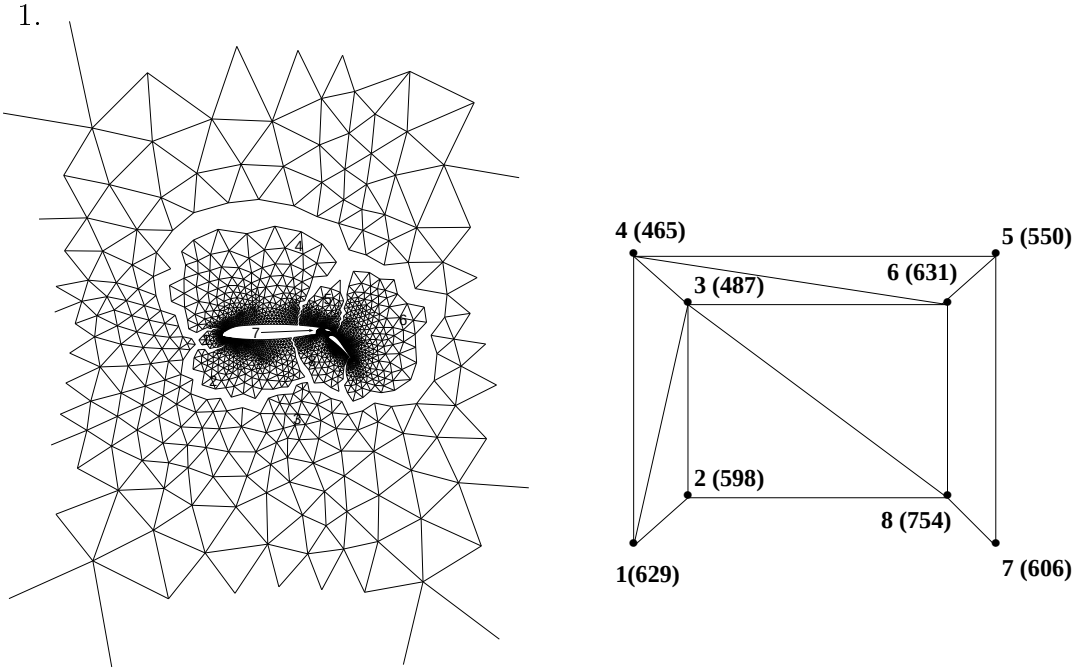

Fig. 1. Mesh and Initial Distribution across 8 Processors

In this case the weighting of the graph represents the number of elements in each partition of the spatial mesh.

In some adaptive mesh codes such as those of [8,11], a fixed coarse mesh is used and then subsequently refined and coarsened. The graph of the coarse mesh may then be used as the basis for partitioning providing that the elements of this graph are weighted by the number of refined elements on each coarse mesh element. The coarse mesh is then distributed so that each processor has roughly the same number of fine mesh elements. In addition the total cut-weight of the partition should be minimised so as to try to reduce the amount of communication.

Of the many existing load-balancing tools, two of the most popular are Metis ([5,6]) and Jostle ([13,14]); both use multilevel partitioning algorithms which produce a hierarchy of coarsenings of the original weighted graph followed by a careful repartition of the coarsest graph. This new partition is then projected back onto the graph at the previous level and modified using a local algorithm in order to improve the partition quality. The process is then repeated until the original weighted graph is recovered together with a modified partition.

P	2	4	8	16	32
Solver Time	2873	1475	833	430	211
Redistribution Time	37	19	42	63	78
Migration Frequency	1	1	2	3	7

Table 1
Timing Results on 32 Processor SGI O2000

Recent experiments by Touheed et al. [12] comparing Jostle and Metis with a number of other methods show that the most robust algorithm in terms of always delivering reasonably well-balanced final partitions is `ParMetis_RepartG` , which is based upon diffusion but still makes use of global information to ensure that a good partition is obtained. This is not usually the best algorithm in any given situation however. Although overall no one algorithm was better than the rest, Jostle and Metis performed better than the other algorithms considered.

Selwood and Berzins [8] investigated the repartitioning of unstructured meshes and provided experimental results based on extensive testing and also quoted results obtained by Touheed [12] using Jostle on a 32 processor Origin 2000. These results are shown in Table 1 in which all times are in seconds. From this table we see that the redistribution time and migration frequency both grow with the number of processors if migration is invoked after a fixed percentage imbalance, (10%) in this case.

Selwood and Berzins' [8] analysis of their application shows that the cost of repartitioning a fixed size mesh is proportional to $log(\frac{P}{2})$ where P is the number of processors. The amount of data moved is also proportional to $log(P)$. Both Selwood and Berzins [8] and Oliker and Biswas [7] suggest that there is a reasonably good correspondence between the maximum number of elements that any processor has to move and the time taken for data redistribution. Touheed et al. [12] make the observation that the number of and quality of repartitions required appear to be the most important factors in the cost and efficiency of parallel dynamic mesh redistribution. A key issue identified in their work (but not addressed by them) is that of determining when repartitioning should take place. An essential ingredient required to make such decisions would appear to be the use of a metric which includes both migration costs and communication/halo costs for a given partition relative to a previous partition.

This issue is also addressed by Simon et al. [10,11], Aravinthan et al. [1] and Oliker and Biswas [7] who express the computational *gain* due to repartition-

ing as

$$gain = t_{iter}\ N_{adapt}\ (W_{max}^{old} - W_{max}^{new})\ +\ t_{refine}\left(\frac{W_{max}^{new}}{W_{max}^{old}} - 1\right)\ -\ t_{datamove} \quad (1)$$

where t_{iter} is the calculation time per element between the mesh being redistributed, N_{adapt} is the number of iterations or timesteps between spatial remeshes. In some situations N_{adapt} is known in advance. In other situations e.g. steady-state calculations every remesh may be the last one and so it may be desirable to get the best possible load balance. W_{max}^{old} and W_{max}^{new} are the maximum loads per processor at the previous remesh and the present remesh and t_{refine} is the cost of refinement. The maximum data movement cost $t_{datamove}$ is given by $t_{datamove} \approx \gamma\ MaxSRE + O$ where γ and O are constants depending on the architecture and $MaxSRE$ is the maximum number of sent/received elements per processor.

As this expression is difficult to use directly for load redistribution, an alternative approach is to try instead to minimise the data movement cost of remeshing and the load imbalance associated with the result of the redistribution. (The actual refinement cost on each processor is small in some applications [8,12] and so t_{refine} is ignored here). Let the extra time due to the maximum processor imbalance be defined by t_{imbal} where

$$t_{imbal}\ =\ t_{iter}\ N_{adapt}\ (W_{max}^{new} - W_{avg}^{new}) \quad (2)$$

where $W_{max}^{new} - W_{avg}^{new}$ is the maximum imbalance in terms of numbers of elements per processor. Hence, from the discussion above, it is necessary to minimise $\{t_{imbal}\ +\ t_{datamove}\}$ the maximum values of the imbalance and the data movement cost. In this work this is only achieved indirectly by addressing the problem in the L_2 norm, rather than the ideal situation in which the maximum L_∞ norm is used. This would however make the load balancing problem much less tractable. In the L_2 norm the expression

$$\{\ \|\ T_{imbal}\ \|_2^2\ +\ \|\ T_{datamove}\ \|_2^2\ \}, \quad (3)$$

is minimised where the vector T_{imbal} is composed of the imbalances across all the processors and the vector $T_{datamove}$ is composed of all the data movement costs between processors. A model of this minimisation problem will be constructed by extending the approach of Hu and Blake [3] to cover the data movement costs.

2 Hu and Blake's Algorithm

Let (V, E) be the connected graph associated with P processors, where $V = 1, 2, ..., P$ is the set of vertices (processors) and E is the set of edges connecting the processors. Each processor i has a load l_i and the average load per processor is

$$l_{avg} = \frac{\sum_{i=1}^{p} l_i}{p}. \tag{4}$$

In the case of heterogeneous processors when the ith processor is a factor of s_i faster than the slowest processor then the average load relative to the slowest processor is

$$l_{avg} = \frac{\sum_{i=1}^{p} l_i/s_i}{p}. \tag{5}$$

Each edge also has associated with it a scalar x_{ij} which is the directional amount of load to be sent from processor i to j. A load balancing algorithm should ensure equal loads i.e.

$$\sum_{j|(i,j)\epsilon E} x_{ij} = l_i - l_{avg}, \quad i = 1, ..., p \tag{6}$$

where the summation is over the, j, edges connected to node i. Again, in the case of heterogeneous processors, the formula must be modified to read

$$\sum_{j|(i,j)\epsilon E} x_{ij} = l_i - l_{avg} s_i, \quad i = 1, ..., p. \tag{7}$$

Should $p - 1$ of these equations be satisfied then the remaining equation will also be satisfied. As there are far more edges in a graph than vertices this equation is likely to have infinitely many solutions and so Hu and Blake [4] choose the solution to minimise the data movement. This is given by:

$$\text{minimise } \tfrac{1}{2}\mathrm{x}^{\mathrm{T}}\ \mathrm{W}\ \mathrm{x} \text{ subject to } \mathrm{A}\ \mathrm{x} = \mathrm{b} \tag{8}$$

where x is the vector with components x_{ij} describing the data moved between nodes. W is the diagonal weighting matrix representing the time taken to communicate a single value between vertices relative to the fastest communications link. The ith component of the vector b is given by the right side of

equation (6). The norm of the vector of data movement cost is thus modelled by

$$\| T_{datamove} \|_2^2 \;=\; \tfrac{1}{2} x^T \; W \; x \tag{9}$$

and the vector of processor imbalances is given by

$$T_{imbal} \;=\; A \; x \;-\; b, \tag{10}$$

where the matrix A is the vertex-edge incident matrix [3] defined by
$[A]_{ij} \;=\; 1$ if vertex i is the head of edge j,
$[A]_{ij} \;=\; -1$ if vertex i is the tail of edge j and the matrix is zero otherwise.
Hu and Blake show that the solution to this problem is found by solving the problem

$$L \; d \;=\; b, \text{ where } \quad \mathrm{L = AW^{-1}A^{T}} \tag{11}$$

where the diagonal inverse of W, denoted by W^{-1}, represents the relative speed of communication between nodes. The values of the vector x are then given by

$$x = W^{-1} \; A^T \; d \; . \tag{12}$$

In the general case where the communication weights of edges c_{ij} vary then the matrix L is the weighted Laplacian of the form:

$$[L]_{ij} \;=\; -\, c_{ij}, \text{ where } \quad \mathrm{i \neq j}, \qquad \mathrm{[L]_{ii} \;=\; \sum c_{ik}, \text{ where } \quad i \leftrightarrow k}, \tag{13}$$

and c_{ij} is the entry of the weighting matrix W^{-1} between vertices i and j. In the case of the graph in Figure 1 the L matrix is given by

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 & & & & \\ -1 & 3 & -1 & & & & & -1 \\ -1 & -1 & 5 & -1 & & -1 & & -1 \\ -1 & & -1 & 4 & -1 & -1 & & \\ & & & -1 & 3 & -1 & -1 & \\ & & -1 & -1 & -1 & 4 & & -1 \\ & & & & -1 & & 2 & -1 \\ & -1 & -1 & & & -1 & -1 & 4 \end{bmatrix}$$

Hu and Blake [3,4] show that the amount of load to be transferred from processor i to processor j is given by

$$x_{ij} \;=\; c_{ij} \; (d_i - d_j), \tag{14}$$

where d_i and d_j are the Lagrange multipliers defined by equation (11) associated with processors i and j.

3 Extending Hu and Blake's Approach

Hu and Blake's [3] formulation does not take into account the fact that the transfer costs must be weighed against the cost of keeping the imbalance until the next load balancing. This can be achieved by using a weighted minimisation of the L_2 norms of the data transfer and the imbalance. This approach thus reflects the observation that at any particular time it may be sub-optimal to exactly load balance the mesh because of the communications cost incurred in doing so. From equations (3), (9) and (10), the minimisation problem thus becomes:

$$\text{Minimise } \{ \mu_1 \tfrac{1}{2} \mathrm{x}^{\mathrm{T}} \mathrm{W} \mathrm{x} + \mu_2 (\mathrm{A} \mathrm{x} - \mathrm{b})^{\mathrm{T}} (\mathrm{A} \mathrm{x} - \mathrm{b}) \} \tag{15}$$

where the parameters μ_1, μ_2 reflect the importance of the cost of movement to the effect of load imbalance and are machine, load and problem dependent. Calculating the derivative of this equation and setting it to zero gives:

$$\mu_1 W \; x + 2\mu_2 \; A^T (Ax - b) = 0$$

Assuming that $\mu_1 \neq 0$ then allows x to be written as

$$x \; = \; W^{-1} \; A^T \; (2(b - Ax)\frac{\mu_2}{\mu_1})$$

This suggests that x can be written as $x = A^T d$ with $d = 2(b - Ax)\frac{\mu_2}{\mu_1}$. Hence as in Hu and Blake's Algorithm, [3], define the vector d by equation (12) and substitute this expression for x into equation (15) to rewrite the minimisation problem as.

$$\text{Minimise } \mu_1 \tfrac{1}{2} \mathrm{d}^{\mathrm{T}} \mathrm{L} \mathrm{d} + \mu_2 (\mathrm{L} \mathrm{d} - \mathrm{b})^{\mathrm{T}} (\mathrm{L} \mathrm{d} - \mathrm{b}) \; . \tag{16}$$

The standard approach for the minimisation of quadratic forms and the use of the identity $L^T = L$ for the Laplacian matrix gives the system of equations, e.g. see [2],

$$L^T \left[\mu_1 \tfrac{1}{2} I + \mu_2 \; L \right] \; d \; = \; \mu_2 L^T \; b \; .$$

Although $L^T e = 0$ for the vector e having all entries with equal value one,[9], this solution is not required as the vector b already contains a multiple l_{avg} of e. Hence the matrix L^T is cancelled to arrive at the system of equations:

$$\left[\mu_1 \tfrac{1}{2} I + \mu_2 \; L\right] \; d \; = \; \mu_2 b \; , \tag{17}$$

Define the ratio $\mu_r = \frac{\mu_1}{2\mu_2}$ as the ratio of mesh movement costs to computation costs to get the system of equations

$$[\mu_r I + \; L] \; d \; = \; b \; , \tag{18}$$

in which only the diagonal entries of the matrix differ from those in equation (11). Hence the algorithm may be implemented in exactly the same way as that of Hu and Blake, [3].

Remarks It is also worth noting that an equivalent matrix problem for the vector x may be given by substituting for d using equation (12) to get

$$(\mu_r \; W + \; A^T A \;)x \; = \; A^T b$$

Thus provinding that μ_r is nonzero the matrix is symmetric positive definite and hence a unique x is guaranteed. It is also worth noting that the final load imbalance is $f = b - Ax = \mu_r d$ and that the traffic is $x^T W x = d^T L d = d^T(b - \mu_r d) = d^T(b - f)$ which is the product of the potential d and the load change $b - f$.

Solving this system of equations for different values of μ_r and with b defined by $b = [39, 8, -103, -125, -40, 41, 16, 164]$ gives the results shown in Table 2. Table 2 shows that as μ_r increases the maximum processor imbalance (Max Imbal) increases and the amount of network traffic decreases. For the largest values of μ_r the cost of movement is so high that the initial distribution is unchanged. For the smallest values of μ_r the movement cost is sufficiently low that the Hu and Blake solution [3] is recovered.

3.1 Uneven processor communication costs Example 2

Now consider the case when processor costs are uneven. Assume that processor pairs (1,2), (3,4) (5,6) and (7,8) can communicate with each other within a pair at cost 1. Further assume that cost of going from one pair to another is 2 except that the cost of moving from processor pairs (1,2) to (7,8) and (3,4) to (5,6) is 4, see Figure 2. In this case the diagonal of the matrix W is given by $W_{diag} = [1, 2, 2, 2, 4, 1, 4, 2, 4, 4, 1, 2, 2, 1]$ where the entries of W_{diag} correspond

μ_r	0.01	0.1	0.5	1.0	2.0	5.0	10	100	1000
Node 1	590	590	590	591	594	601	609	626	629
Node 2	591	592	595	598	599	602	601	598	598
Node 3	589	588	582	575	564	544	526	493	487
Node 4	589	586	572	560	542	515	496	469	465
Node 5	589	588	582	576	569	560	555	550	550
Node 6	591	591	592	594	598	607	614	629	631
Node 7	591	592	598	603	606	610	610	606	606
Node 8	591	595	610	625	647	682	707	748	754
Max Imbal	1	5	20	35	57	92	117	158	164
Traffic	418	406	363	323	264	173	108	10	0
Max Traf	62	60	54	49	40	27	18	2	0

Table 2
Load balance values with variable μ_r for Example 1

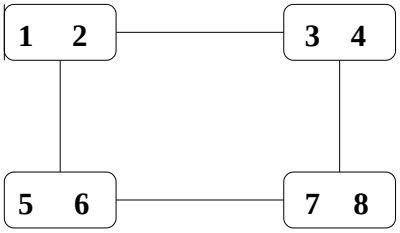

Fig. 2. Processor Configuration

to increasing the starting processor number i after listing all the previously unlisted edges to processors in increasing number j where $j > i$ for i, j as defined in Figure 1. This model architecture has some similarities with that of the Origin 2000. The L matrix for Example 2 is given by

$$L = 1/4 \begin{bmatrix} 8 & -4 & -2 & -2 & & & & \\ -4 & 7 & -2 & & & & & -1 \\ -2 & -2 & 11 & -4 & & -1 & & -2 \\ -2 & & -4 & 8 & -1 & -1 & & \\ & & & -1 & 7 & -4 & -2 & \\ & & -1 & -1 & -4 & 8 & & -2 \\ & & & & -2 & & 6 & -4 \\ & -1 & -2 & & & -2 & -4 & 9 \end{bmatrix}.$$

Table 3 shows that as μ_r increases the maximum processor imbalance (Max Imbal) increases and the amount of network traffic decreases. The table also shows that if we compare the amount of data moved from processor 2 to 8, 3 to 6, 4 to 5 and 4 to 6 in both the weighted case (when the cost is 4 times

Edge /μ_r	0.01	0.1	0.5	1.0	2.0	5.0	10	100	1000
(2,8) original	34	33	30	27	23	15	10	1	0
(2,8) weighted	26	17	26	15	9	5	3	0	0
(3,6) original	19	19	18	18	16	12	8	1	0
(3,6) weighted	20	16	16	11	8	4	2	0	0
(4,5) original	21	20	18	16	13	9	5	0	0
(4,5) weighted	22	16	20	11	7	3	1	0	0
(4,6) original	42	41	37	33	27	18	11	1	0
(4,6) weighted	29	24	23	16	11	6	3	0	0
Traffic	416	384	393	263	192	107	60	3	0
Weighted Traf	938	790	886	591	414	229	131	10	0
Orig WT Traf	978	972	875	781	646	431	277	35	0
Max Imbal	6	63	76	65	85	116	135	161	164
Max Traffic	76	60	71	47	33	18	10	5	0

Table 3
Load balance values with variable μ_r and weights

the unweighted case) and the unweighted case, then less use is made of the expensive links in the weighted case, Example 2. Furthermore Table 3 also shows that the weighted traffic (Weighted Traf) summed over all the edges is less than would have been the case had the original unweighted graph of the previous section been used in the weighted case (Orig WT Traf). The maximum amount of traffic (Max Traffic) also decreases as μ_r increases, though there is an unexplainable blip at $\mu_r = 0.5$.

3.2 Uneven processor communication costs Example 3

The following simple example is instructive in showing how the proposed approach deals with variable communications costs and a variable ratio of μ_r. Consider the simple network with 4 nodes and six edges given in Figure 3. Assume that the outside edges e1, e2 and e4 have a communications cost $\frac{1}{tc}$ and the interior edges e3, e5 and e6 have a communications cost of one. The initial values of N1,N2,N3 and N4 are 30,10,5 and 15 respectively. The matrix $[\mu_r I + L]$ is given by

$$[\mu_r I + L] = 1/4 \begin{bmatrix} 2tc+1+\mu_r & -tc & -tc & -1 \\ -tc & 2tc+1+\mu_r & -tc & -1 \\ -tc & -tc & 2tc+1+\mu_r & -1 \\ -1 & -1 & -1 & 3+\mu_r \end{bmatrix}$$

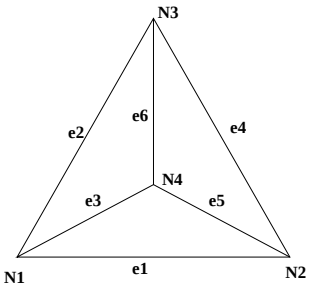

Fig. 3. Simple 4 Node Configuration

tc	0.01			1			10		
μ_r	0.01	1.0	100	0.01	1.0	100	0.01	1.0	100
Load node N1	16	23	30	16	18	30	16	16	27
Load node N2	14	12	10	14	14	10	14	14	11
Load node N3	14	9	5	14	13	5	14	14	7
Load node N4	15	15	15	14	15	15	15	15	15
Max Imbal	1	8	15	1	3	15	1	1	12
Comms edge e1				4	4		6	6	1
Comms edge e2				6	5		8	7	1
Comms edge e3	14	7		3	3				
Comms edge e4				1	1		1	1	
Comms edge e5	4	2		1	1				
Comms edge e6	9	4		2	2				
Total Traffic	27	13		17	16		15	14	2

Table 4
Load balance values with variable μ_r and weights

Table 4 shows the loads on the nodes N1,N2,N3 and N4 and the communications taking place along edges e1 to e6. It is straightforward to see that when tc is small communications should take place along interior edges e3, e5 and e6. Furthermore when tc is large communications should take place along exterior edges e1, e2 and e4. Table 4 shows that this is indeed what happens. As expected, as μ_r is increased the amount of communication is reduced. This example thus provides a good illustration of how the proposed approach takes the cost of moving data into account.

4 An Alternative Application of the New Metric

Although the previous section considered how to define the amount of data movement between processors, the same idea may be used with a similar metric to estimate how to partition a computational mesh between a pair of processors as part of a recursive bisection procedure. The central idea is to extend the original graph-based partitioning approach suggested by Simon [9] to include the costs of data movement. For example, consider the partitioning problem after the $(n+1)$th remesh with just two processors. For simplicity assume that the number of elements before and after partitioning is the same. Let L^{n+1} be the weighted Laplacian matrix of the weighted dual graph of the coarsest level mesh after the $(n+1)$th remesh (see, for example, [2]) and let x^n be the latest partition vector ($x_i^n = \pm 1$ according to which subdomain coarse element i belongs to). The communication/halo overhead at the next step of the solver is therefore proportional to $(x^n)^T L^{n+1} x^n$. Conversely, if repartitioning were to take place (leading to a new partition vector x^{n+1}) before the next step of the solver, the new communication/halo overhead plus the movement cost would be proportional to

$$(x^{n+1})^T \; L^{n+1} \; x^{n+1} + \lambda (x^{n+1} - x^n)^T (x^{n+1} - x^n) \tag{19}$$

for some constant λ (the ratio of moving cost to communications costs). With a suitable choice of this constant, this quadratic form could be minimised by solving the equations:

$$(L^{n+1} + \lambda I) \; x^{n+1} = \; \lambda \; x^n \tag{20}$$

for a new partition vector x^{n+1}, thus yielding an alternative dynamic load-balancing heuristic. In contrast to the approach of the previous section, which provides a way of deciding how much data should be moved between processors, this approach provides an explicit mechanism for deciding which elements should be moved.

One issue for this metric is that the matrix L is singular and hence the degenerate case of $\lambda = 0$ must be treated separately as in the work of Simon [9] and many others since. Although the vector e with every entry having value one satisfies $Le = 0$ and can be subtracted from the solution obtained by solving equations (20) it is not necessary to do this in order to calculate a partition, see [2]. Furthermore unlike the approach described earlier there is no direct attempt to take into account the imbalance of the existing partition. The value of this imbalance is in general given by $e^T x^{n+1}$. This does open up the possibility of including the square of this expression in equation (19). As with the earlier approach we need to determine appropriate values of the

balancing parameter λ.

Another even more taxing issue is how to solve the equations defined by equation (20) is a sufficiently fast way so as to not add excessive overhead to the load-balancing calculation. The issue is more pressing in this case as the overall number of equations to be solved is proportional to the number of mesh elements rather than the number of processors.

4.1 Rectangular Refined Mesh Example 4

In order to illustrate this approach consider the following simple example consisting of a rectangular coarse mesh with horizontal and vertical links only. This mesh may be represented by a matrix in which the value 1 represents one coarse mesh cell, 4 represents one coarse mesh cell divided into 4, and 16 represents one coarse mesh cell divided into 16 equal cells. One domain is defined by positive signs while negative signs are assigned to the other domain. Define the communications cost between the domains as the sum of the absolute values on both sides of the interface. Suppose the original mesh is given by

$$\begin{matrix} 4 & 4 & 1 & -1 \\ 4 & 1 & -1 & -4 \\ 4 & 1 & -1 & -4 \\ 1 & -1 & -4 & -4 \end{matrix} \quad \text{which is remeshed to} \quad \begin{matrix} 1 & 4 & 16 & -16 \\ 1 & 1 & -1 & -4 \\ 4 & 1 & -1 & -1 \\ 16 & -16 & -4 & -1 \end{matrix}$$

The communications cost for the original mesh is 8 while for the new mesh with the original partition it is 68. Applying the algorithm described above for three different values of λ gives three new partitions

$$\begin{matrix} -1 & -4 & -16 & -16 \\ 1 & -1 & -1 & -4 \\ 4 & 1 & 1 & -1 \\ 16 & 16 & 4 & 1 \end{matrix} \qquad \begin{matrix} 1 & -4 & -16 & -16 \\ 1 & -1 & -1 & -4 \\ 4 & 1 & -1 & -1 \\ 16 & 16 & 4 & 1 \end{matrix} \qquad \begin{matrix} 1 & -4 & -16 & -16 \\ 1 & 1 & -1 & -4 \\ 4 & 1 & 1 & -1 \\ 16 & 16 & -4 & 1 \end{matrix}$$

(a) $\lambda \leq 0.1$ (b) $\lambda = 1$ (c) $\lambda = 10$

The calculation of the partition is performed as in Hodgson and Jimack [2]. In order to assess the effectiveness of this approach it is necessary to define the moving costs associated with this approach. Let c_i^{old} be the weight of cell i before remeshing and let c_i^{new} be the weight of cell i after remeshing. Assuming that a coarse mesh cell moving from one partition to another is represented by a change of sign of x_i then the moving cost used is given by

$$Cost_{moving} = \sum_i \; min\; (c_i^{old}, c_i^{new})\; \frac{1}{4}\; (x_i^{old} - x_i^{new})^2 \tag{21}$$

where the implicit assumption is that mesh cells are coarsened in their existing partition and refined in their new partition, see [7]. In this case the moving cost for $\lambda \leq 0.1$ is 14, while for $\lambda = 1$ the moving cost = 12 and for $\lambda = 10$ the moving cost is 8. The imbalances in each of these cases are zero except for Case (c) in which there is an imbalance of 2. The communications cost for each of the new patterns are 8, 15 and 30 in cases (a), (b) and (c) respectively. Thus as the cost of moving data rises less data is moved, but at the penalty of performing more communications at the solver stage.

5 Conclusions

This work has shown by means of simple and easily comprehended examples that the idea of taking into account data redistribution costs has some merit with regard to the load balancing of dynamically varying unstructured meshes. Although the simple model presented here is a promising start much more work remains to be done. In particular the model needs to be parameterised against a range of realistic examples.

Acknowledgements

The author would like to thank Yifan Hu for supplying the remarks given directly below equations (15) and (18) and for supplying the mesh diagram in Figure 1. The author would like to thank Peter Jimack and Chris Walshaw for commenting on drafts of this paper, Bruce Hendrickson for his valuable comments on this topic and Guy Lonsdale, Joe Flaherty and George Karypis for invitations to meetings that indirectly led to this work being done.

References

[1] V.Aravinthan, S.P. Johnson, K. McManus, C. Walshaw and M. Cross, Dynamic Load Balancing for Multi-Physical Modelling using Unstructured Meshes, Proc. 11th Intl. Conf. Domain Decomposition Methods, Greenwich, UK, 1998, eds. C.H. Lai P.E. Bjørstad, M. Cross, and O. Widlund, DDM.org, http://www.ddm.org, www.ddm.org/DD11/Aravinthan.ps.gz", pp. 380-387, 1999.

[2] D.C. Hodgson and P.K. Jimack, *"Efficient Mesh Partitioning for Parallel Elliptic Differential Equation Solvers."*, Computing Systems in Engineering, 6, 1–12, 1995.

[3] Y.F. Hu and R.J. Blake *"An Optimal Migration Algorithm for Dynamic Load Balancing "*, Concurrency: Practice and Experience, Vol. 10 (6) 467-483 (1998)

[4] Y.F. Hu and R.J. Blake *"An Improved Diffusion Algorithm for Dynamic Load Balancing "*, Parallel Computing, 25, 417–444 (1999)

[5] G. Karypis and V. Kumar, *"A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm"*, Proc. of 8th SIAM Conf. on Parallel Proc. for Scientific Computing, SIAM, 1997.

[6] G. Karypis, K. Schloegel and V. Kumar, *"ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 2.0"*, Department of Computer Science, University of Minnesota, 1998.

[7] L. Oliker and R. Biswas, *"PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes"*, J. Parallel and Distributed Computing, 52, 150–177, 1998.

[8] P.M. Selwood and M. Berzins, *"Portable Parallel Adaptation of Unstructured Tetrahedral Meshes"*, Concurrency Vol. 11 (13) 1-22, 1999.

[9] H.D. Simon, *"Partitioning of Unstructured Problems for Parallel Processing"*, Computing Systems in Engineering, 2, 135–148, 1991.

[10] H.D. Simon, A.Sohn and R. Biswas. Harp A Dynamic Spectral Partitioner. Journal of Parallel and Distributed Parallel Computing, 50, 83-103 1998.

[11] A.Sohn, R.Biswas and H.D. Simon. Impact of Load Balancing on Unstructured Adaptive Mesh calculations for Distributed-Memory Multiprocessors. In Proc of Eighth IEEE Symposium on Parallel and Distributed Computing. pp 26-33 New Orleans, Louisiana, 1996.

[12] N.Touheed , P. Selwood, P.K. Jimack, and M. Berzins A Comparison of Some Dynamic Load Balancing Algorithms for a Parallel Adaptive Flow Solver. Parallel Computing 26(12) (2000) pp. 1535-1554.

[13] C. Walshaw, M. Cross and M.G. Everett, *"Dynamic Load-Balancing for Parallel Adaptive Unstructured Meshes"*, Proc. of 8th SIAM Conf. on Parallel Proc. for Sci. Comp., SIAM, 1997.

[14] C. Walshaw, M. Cross and M.G. Everett, *"Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes"*, J. Par. Dist. Comput., 47, 102-108, 1997.