Algorithms for Scheduling with Applications to Parallel Computing

Y. F. Hu [a] and R. J. Blake [a]

[a] *Daresbury Laboratory, CCLRC, Warrington WA4 4AD, UK*

Abstract

Scheduling of message passing for synchronous communication is found to be equivalent to colouring the edges of a graph without conflict. The graph edge-colouring problem, which has other applications, is studied. An algorithm which colours the graph with no more than *deg* colours, where *deg* is the degree of the graph, is implemented. The problem of minimising the sum of the largest weight for each colour is also investigated and an algorithm suggested. These algorithms are used to organise the communication as part of a finite element Euler solver. Different communication schemes and their effect on the performance of the flow solver are compared.

Keywords: graph edge-colouring; scheduling; communication time; grid partitioning; message passing.

1 Introduction

The numerical solution of differential equations using finite element methods involves constructing a mesh of many small elements over the physical domain. On a distributed memory parallel computer, the mesh should be further decomposed into subdomains, the number of which usually corresponds to the number of processors. The decomposition should be such that

- a) each subdomain has roughly the same number of elements, so as to achieve load balance;
- b) the number of faces shared between subdomains is as little as possible, so as to minimise the communication.

This task of mesh decomposition is related to the graph partitioning problem, which was proved to be NP-hard. A number of graph partitioning algorithms have been suggested, among them are those based on geometric information (e.g., recursive coordinate bisection, recursive inertial bisection);

algorithms based on connectivity information (e.g., recursive spectral bisection [11], greedy algorithm [3]; local and global optimisation type algorithms (e.g., Kernighan-Lin algorithm [9], simulated annealing [15]. Multilevel techniques [1] have been introduced to speed up the algorithms as well as to provide a global view of the problem. Algorithms combining the multilevel idea and the local optimisation algorithms are found to be most successful [14,8,5]. For very large meshes (graphs) parallel algorithms are necessary [8,2].

Once a partitioning is given, each subdomain is assigned to a processor, and after some computation, processors exchange data on the shared boundaries of the subdomains. This process of computation and exchange of data goes on until the solution has converged. The data exchange is achieved by each processor sending messages to and receiving messages from processors that it shares subdomain boundaries with.

Figure 1 (a) shows a mesh which is partitioned into 4 subdomains, assigned to processors numbered 1, 2, 3, and 4. Figure 1 (b) shows the *communication task graph* of the partitioning. For example, processor 2 has to exchange information with processors 1, 3 and 4. The numbers in brackets in Figure 1 (b) are the number of edges shared between processors, thus, for example, processors 2 and 4 share 2 edges. The communication task of Figure 1 (b) can also be described by a *communication task table* (Table 1), in which the first column lists the identities of the processors, the other entries list the identities of adjacent processors (*adjproc*) and the corresponding message lengths (*msglen*, in brackets) for data exchange. Table 1 shows that processor 3 has to exchange messages with processors 2 and 4, with message lengths 14 and 7 respectively.

On many parallel platforms the communication time for sending a message is roughly linear in the message length, subject to a start-up cost. Therefore for convenience and machine independence, in this paper the message length is used as an indication of the actual communication cost. However one can always replace the message length by the estimated time of sending a message of this length and all the algorithms of the paper still apply.

It is important to minimise the communication time because as more processors are used, the communication overhead usually becomes the bottleneck for good efficiency.

The problem of scheduling of message passing for finite element type applications was addressed by Venkatakrishnan et al. [12]. They assumed that there were no link contentions and that the communication time between the processors were equal. This implies that either the message lengths between processors are uniform or that the start-up time of communication dominates. They suggested that the communication can be done in a number of stages. At each stage, processors are grouped into pairs and do "pair-wise exchange",

without interference from other processors. For example, the communication task given by the graph shown in Figure 1 (b) can be executed in three stages as described by Table 2. In the first stage, processors (1,2) and (3,4) exchange messages. In the second stage, processors (1,4) and (2,3) exchange messages and in the final stage processors (2,4) exchange messages. In between each stage, a synchronisation is carried out.

A scheduling scheme which incorporates the feature that one processor is paired with at most one other processor at any stage, and that any two processors that are linked in the communication graph are paired at some stage, is equivalent to the solution of the problem of colouring all the edges of the communication task graph so that no two edges that start from the same vertex have the same colour.

For example, the scheduling scheme given in Table 2 is equivalent to colouring the edges between 1, 2 and 3, 4 in Figure 1 (b) with colour c_1, the edges between 2, 3 and 1, 4 with colour c_2, and the edge between 2, 4 with colour c_3.

Let deg denotes the maximum number of messages to be sent, or degree of the communication task graph (in Figure 1 (b) $deg = 3$). Venkatakrishnan et al. [12] derived a scheme that can complete the message exchange in no more than $deg + 1$ stages, although they did not give details of their algorithm.

In this paper it is assumed that there are no link contentions, but the message lengths may vary and the start-up cost may not necessarily dominate the communication time. Under these assumptions the total communication time of a scheme is the sum of the longest time in each stage (plus the synchronisation cost). For example the communication cost of the scheduling scheme given in Table 2 is $9 + 17 + 2 = 28$. Two scheduling schemes of the same communication task and with the same number of stages can thus give different communication costs. For example Figure 2 shows two schemes for the same communication task graph. The communication time between processors is given in brackets and three colours c_1, c_2 and c_3, which correspond to three stages of the communication schemes, are used. The first scheme (Figure 2 (a)) requires $6 + 6 + 5 = 17$ units of time, the second scheme (Figure 2 (b)) needs only $6 + 1 + 5 = 12$ units of time. Thus in deriving scheduling schemes, one should also seek to minimise the sum of the largest weight of edges with the same colour.

The scheduling of communication is only one example in parallel computing where the graph edge-colouring problem appears. There are other applications. For instance, the Kernighan-Lin (KL) algorithm [9] is usually used to reduce the number of shared edges (edge-cuts) of an initial crude partition of a graph. In the case of bisection, a gain is worked out for each vertex on the

two subdomains, which represents the reduction in edge-cuts had the vertex been moved to the other subdomain. Every iteration, one vertex will be picked and moved from the subdomain with more vertices to the one with less vertices. The vertex is chosen to be the one with the largest gain. Implementing the algorithm in parallel on many processors presents a challenge, as without modification the KL algorithm is sequential in nature. One way of implementing the KL algorithm in parallel [2] is to do the refinement in a number of stages. In each stage processors (subdomains) are grouped in pairs. Paired subdomains will refine their boundaries without interference from others. To derive such a scheduling scheme is again the graph edge-colouring problem. As the amount of time taken by each pair in refining their boundary is usually proportional to the length of the boundary, the colouring should be done such that the sum of the largest length of the boundary of each stage is minimised.

In Section 2, the graph edge-colouring problem is looked at and the implementation of an algorithm which produces an edge-colouring scheme with no more that $deg + 1$ colours will be given. Secondly, an algorithm that attempts to minimise the sum of the largest weights of each colour is introduced.

In Section 3 the algorithms of Section 2 will be used to derived synchronous communication schemes and applied to message passing tasks resulting from the partitioning of irregular meshes.

In Section 4 the algorithms of Section 2 will be applied to a parallel finite element code and the effect of various algorithms on the execution time of the code will be discussed.

Notice that the synchronous communication scheme described before is only one way of organising the communication. Other alternative ways exist. Different ways of communication will also be compared the Section 4

Section 5 concludes the paper with further discussion.

2 Graph Edge-Colouring Problems and Algorithms

In this section the graph edge-colouring problem will be considered. An algorithm which colours the graph with no more than $deg + 1$ colours is first given. The problem of minimising the sum of the largest weight of each colour is then considered and an algorithm will be suggested that attempts to solve the problem.

Let $G = (V, E)$ be a graph, where V is the set of vertices and E the set of edges. The degree of a vertex is the number of edges that are linked to the

vertex. The degree of a graph, denoted by deg, is the maximum degree of all vertices of the graph. Two edges are said to be adjacent if they share the same vertex.

The edge-colouring problem can be defined as follows:

edge-colouring problem assign each edge of the graph G with a colour, such that no adjacent edges have the same colour.

The *chromatic number* of a graph, $\chi_1(G)$, is the least number of colours needed to edge-colour the graph. It is easy to see that at least deg colours are needed to colour the graph, so $\chi_1(G) \geq deg$. In fact an interesting theorem in graph theory, proved by Vizing [13], shows that

Theorem 1 *If G is a graph, then $deg \leq \chi_1(G) \leq deg + 1$.*

Therefore it is always possible to colour the edges of a graph with no more than $deg + 1$ colours. However to decide whether it is possible to colour the graph with only deg colours is an NP-hard problem. So a practical bound is $deg + 1$ colours.

The proof of the theorem (see, e.g., [4]) is based on adjusting the colouring of the graph. Let C be the set of $deg + 1$ colours. Assume that part of the graph is coloured with colours from C. Consider adding an uncoloured edge to the coloured part of the graph. If there is a common unused colour from the colour set C at both ends of the edge, then colour the edge with that colour. Otherwise adjust the colouring of the coloured part of the graph to allow the new edge to be coloured with no conflict.

The following algorithm, which colours a graph G with no more than $deg + 1$ colours, is based on the proof of the theorem. Here a path is denoted to be a list of vertices of the form $w_1, w_2, \ldots w_k$, where for $2 \leq i \leq k$, the pair (w_i, w_{i-1}) constitutes an edge in E and vertices are not repeated, apart from the case $w_1 = w_k$.

Algorithm COLOUR

- step 1 Let C be the set of $deg+1$ colours. Let G^C be the part of the graph whose edges were coloured using colours in C with no conflict.
- step 2 Find an edge $e = (v, w) \in G$ that is not in G^C, where v and w are two ends of the edge e. If no such edge exists then all the edges have been coloured with colours in C, so terminate.
- step 3 If there is a colour $c \in C$ unused at both ends v and w, then colour the edge $e = (v, w)$ with the colour c, add the edge and both ends to G^C, go to step 2. Otherwise let $c_1 \in C$ be an unused colour at v and $c_2 \in C$

be an unused colour at w.

- step 4 Starting from w, find the largest path in G^C coloured with colours c_1 and c_2 alternately. Let the path be $w, w_1, w_2, \ldots, w_k$.
- step 5 If $w_k \neq v$ then exchange the colours c_1 and c_2 in the path, colour the edge $e = (v, w)$ with c_1, add the edge and its ends to G^C, go to step 2.
- step 6 Otherwise, if there is a colour c_3 unused both at v and w_{k-1} then change the colour of the edge (v, w_{k-1}) from c_2 to c_3. Colour the edge $e = (v, w)$ with c_2, add e and its ends to G^C, go to step 2.
- step 7 Else let c_3 be an unused colour at w_{k-1}. Uncolour the edge (v, w_{k-1}) and delete it and its ends from G^C. Colour the edge $e = (v, w)$ with colour c_2 and add it and its ends to G^C. Rename w_{k-1} as w, c_3 as c_2, the edge (v, w_{k-1}) as e and repeat the procedure from step 4.

In practice there is usually more than one colouring scheme that edge-colours the graph with no more than $deg + 1$ colours. Let S be the set of all such colouring schemes. The cost of each colouring scheme, is defined as the sum of the largest weight of the edges that have the same colour. For example, in Figure 2 (a), edges (1,6), (2,5) and (3,4) have colour c_1, with a largest weight of $\max\{4, 1, 6\} = 6$; edges (1,2) and (6,5) have colour c_2, with the largest weight of $\max\{6, 1\} = 6$; edges (2,3) and (5,4) have colour c_3, with the largest weight of $\max\{1, 5\} = 5$, thus the cost of the colouring scheme given by Figure 2 (a) is $6 + 6 + 5 = 17$.

The minimum cost problem can be stated as following

minimum cost problem Among all colouring scheme $s \in S$, find one that has the minimum cost.

Although the algorithm COLOUR can give a colouring scheme with no more than $deg + 1$ colours, it may not be the one with minimum cost. To illustrate this, Table 3 presents a communication task table given by partitioning a mesh of 788 elements into 16 subdmains, using a spectral bisection algorithm. Thus the graph described by the table has 16 vertices. The maximum degree is $deg = 5$. Applying algorithm COLOUR results in the scheduling scheme given by Table 4. In this case 5 colours are used. If one views the table as a communication schedule, then in the 5-th stage, processor 10 exchanges messages with processor 13, which needs 10 units of time. Processors 11 and 12 exchange messages which need 2 units of time, the rest of the processors are idle. A better balancing of the communication load at each time step will reduce the overall communication costs.

The following algorithm is proposed in an attempt to solve the minimum cost problem. The idea is to colour as many as possible edges with large weights using one colour, then repeat the procedure recursively for the remaining edges.

The algorithm, named REDUCE, starts with a graph edge-colouring with no more than $deg+1$ colours. Roughly speaking, first the edge e_1 with the largest weight is found and locked, its colour is c_1. Then the unlocked edge e_2 with the next largest weight is found, its colour is c_2. An attempt is then made to colour the two edges e_1 and e_2 with the same colour. If $c_1 = c_2$, then the two edges with largest weights have indeed the same colour, so e_2 is locked. Otherwise the following steps are carried out to change the colour of e_2 into c_1. If at both ends of the edge e_2, colour c_1 is unused, then the colour of e_2 can simply be changed to c_1. Otherwise the logic used in the algorithm COLOUR can be adapted. The largest path of unlocked edges containing the edge e_2 and coloured alternately with c_1 and c_2 is found. The two colours on the path are exchanged making e_2 having the same colour as e_1. Then e_2 is locked. This process is repeated until all edges are locked. The edges with colour c_1 are then removed and the process repeated on the remaining graph. In this way edges with similar weights are coloured with the same colour. The detailed algorithm is as follows, where

$c(e)$ = the colour of edge e,
$E(G)$ = the set of edges of a graph G,
E_L = the set of locked edges,
E_U = the set of unlocked edges.

The algorithm starts with the graph G, and first colours it with the algorithm COLOUR.

Algorithm REDUCE

- step 1 Edge-colour the graph G with algorithm COLOUR.
- step 2 Let $E_L = \emptyset$, $E_U = E(G)$.
- step 3 If $E_U = \emptyset$ then terminate, otherwise let e_1 be the edge in E_U with the largest weight. let $c_1 = c(e_1)$, $E_L = E_L \cup \{e_1\}$, $E_U = E_U \setminus \{e_1\}$.
- step 4 If $E_U = \emptyset$ go to step 7, otherwise let e_2 be the edge in E_U with the largest weight, let $c_2 = c(e_2)$.
- step 5 If $c_2 = c_1$ then $E_L = E_L \cup \{e_2\}$, $E_U = E_U \setminus \{e_2\}$, go to step 4.
- step 6 Find the largest path that contains e_2 and is coloured with c_1 and c_2 alternately. If the path contains locked edges, then let $E_L = E_L \cup \{e_2\}$, $E_U = E_U \setminus \{e_2\}$, go to step 4. Otherwise exchange the colours c_1 and c_2 on the path, let $E_L = E_L \cup \{e_2\}$, $E_U = E_U \setminus \{e_2\}$, go to step 4.
- step 7 Remove the edges with colour c_1 from the graph. Rename the remaining graph as G, go to step 2.

This algorithm is a descent algorithm, in the sense that given a colouring scheme of a graph, applying REDUCE to it will always produce a colouring scheme with a cost not more than that of the original scheme. In the actual implementation of the algorithm, if the cost is reduced, then the algorithm is

restarted on the new coloured graph. This process is repeated until no further reduction is possible.

An example is given by the graph in Figure 2 (a). The process of applying REDUCE to the graph is shown in Figure 3. As a further example, applying the algorithm to the scheduling scheme of Table 4 gives the scheduling scheme given by Table 5, the cost is reduced from 36 to 26 – a significant saving.

When the algorithm REDUCE terminates, it is not necessary that a global minimum has been found. For that reason we will also test an algorithm which takes the best result for ten runs of the algorithm REDUCE, each run starts with a randomly permuted edge and vertex indices. This algorithm is denoted as REDUCE10.

The three algorithms are applied to the communication task graphs resulting from partitioning five meshes of various sizes using a spectral bisection algorithm. The number of colours and the cost of the colouring scheme generated by the three algorithms are given in Table 6.

From Table 6 it is clear that both the algorithms REDUCE and REDUCE10 improve significantly the cost of the colouring scheme given by the algorithm COLOUR.

3 Simulation on a Parallel Computer

In this section the scheduling schemes given by the edge-colouring algorithms are applied in organising the communication for an irregular mesh based calculation. Simulations were done on 16 nodes of a Intel iPSC/860 parallel computer using the scheduling schemes to evaluate the practical effect of the edge-colouring algorithms.

On the Intel, there are two ways of sending and receiving messages. When the *blocked* (or *synchronous*) send/receive subroutines **csend/crecv** are called, the program does not leave the calls until the sending/receiving action is completed. When the *unblocked* (or *asynchronous*) send/receive subroutines **isend/irecv** are called, the program immediately leave with a message id, and carries on executing other parts of the code. A **msgwait** subroutine with the appropriate message id can be called at a later stage to make sure that the messages have been actually sent/received.

On an Intel iPSC/860, the communication time needed to **csend/crecv** a

message is a linear function of the length. More specifically [6],

$$t = \begin{cases} 73 + 0.42 * n, & \text{if } n \leq 100, \\ 175 + 0.36 * n, & \text{if } n > 100, \end{cases} \tag{1}$$

where t is the communication time in microseconds, n is the number of bytes in the message. In the above model the effect of the distance between processors and possible link contention are not included.

Given a scheduling scheme such as that of Table 5, generated by an edge-colouring algorithm, the pseudo code for the synchronous pair-wise exchange type communication scheme on any processor me is

do $i = 1, num\_stages$
  csend/crecv message of length $msglen(me, i)$
   to/from processor $adjproc(me, i)$
  synchronisation
end do

Thus the communication is done in a number of stages, at each stage a processor pair exchanges messages, and this is followed by a synchronisation.

The cost given by a colouring scheme, such as those presented in Tables 4 and 5, are just predictions of the actual cost that the scheduling scheme might give. In order to see if these predictions have any significance in practice, simulations have been run on the Intel i860 hypercube. Two meshes are considered and five partitioning algorithms [7] are used to partition each mesh into 16 subdomains. For each partitioning, three message passing schemes are generated using the three scheduling algorithms COLOUR, REDUCE and REDUCE10. The five partitioning algorithms used are: the recursive coordinate bisection (RCB) , the recursive graph bisection (RGB), the recursive spectral bisection (RSB) (see, e.g. [11]) for these three algorithm), the KL algorithm [9] and a hybrid algorithm of KL and RGB, which is called MINGRAPH [7].

For each shared face between two subdomains, it is assumed that 10 double precision numbers need to be exchanged. Thus for example, in Table 4, node 7 exchanges 30 double precision numbers with node 4 at stage 1, and 40 double precision numbers with node 13 at stage 2, etc.. After each stage, a global synchronisation step is executed by calling the Intel i860 FORTRAN subroutine **gsync()**, to ensure that each node has finished sending and receiving data, before proceeding to the next stage. The message passing is repeated 1000 times, and the communication time is taken to be the elapsed time between the start and the finish of the message passing. In Table 7, the number of

stages, the communication costs (predicted) and the actual communication time (in milliseconds, or ms for short) recorded in the simulation are reported. For the partitioning of the 788 mesh given by the recursive coordinate bisection, Algorithm COLOUR gives a message passing scheme with 7 stages and a predicted communication cost of 47 units. The actual communication time is about 8.4 seconds. This can be reduced to about 7.0 seconds by using Algorithm REDUCE10. In general, Table 7 illustrates that the predicted costs reflect the trend of actual communication times very well, and that the predicted reductions of communication costs by the Algorithms REDUCE and REDUCE10 are actually delivered in the measured communication time.

It is also interesting to compare the communication time given by various partitioning algorithms. The recursive coordinate bisection algorithm (RCB) on the 5520 mesh needs at least 35 seconds, almost 3 times greater than the best communication time given by the MINGRAPH algorithm. The partitioning algorithm KL performs badly on the 788 mesh, mainly because the algorithm generates disconnected subdomains which border a greater number of neighbouring subdomains. This increased the number of stages for the communication, therefore the start-up and synchronisation costs.

It is interesting to relate the communication time in Table 7 to formula (1). The synchronisation costs for one call of the synchronisation routine **gsync()** on a 16 node configuration of the Intel is found to be about 530 microseconds. Assuming no link contentions, the communication time (ms) for executing k stages of synchronous pair-wise exchange 1000 times is

$$t = 202 * k + 0.36 * \sum_{i=1}^{k} n_i + 530 * k, \tag{2}$$

where n_i is the maximum number of bytes of message transferred in stage i. For example, in Table 5, the maximum number of shared edges at each stage is 6, 8, 6, 6 and 10 respectively. Thus if 10 double precision numbers are transferred for each shared edge, then the message length will be 480, 640, 480, 480 and 800 bytes respectively, so $t = 5226$ ms. For the communication task given by Table 6, we calculated that $t = 4939$ ms. These figures are proportional but less than the actual communication time (6286 ms and 6052 ms) found in Table 9. The under-prediction is probably due to the link contentions. In both cases, because the message lengths are short, the start-up and synchronisation account for majority of the time.

For large meshes, the start-up and synchronisation costs are less dominant. For example on the partitioning of the mesh-5520 using RSB, the message passing algorithm COLOUR gives a scheme of 5 stages, with a sum of largest message lengths of 385. This gives a calculated $t = 14748$ ms. The message passing Algorithm REDUCE10 gives a scheduling scheme of 4 stages with a

sum of largest message lengths of 216, and a calculated $t = 9148$ ms. Again the two calculated timings are proportional to, but under estimate the actual communication times (which are 22200 ms and 15010 ms respectively, from Table 9). Between 25−32% of time is spent in the start-up and synchronisation.

4 Effect of the Scheduling Algorithms on the Finite Element Code

The scheduling algorithms are implemented in an explicit unstructured finite element Euler solver code FELISA (Peiro et al.). The code was run on a number of parallel platform, including the Intel iPSC/860 and the Cray T3D.

Apart from the synchronous communication scheme discussed in the previous sections, there are many alternative ways of organising the communication. The following is an asynchronous communication scheme, which uses the asynchronous communication subroutines **isend/irecv** available on the Intel.

do $i = 1, num\_neighbours$
  irecv message of length $msglen(me, neighbour(i))$ from processor $neighbour(i)$, message id $msg(i)$
end do
do $i = 1, num\_neighbours$
  isend message of length $msglen(me, neighbour(i))$ to processor $neighbour(i)$
end do
do $i = 1, num\_neighbours$
  call **msgwait**$(msg(i))$
end do

In the above code, the asynchronous receive is posted first, followed by sending the messages to all the neighbours. Finally a message waiting subroutine is called to ensure that the messages are received. The advantage of this asynchronous communication scheme, compared with the synchronous scheme detailed in the last section, is that there is no need to organise the communication in a pair-wise fashion. Furthermore no synchronisation is needed, and a processor can carry on its computation as soon as it has received messages it needs, without waiting for other processors to finish. The disadvantage is that the receiving/sending buffer space can not be reused. For example the receiving buffer space needed will be the sum of the message lengths from all the neighbouring processors, while if synchronous communication is used then the receiving buffer size needed is the largest message length to be expected from neighbouring processors. The

asynchronous communication is not supported on all parallel platforms, thus a program uses asynchronous communication would be less portable than one that uses synchronous pair-wise exchange.

The asynchronous algorithm, and the synchronous algorithm combined with the two edge-colouring algorithms COLOUR and REDUCE, are compared as communication schemes for the finite element code. The particular problem solved is the flow field around a aircraft, the mesh has 353710 tetrahedra. The mesh is partitioned using the spectral bisection algorithm into subdomains with equal numbers of tetrahedra. The finite element code takes 1173 iterations to converge.

It is noted that although each subdomain has an equal number of tetrahedra, the time spent on computing is found to vary between different processors. This is because for the particular finite element code used, the amount of computation on each processor is not only related to the number of tetrahedra on the subdomain, but also to other factors such as the number of boundary points. The latter are difficult to control statically. If the communication time t_{comm} is defined as the total elapsed time t_{total} for the code minus the computation time, then the communication time also varies. This variation is due to the time spent in waiting. For the synchronous communication schemes, each processor has to wait for all the processor to finish computing before communication can begin, and it also has to wait for all processors to finish communication before further computation can start. For the asynchronous schemes a processor needs to wait for its neighbouring processors to finish computation before it is able to receive messages from them, but can carry on its computation as soon as it has received the messages it needs, even if other processors are still communicating or waiting. Asynchronous communication schemes have the ability to overlap computation with communication, which reduces the total execution time of running the program.

The communication time (including waiting time) thus varies between processors. In order to give an indication of the proportion of the communication (and waiting) time to that of the total elapsed time, in Table 8 t_{total} and the averaged/minimum/maximum t_{comm} are reported. For the synchronous communication, the optimised edge-colouring algorithm REDUCE is able to reduce the total time of the finite element code by 8-13%. When asynchronous communication is used, a further reduction of 14-30% can be found.

5 Conclusions

In this paper algorithms have been implemented to generate edge-colouring schemes for graphs, which need at most $deg + 1$ colours, where deg is the degree of the graph. An algorithm which attempts to minimise the sum of largest weight has been suggested.

The algorithms were used to organise communication for irregular grid type applications. The optimised algorithm was found to improve the communication time of the synchronous communication scheme. Synchronous and asynchronous communication schemes were also compared and it was found that the latter reduces the total elapsed time of running a finite element code, at the expense of increased communication buffer size and reduced portability of the code.

Parallel KL algorithm for mesh partitioning is a potential application of the graph edge-colouring algorithms presented in this paper which deserves further investigation.

Acknowledgement

This work is sponsored by ICI PLC through a collaborative project with the Advanced Research Computing Group at Daresbury Laboratory.

References

[1] S. T. Barnard and H. D. Simon, Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, *Concurrency: Practice and Experience* **6** (1994) 101-117.

[2] P. Diniz, S. Plimpton, B. Hendrickson and R. Leland, Parallel algorithms for dynamically partitioning unstructured grids, in: D.H. Bailey, P. E. Bjorstad, Jr Gilbert, M. V. Mascagni, R. S. Schreiber, H. D. Simon, V. J. Torczon, J. T. Watson, eds., *SIAM Proceedings Series* **195** (SIAM, Philadelphia, 1995) 615-620.

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

[4] R. Gould, *Graph Theory* (The Benjamin/Cummings Publishing Company, 1988).

[5] B. Hendrickson and R. Leland, The Chaco User's Guide, Version 1.0, Technical Report SAND 93-2339, Sandia National Laboratories, Allbuquerque, NM., 1993

[6] A. J. G. Hey, R. Hockney, V. Getov, I. Wolton, J. Melin and J. Allwright, The Genesis distributed memory benchmarks. Part2: COMMS1, TRANS1, FFT1 and QCD2 benchmarks on the SUPERNUM and iPSC/860 computers, *Concurrency: Practice and Experience* **7** (1995) 543-570.

[7] Y .F. Hu and R. J. Blake, Numerical experiences with partitioning of unstructured meshes, *Parallel Computing* **20** (1994) 815-829.

[8] G. Karypis and V. Kumar, Parallel multilevel graph partitioning, Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, 1995.

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

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

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

[12] V. Venkatakrishnan, H. D. Simon and T. J. Barth, A MIMD implementation of a parallel Euler solver for unstructured Grids, *Journal of Supercomputing* **6** (1992) 117-137. NASA Ames Research Center.

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

[14] C. Walshaw, M. Cross, S. Johnson and M. Everett, A parallelisable algorithm for partitioning unstructured meshes, in: *Proc. Irregular'94: Parallel Algorithms for Irregularly Structured problem.*

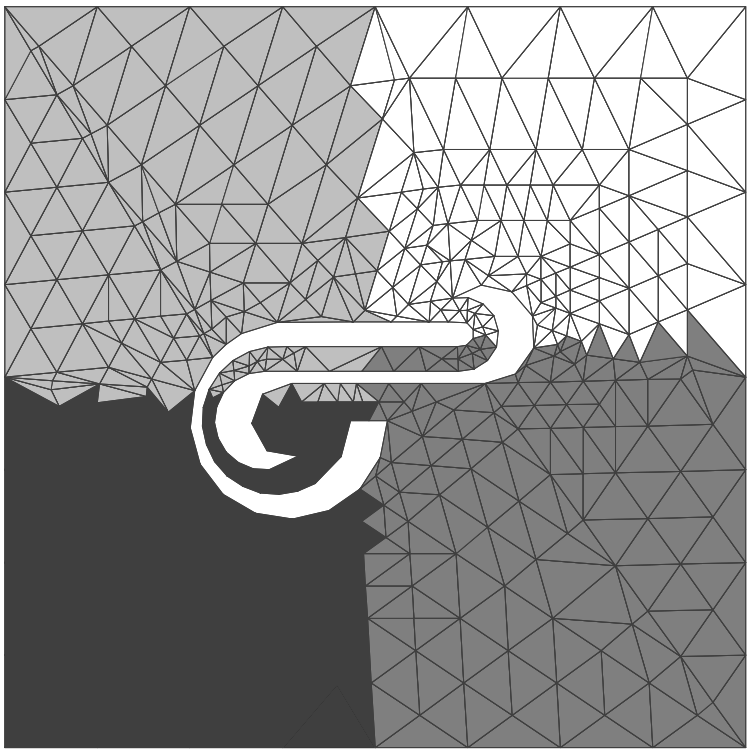

Figure 1 (a) Partitioning of a mesh into 4 subdomains using recursive coordinate bisection

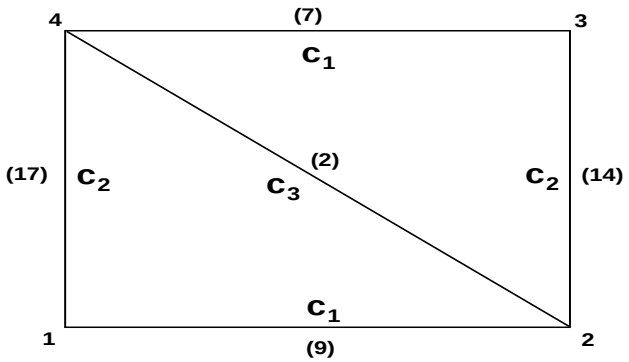

Figure 1 (b) Corresponding communication task graph of Figure 1 (a)

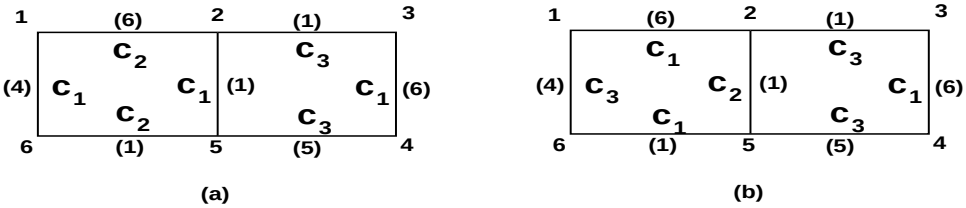

Figure 2 Two scheduling schemes of the same communication task.
(a)communication cost $= 6 + 6 + 5 = 17$;
(b) communication cost $= 6 + 1 + 5 = 12$.

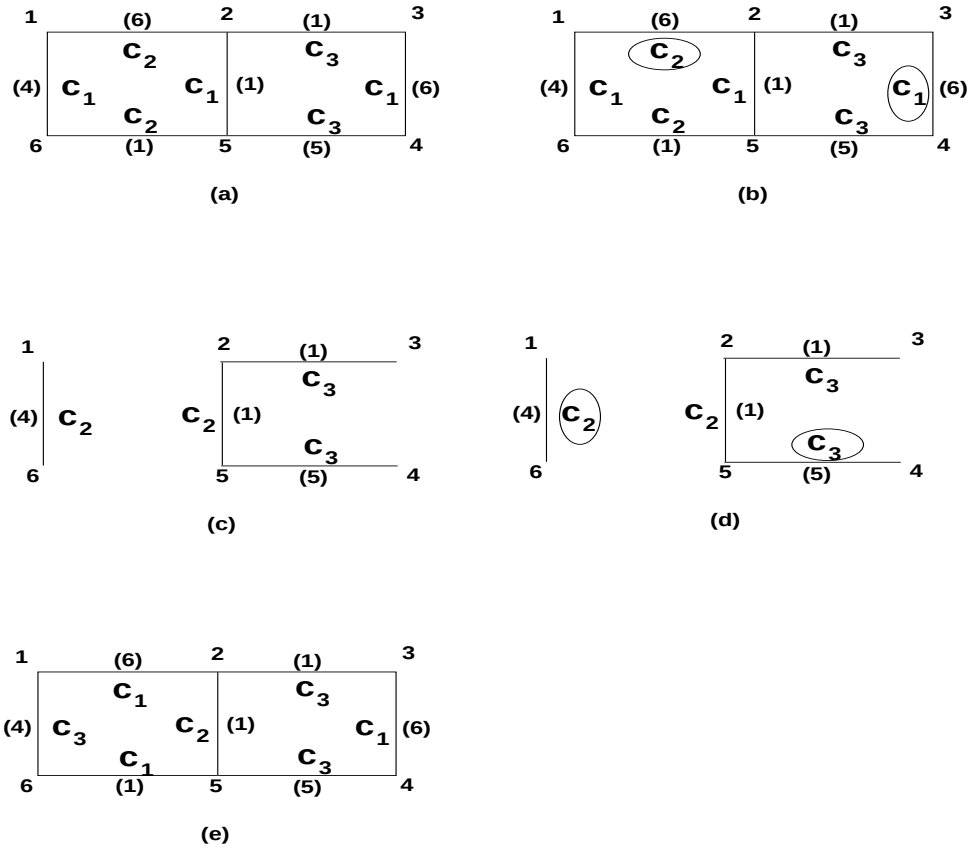

Figure 3 Applying algorithm REDUCE to a communication task graph.
(a) Communication cost= $6 + 6 + 5 = 17$;
(b) $e_1 = (3,4)$, $e_2 = (1,2)$, path= 1, 2, 5, 6. Exchange colours c_1 and c_2 on the path. Lock e_1 and e_2;
(c) No more improvement possible. Remove colour c_1;
(d) $e_1 = (5,4)$, $e_2 = (1,6)$, change colour of e_2 to c_3. Lock e_1 and e_2;
(e) No more reduction possible, final scheme with communication cost $= 6 + 1 + 5 = 12$.

Table 1 A communication task table

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

Table 2 A scheduling scheme

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

Table 3 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)	–	–

Table 4 An initial scheduling scheme

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

Table 5 A scheme with reduced communication cost

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

Table 6 Comparison of three edge-colouring algorithms by the "number of colours/cost", on the communication task graphs resulting from the partitioning of five meshes using a recursive spectral bisection algorithm. p is the number of sudomains

p	Mesh	COLOUR	REDUCE	REDUCE10
16	771	6/44	6/34	6/30
16	788	5/36	5/26	5/25
16	3564	6/74	6/59	5/56
16	5520	5/385	4/216	4/216
32	353710	17/4678	17/2042	17/2028
64	353710	28/4140	28/1853	28/1823
128	353710	36/3433	36/1214	36/1196
256	353710	37/2715	37/930	37/902

Table 7 Predicted and simulated communication costs*

Par. methods	Mesh	COLOUR	REDUCE	REDUCE10
RCB	788	7/47/8423	6/35/6958	6/31/6992
RCB	5520	7/842/45582	7/666/40006	7/521/35662
RGB	788	6/47/7580	5/34/6103	5/32/5975
RGB	5520	7/598/33528	7/511/30266	7/427/28418
RSB	788	5/36/6286	5/26/6052	5/25/5634
RSB	5520	5/385/22200	4/216/14778	4/216/15010
KL	788	10/52/10935	9/36/9424	9/36/9366
KL	5520	6/326/20345	6/263/19170	6/256/18471
MINGRAPH	788	6/40/7230	6/29/6838	6/28/6505
MINGRAPH	5520	4/209/14136	3/189/13582	3/189/13020

* Simulation: message passing 1000 times using the schemes given by the three scheduling algorithms on the partitions generated with the five partitioning methods. For each shared face between two subdomains 10 double precision numbers are exchanged. The results are reported in the form

number of stages/communication costs (units)/communication time (ms)

Table 8 The total elapsed time t_{total} and and communication (including waiting) time t_{comm} for the finite element code to converge on a mesh around an aircraft, three communication schemes are compared. $p = 32$ and 64 is the number of processors.

p	comm. scheme	t_{total}	t_{comm} (average/min/max)
32	COLOUR	7922.70	3528.76/2046.63/4052.41
32	REDUCE	7255.69	2854.82/1376.33/3406.18
32	asynchronous	6239.69	1775.78/289.56/2334.00
64	COLOUR	5560.26	3437.08/2624.29/3837.24
64	REDUCE	4843.32	2721.01/1919.71/3112.97
64	asynchronous	3373.56	1169.84/357.39/1574.72