Partitioning and Scheduling Algorithms and Their Implementation in FELISA – An Unstructured Grid Euler Solver

Y. F. Hu[a], R. J. Blake[a], J. Peiro[b], J. Peraire[b] and K. Morgan[c]

[a] Daresbury Laboratory, S.E.R.C., Warrington WA4 4AD, United Kingdom

[b] Dept. of Aeronautics, Imperial College of Science, Technology and Medicine, London SW7 2BY, United Kingdom

[c] Dept. of Civil Engineering, University College of Swansea, Swansea SA2 8PP, United Kindom

1. INTRODUCTION

The numerical solution of partial differential equations with finite element algorithms on distributed memory parallel computers demands that the global mesh be divided into subdomains, the number of which corresponds to the number of processors. The decomposition should be such that the number of elements per subdomain is roughly the same, to ensure global load balancing, and the number of shared faces (edges) between subdomains are minimised to minimise the communication costs. In this paper we shall start with a comparison of a number of established grid partitioning algorithms [1-2]. A new hybrid algorithm [3] is then proposed and found to be highly competitive on small to medium size meshes.

Once a partitioning has been established for an irregular grid, a communication scheme need to be devised to organize the communication between processors [4]. Message passing scheme of blocked pairwise exchange in a number of stages is studied. In this paper we develop algorithms to deal with non-uniform message lengths. The algorithms [5] attempt to minimise the communication time by scheduling messages of similar lengths in the same stage of the message passing scheme. It is found that this load balancing of communication can reduce the communication costs by up to 30%. Some results on other message passing methodology will also be given.

The partitioning and scheduling algorithms are used as a preprocessor in a parallel version of the unstructured grid finite element 3D explicit Euler equation solver FELISA [6]. Results of this parallel solver will be given.

2. PARTITIONING OF MESHES

Since finding the best partitioning is in general an NP hard problem, a number of heuristic partitioning algorithms have been suggested which attempt to find an approximation to the best partitioning. Among these algorithms are: recursive coordinate bisection (RCB), recursive graph bisection (RGB) and recursive spectral bisection (RSB) [1]; MINCUT algorithm [2]; greedy algorithm

[7]; inertial algorithm (see [8]); physical algorithms such as simulated annealing, genetic algorithm and neural net [9-10].

The first four algorithms listed above are tested on 4 meshes with the number of elements ranging from 771 to 5520. The first three meshes are 2D meshes, the last is a 3D mesh. For each mesh, its dual graph is partitioned. Because some of these four algorithms can give erroneous results if the mesh to be worked on is disconnected, therefore in our implementations a preprocessor is included which checks the connectivity of the graph. A disconnected graph will be made connected by linked different components of the graph. The graphs are partitioned into 16 subdomains. We report in Table 1 the edges cut (the number of shared faces (edges) between subdomains) for each algorithm on the 4 problems.

Table 1 Comparison of edges cut of five partitioning algorithms

Meshes	RCB	RGB	RSB	MINCUT	MINGRAPH
788	142	142	108	133	116
771	232	201	150	188	148
3564	293	334	261	500	233
5520	2876	1737	1000	896	822

It is found that the simple recursive coordinate bisection (RCB) algorithm takes little time, but the number of edges cut is relatively high compared with other algorithms. The recursive graph bisection (RGB) algorithm also takes very little time, it is comparable with or even worse than RCB on 2D problems, but much better on the 3-D problem, which confirms an observation of Simon [1]. The MINCUT gives very competitive numbers of edges cut, but suffers from very long running time. The recursive spectral bisection (RSB) algorithm gives very good results of edges cut in a reasonable time. Therefore comparatively RSB is the best algorithm of the four.

The very competitive results on edges cut for the MINCUT make us try to modify the algorithm to reduce the CPU time. The MINCUT algorithm is normally started with randomly generated initial partitions. It attempts to improve the partitioning by swapping elements between subdomains to reduce the edges cut. Randomly generated partitions have very high numbers of edges cut, thus will take the MINCUT algorithm many iteration to bring the number of edges cut down, which results in a long CPU time. It is natural to try to start with a reasonably good initial partition and to improve upon it using MINCUT. On the other hand this initial partition should not take too much time to generate. The graph bisection algorithm is thus chosen to generate an initial bisection of the graph, then the bisection is improved upon using the MINCUT algorithm. The resulting algorithm is called MINGRAPH. The results for the hybrid algorithm are reported in the last column of Table 1. It is found that in 3 out of the 4 cases the MINGRAPH algorithm gives partitioning of better quality than the recursive spectral bisection algorithm. Its running time is also competitive.

These algorithms are also tested on large size meshes of up to 353710 elements. It is found that with meshes of size more than 10000, MINCUT and MINGRAPH takes too long time to converge. RCB and RGB suffer from poor quality of the partitioning. Thus for large meshes, we will used RSB algorithm for partitioning. Recently, a multilevel implementation of the RSB has been suggested [11]. This uses ideas similar to multigrid methods to speed up the

RSB algorithm. We found that the new approach does give a much faster RSB algorithm, making it suitable for the partitioning of large meshes.

3. SCHEDULING OF MESSAGE PASSING

Considering the 2D mesh consisting of 788 elements used in Table 1. Let the mesh be partitioned into 4 subdomains of 197 elements each using the recursive coordinate bisection (RCB) algorithm, and each subdomain is assigned to one processor. It is found that, for example, processor 1 shares 9 edges with processor 2. Assume the amount of communication between processors is proportional to the number of edges they share, then the communication task can be described by Table 2, in which the adjacent processors (*adjproc*) and the message lengths (*msglen*, in brackets) are given. For instance from the table it is seen that processor 4 is to communication with processors 2 and 3, the amount of communication is 14 and 7 (units) respectively.

Table 2 A communication task table

Processor	*adjproc* (*msglen*)		
1	2(9)	3(17)	–
2	1(9)	3(2)	4(14)
3	1(17)	2(2)	4(7)
4	2(14)	3(7)	–

3.1 Scheduling of Blocked Message Passing

The parallel computer we use, Intel i860 hypercube, has two ways of message passing. The *blocked* message passing is given by calling FORTRAN subroutines *csend* and *crecv*. A processor calling *crecv* for example, has to wait for the message to be received before it can starting other part of the computer program. The *unblocked* message passing is given by subroutine *isend* and *irecv*. The processor can start doing other work immediately after calling these subroutines. When it reaches a point where the message to be received is needed, it can call subroutine *msgwait* to wait for the message to arrive. In this subsection the *blocked* message passing will be assumed.

Table 3 A scheduling scheme

Processor	*adjproc* (*msglen*)		
1	2(9)	3(17)	–
2	1(9)	4(14)	3(2)
3	4(7)	1(17)	2(2)
4	3(7)	2(14)	–

Venkatakrishnan, Simon and Barth [4] suggested that communication can be scheduled in stages, each stage processors grouping in pairs will do message exchanges. For example the communication task given by Table 2 can be done in three stages, as shown by Table 3. In the first stage processors (1,2) and (3,4) exchange messages, in the second stage processors (1,3) and (2,4) exchange messages and in the final stage processors (2,3) exchange messages. Each stage

is followed by a synchronization. In the following we will discuss how to form such a scheme that has minimal communication costs.

Relating to each communication task table, a communication task graph is defined as an undirected graph. Each of its vertices corresponds to a processor. Two vertices i and j are linked by an edge with weight *msglen* if and only if processor i and j shares *msglen* edges. The scheduling of communication is found to be equivalent to colouring the edges of the communication task graph so that no edges that start from the same vertex have the same colour [4].

Let *maxadj* denotes the maximum number of adjacent processors to any one processor (*maxadj* is also the largest degree in the communication task graph). Venkatakrishnan, Simon and Barth [4] derived a scheduling algorithm that can organize the communication in no more than $maxadj + 1$ stages, although the details of how this can be done are not given in their paper. They assumed that the message lengths are uniform. Under this assumption any scheduling schemes which have the same number of stages will have the same communication cost. We shall however consider the general case when the message lengths are not uniform.

Table 4 A communication task table

Processor	*adjproc* (*msglen*)				
1	9(2)	12(2)	–	–	–
2	3(3)	10(1)	11(3)	13(6)	–
3	2(3)	10(4)	11(8)	12(1)	–
4	7(3)	14(5)	15(6)	–	–
5	9(2)	10(4)	–	–	–
6	11(4)	12(5)	16(4)	–	–
7	4(3)	10(2)	13(4)	14(6)	–
8	15(6)	16(6)	–	–	–
9	1(2)	5(2)	–	–	–
10	2(1)	3(4)	5(4)	7(2)	13(10)
11	2(3)	3(8)	6(4)	12(2)	–
12	1(2)	3(1)	6(5)	11(2)	16(6)
13	2(6)	7(4)	10(10)	14(3)	–
14	4(5)	7(6)	13(3)	–	–
15	4(6)	8(6)	–	–	–
16	6(4)	8(6)	12(6)	–	–

The colouring of edges of a graph with no colour conflict has been studied in Graph Theory and a theorem by Vizing [12] states that the minimum number of colours needed to colour the edges of a graph (whose maximum degree is *maxadj*) without conflict is either *maxadj* or $maxadj+1$, but to decide whether the graph can be coloured with only *maxadj* colours is an NP-hard problem. A scheduling algorithm VIZING based on the proof, which can produce a scheduling scheme having no more than $maxadj + 1$ stages, is implemented [5]. This algorithm is applied to the communication task shown in Table 4, which is a task given by using RSB to partition the mesh with 788 elements into 16 subdomains. The

scheduling scheme produced by algorithm VIZING is given by Table 5 (left), which has a communication cost of 36 (equals to the sum of the maximum *msglen* in each stage). However since now the message lengths are no assumed to be uniform, the scheduling scheme given by Table 5 (left) is not optimal.

Table 5 A scheduling scheme (left) and a near optimal scheme (right)

Processor	*adjproc* (*msglen*)					*adjproc* (*msglen*)				
1	9(2)	12(2)	–	–	–	12(2)	–	–	–	9(2)
2	3(3)	10(1)	11(3)	13(6)	–	3(3)	11(3)	10(1)	13(6)	–
3	2(3)	11(8)	10(4)	12(1)	–	2(3)	12(1)	–	10(4)	11(8)
4	7(3)	14(5)	15(6)	–	–	–	–	7(3)	14(5)	15(6)
5	10(4)	9(2)	–	–	–	10(4)	–	–	9(2)	–
6	11(4)	16(4)	12(5)	–	–	16(4)	–	–	11(4)	12(5)
7	4(3)	13(4)	14(6)	10(2)	–	13(4)	10(2)	4(3)	–	14(6)
8	15(6)	–	16(6)	–	–	–	–	–	15(6)	16(6)
9	1(2)	5(2)	–	–	–	–	–	–	5(2)	1(2)
10	5(4)	2(1)	3(4)	7(2)	13(10)	5(4)	7(2)	2(1)	3(4)	13(10)
11	6(4)	3(8)	2(3)	–	12(2)	–	2(3)	12(2)	6(4)	3(8)
12	16(6)	1(2)	6(5)	3(1)	11(2)	1(2)	3(1)	11(2)	16(6)	6(5)
13	14(3)	7(4)	–	2(6)	10(10)	7(4)	14(3)	–	2(6)	10(10)
14	13(3)	4(5)	7(6)	–	–	–	13(3)	–	4(5)	7(6)
15	8(6)	–	4(6)	–	–	–	–	–	8(6)	4(6)
16	12(6)	6(4)	8(6)	–	–	6(4)	–	–	12(6)	8(6)
max. *msglen*	6	8	6	6	10	4	3	3	6	10
comm. cost	36					26				

We suggest here a near optimal scheduling algorithm REDUCE to be used after using the algorithm VIZING. The idea of algorithm REDUCE is to group messages of similar lengths in the same stage, so as to attempt to minimise the overall communication cost. The algorithm is described as follows on the communication task graph. a) First find the edge e_1 with the largest message length and assume its colour is c_1. Lock the edge. b) Then find among unlocked edges an edge e_2 with the largest message length, assume the colour of e_2 is c_2. If $c_1 = c_2$ then the two large messages are indeed in the same stage, lock e_2. Otherwise find the largest path containing e_2 and coloured alternately with c_2 and c_1. If such a path contains locked edges, then it is impossible to group e_1 and e_2 in the same stage, so lock e_2; otherwise exchange colours c_2 and c_1 on the path, lock e_2. c) Repeat the process from b) until all edges are locked, then remove edges coloured with c_1 and restart from a). Applying the algorithm REDUCE on the scheduling scheme given by Table 4 results in Table 5 (right), with the communication cost reduced from 36 to 26.

The communication costs of 36 or 26 in Tables 5 are just predictions of the actual costs that the scheduling schemes will give, under the assumption that there are no link contentions. In order to see if such prediction still has any significance in practice, simulations have been run on an Intel i860 hypercube. Two

meshes are considered and the five partitioning algorithms discussed in Section 1 are used to partition each mesh into 16 subdomains. For each partitioning, two message passing schemes are generated using VIZING and VIZING+REDUCE respectively. For each message passing scheme, a simulated communication time is derived as follows. Sixteen nodes of the i860 are each given a copy of the same message passing scheme (such as those of Table 5). Each node determines from the scheme to whom (given by *adjproc*) it needs to communicate, and the length of the message (given by *msglen*). For each shared edge, it is assumed 10 double precision numbers need to be exchanged. The communication time is taken to be the elapsed time between the start and the finish of the message passing. It is found [5] that the predicted costs reflect the trend of actual communication time very well. The nearly optimal scheduling schemes given by VIZING+REDUCE reduce the communication cost of scheduling scheme, given by using VIZING only, by 16% on average, and over 30% in some cases.

3.2 Other Message Passing Strategies

The communication strategy in Section 3.1 will be termed *blocked and scheduled* (BS) message passing strategy. *Blocked but unscheduled* (BU) message passing strategy is also tested, given by the following pseudo code:

Do $i = 1$, *Number of Neighbors*
  csend data to all neighbors
End do
Do $i = 1$, *Number of Neighbors*
  crecv data from all neighbors
End do

so is *unblocked and unscheduled* (UU) strategy, given by the above code but with *csend* and *crecv* replaced by *isend* and *irecv*. These three strategies are compared on a number of communication tasks. A typical example is given in Table 6, where the communication task is given by using the multilevel implementation of the recursive spectral bisection (RSB) algorithm to partition a mesh with 353710 elements into 16 subdomains. The communication time (in milliseconds) to do one message passing using one of the three communication strategies is listed in Table 6.

Table 6 The communication time taken by the three strategies

Scheduling Scheme	BS	BU	UU	
Not scheduled	–	313	216	
VIZING		343	262	211
VIZING + REDUCE	239	243	213	

As can be seen from Table 6, for the two *blocked* communication strategies, our near optimal scheduling algorithm REDUCE improves the communication time by between 7% to 30%. The *unblocked* communication strategy takes the least communication time. However the two *unscheduled* strategies BU and UU put more strain to the computer, and some systems may not support *unblocked* communication. The *blocked and scheduled* communication strategy is thus adopted in our work on FELISA (see next section) for its portability. Of course when code optimization is the main consideration, one should use *unblocked* strategy if the system allows.

4. APPLICATION TO FELISA

The partitioning and scheduling algorithms discussed in the previous two sections form a preprocessor. This preprocessor is used in a parallel version of an unstructured grid Euler solver – FELISA. A typical result of running 10 iteration of the parallel code on a full aircraft configuration (with 353710 elements) is shown in Table 7. As can be seen, the algorithm scales quite well with the increase of the number of processors. On 32 processors, the performance is about equivalent to that of three CRAY YMP nodes.

Table 7 Time taken by FELISA

Cray YMP	Elapse time (seconds)
1 Processor	183
Intel i860 hypercube	Elapse time (seconds)
8 Processors	208
16 Processors	114
32 Processors	67

REFERENCE

[1] H. D. Simon, Partitioning of unstructured problems for parallel processing, Computer Systems in Engineering, 2 (1991) 135-148.

[2] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Systems Tech. J. 49 (2) (1970) 291-308.

[3] Y. F. Hu and R. J. Blake, Numerical experiences with partitioning of unstructured meshes, Daresbury Laboratory preprint DL/SCI/P865T, 1993.

[4] V. Venkatakrishnan, H. D. Simon and T. J. Barth, A MIMD implementation of a parallel Euler solver for unstructured Grids, Report RNR-91-024, NAS Systems Division, Applied Research Branch, NASA Ames Research Center, 1991.

[5] Y.F. Hu and R. Blake, Some algorithms for the scheduling of message passing, Daresbury Laboratory preprint DL/SCI/P871T, 1993.

[6] J. Peiro, J. Peraire and K. Morgan, FELISA System Version 1.0, User Manual.

[7] C. Farhat, A simple and efficient automatic FEM domain decomposer, Computers and Structures 28 (1988) 579-602.

[8] B. Hendrickson and R. Leland, Multidimensional spectral load balancing, Sandia National Laboratories, Albuquerque, NM 87185, 1993.

[9] R. D. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency: Practice and Experience 3 (1991) 457-481.

[10] N. Mansour, Allocating data the multicomputer nodes by physical optimization algorithms for loosely synchronous computations, Concurrency: Practice and Experience 4 (1992) 557-574.

[11] S. T. Barnard, H. D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Report RNR-92-033, NASA Armes Research Center, November 1992.

[12] V. G. Vizing, On an estimate of the chromatic class of a p-graph (Russian), Diskret. Analiz. 3 (1964) 25-30.