Partitioning & Mapping of Unstructured Meshes to Parallel Machine Topologies

C. Walshaw, M. Cross, M. G. Everett, S. Johnson, and K. McManus *

Parallel Processing Group, Centre for Numerical Modelling & Process Analysis,
University of Greenwich, London, SE18 6PF. E-mail: C.Walshaw@gre.ac.uk

Abstract. We give an overview of some strategies for mapping unstructured meshes onto processor grids. Sample results show that the mapping can make a considerable difference to the communication overhead in the parallel solution time, particularly as the number of processors increase.

1 Introduction

The use of unstructured mesh codes on parallel machines can be one of the most efficient ways to solve large Computational Mechanics problems. Completely general geometries and complex behaviour can be readily modelled and, in principle, the inherent sparsity of many such problems can be exploited to obtain excellent parallel efficiencies. An important issue for such codes is the problem of distributing the mesh across the memory of the machine at runtime so that the computational load is evenly balanced and the communication overhead is minimised. It is well known that this problem is NP complete, so in recent years much attention has been focused on developing suitable heuristics, and some powerful methods, many based on a graph corresponding to the communication requirements of the mesh, have been devised, e.g. [2].

A pertinent but often ignored factor in parallel processing is the underlying topology of the machine's interconnection network. For example, even on machines with small numbers of processors, it is possible to detect variations between the latencies of processors which are closely linked and those which are 'far apart'. Although most machines now have facilities for 'wormhole routing' (i.e. the passing of messages between two non-adjacent processors without interrupting intermediate processors), high contention of the interprocessor links can result if adjacent partitions are mapped to, say, opposite corners of a processor array. As the trend towards massively parallel machines continues, these effects are likely to be exacerbated and the machine topologies will have an increasingly important effect on the parallel overhead arising from any given partition. Most of the current generation of mesh partitioning algorithms, however, take no account of the topology. The mapping to the machine is either treated as a post-processing step, applied after the data has been partitioned, or even ignored. For some machines with small numbers of processors this may be a legitimate simplification, but as machine sizes increase it is likely that a poor mapping will cause significant performance degradation.

* In: A. Ferreira and J. Rolim, editors, Proc. Irregular '95: Parallel Algorithms for Irregularly Structured Problems, volume 980 of LNCS, pages 121-126. Springer, 1995.

1.1 Overview

The strategy employed here to tackle the partitioning/mapping problem is to derive an initial partition quickly and cheaply as possible and then use powerful optimisation techniques to achieve a high quality solution. This multi-stage approach has been shown to provide an efficient and flexible approach to partitioning, [6] and is similar to the work of Vanderstraeten *et al.*, [5], although the techniques vary in that we employ deterministic heuristics to optimise the partition.

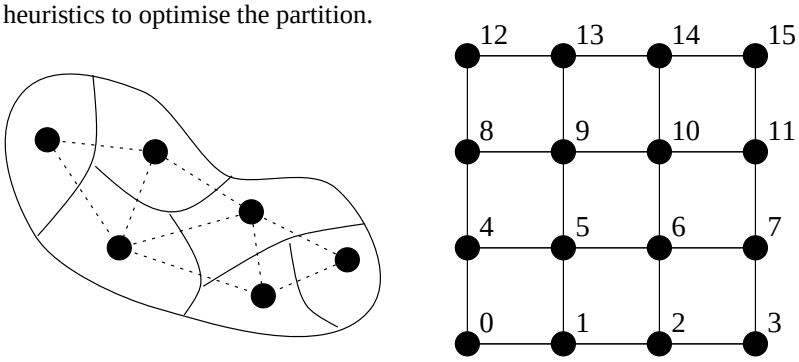

Fig. 1. A typical partition with subdomain graph and a 4×4 processor grid

We use an undirected graph $G(N, E)$, of N nodes & E edges, to represent the data dependencies arising from the unstructured mesh. Any partition of G induces a subdomain graph S and loosely the mapping problem can be thought of as the placing of this S onto the processor topology such that the communication overhead is minimised. We concentrate here on mapping onto a grid topology where we assume that the processors are connected as a 1D, 2D or 3D array. This is a realistic restriction as grids can be found in some of the current range ofparallel machines such as the Intel Paragon (2D) or Cray T3D (3D). Figure 1 shows a typical partition, the resulting subdomain graph and a 2D grid topology.

2 The initial partition

The aim of the initial partitioning is to divide up the graph as rapidly as possible prior to optimisation where most of the work takes place. We use two different initial partitioning algorithms; the Greedy Algorithm ignores the processor topology completely, whilst the other, Geometric sorting, does a very crude mapping onto a processor grid.

2.1 The greedy algorithm

The Greedy algorithm used here is a simple variant of that originally proposed by Farhat and fully described in [1]. It derives its name from the way in which it 'bites' into the mesh; each fresh partition grows out in level sets from a seed node until an appropriate proportion of the graph has been 'eaten' and the next partition is then seeded (if possible)

from a node on the border of the previous subdomain. This is clearly seen to be the fastest *graph-based* method as it only visits each graph edge once. However, it takes no account of the processor topology except for the fact that two contiguously numbered domains are likely to be (but not necessarily) adjacent. The variant employed here differs from that proposed by Farhat only in that it works solely with a graph rather than the nodes and elements of a finite element mesh.

2.2 Geometric sorting

This simple and intuitive algorithm does not use graph connectivity information, but instead partitions solely on the geometric coordinates of the nodes. Thus, to map a graph onto an $p \times q$ processor grid (where $p \geq q$) the nodes are first sorted by x-coordinate, say, and split into p sets each of weight N/p. The nodes of each of these sets are then sorted by y-coordinate and split into sets of N/pq. Of course, neglecting connectivity information may result in a very poor quality partition and/or mapping, but if nodes which are adjacent in the graph are also adjacent geometrically, as is frequently the case in graphs arising from finite element/finite volume discretisations, it can be very successful.

For the results reported in this paper the choice of which coordinate to sort on first is left to the user. In fact, using x, y, z-coordinates may not be ideal as it takes no account of the orientation of the mesh and a more successful technique might be to determine the principal axes of inertia of the graph nodes (as in the commonly used Recursive Inertial Bisection – see for example, [4]).

3 The optimisation methods

Once the graph is partitioned, optimisation can take place to improve the quality of the partition. The two methods outlined here have different aims; 'uniform optimisation' treats the processor topology as uniform and tries to minimise the number of interprocessor cut-edges. 'Grid optimisation', on the other hand, treats the processor topology as a grid and attempts to optimise the mapping by eliminating non-local communications. Throughout the optimisation, it is assumed that the final partition will not deviate too far from the initial one. Thus, in general, only border nodes are allowed to migrate to neighbouring subdomains.

3.1 Uniform optimisation

This algorithm is fully described in [6] where it is seen that a key part of the technique is the way in which each subdomain tries to minimise its own surface energy. In the physical 2D or 3D world the object with the smallest surface to volume ratio is the circle or sphere. Thus the idea behind the subdomain heuristic is to determine the centre of each subdomain (in some graph sense) and to then measure the radial distance from the centre to the edges and attempt to minimise this by migrating nodes which are furthest from the centre.

Determining the 'centre' is relatively easy and can be achieved by moving in level sets inwards from the subdomain border until all the nodes have been visited. The final

set defines the centre of the subdomain and, the reverse of this process can then be used to determine the radial distance. Having derived these sets each node can be marked by its radial distance. The code finally decides which nodes to migrate based on a combination of radial distance, load-imbalance and the change in cut-edges.

3.2 Grid optimisation

The grid optimisation algorithm is based very much on the uniform optimisation algorithm with some minor changes and a more appropriate method for minimising the surface energy. In summary, the minor changes are that subdomains can only migrate nodes to their neighbours in the *processor grid* and that, when assessing the cost of a partition, interprocessor edges are weighted according to how close together they are in the processor array. We use the square of the number links between the two processors, so that for example, in Figure 1, the cost of an interprocessor edge between 0 and 6 is $9 = (2+1)^2$.

After some experimentation it was found that using the radial distance as a basis for migrating nodes which are far from the subdomain centre was simply not appropriate for achieving a grid mapping, as nodes which are relatively far away from the centre of the subdomain may be well placed for the topology mapping. To see this, consider the partition of the unit square for a 1D processor array where the topology preserving partition is just a series of strips. Migrating nodes which are far away from the centre of the subdomain (i.e. at the extremes of each strip) does not preserve the partition as a 1D array. However, if we attempt to minimise the width of each strip, rather than the radial distance, we do find that the partition can preserve the machine topology. Thus, instead of measuring the radial distance of the subdomain, we measure (in a graph sense) the distance between the borders with processor on the left and the processor on the right.

This technique can also be extended to higher dimensional arrays by each processor classifying the other processors as lying, in the 2D case, to either the north, south, east or west, with processors lying on a diagonal falling into two sets. Thus in Figure 1, relative to 5, processors 0, 4 & 8 are positioned to the west, 2, 3, 6, 7, 10, 11 & 15 to the east, 0, 1 & 2 to the south and 8, 9, 10, 12, 13, 14 & 15 to the north. After measuring the width in each direction border nodes are marked with the maximum of the east/west and north/south distance and in the case where a subdomain does not have nodes on one of the borders (e.g. in Figure 1 processors 0, 1, 2 & 3 do not have a southern border) nodes on the opposite border are simply marked with 0. Nodes are then migrated as for uniform optimisation (as described in [6]).

4 Mapping strategies

Table 1 shows the four mapping strategies tested. The *unmapped* partitioning completely ignores processor topology as does the *postmapped*, although it additionally employs a simple processor allocation algorithm at the end. This algorithm continually swaps subdomains between processors until no further improvement in the map cost is possible. The *premapped* partitioning method works the other way round; the graph is initially mapped, albeit crudely, onto the processor grid and then optimised to minimise the number of interprocessor cut-edges. Because the final partition does not deviate *too*

Strategy	Initial partition	Optimisation	Processor Allocation
Unmapped	Greedy	Uniform	No
Postmapped	Greedy	Uniform	Yes
Premapped	Geometric sort	Uniform	No
Partition mapped	Geometric sort	Grid	No

Table 1. Mapping strategies

much from the initial one the resulting subdomain graph still 'fits' reasonably well onto the processor grid. Indeed, although processor allocation was not used for these results, in tests it was very rare that it could find better allocations. The premapping would also be far more attractive than postmapping if the optimisation algorithm were running in parallel as the remapping of a distributed partition can involve a vast exchange of data, with a resulting loss of efficiency. Finally the *partition mapping* strategy acknowledges the processor topology throughout.

5 Results

We have tested the mappings using a parallel control volume unstructed mesh flow and stress code developed at Greenwich and described in [3]. The test mesh came from a casting simulation and the resulting graph contains 30,064 nodes and 44,693 edges. The parallel code was run on a Transtech Paramid with i860 processing nodes with an topology can be best modelled by a $p \times 2$ grid.

	Partition			Time (s)	
Strategy	E_c	C	D_a	Partition	Solution
Unmapped	939	14753	4.17	5.13	154.1
Postmapped	939	2225	4.17	5.19	148.9
Premapped	982	1598	3.92	6.88	144.1
Partition mapped	1061	1069	3.17	11.39	131.7

Table 2. Results for $P = 24$

Table 2 show some typical partitioning results for $P = 16 := 8 \times 2$. Here E_c is the number of cut edges, C is the partition cost using the distance-squared weighting described in §3.2 and D_a is the average degree of each subdomain. Unfortunately, however, when compared for several values of P, none of these measures could really be used to predict the parallel solution time and a future aim is to derive a good cost metric. The timings for the partition come from a Sun 20 workstation and are generally of the same order (although partition-mapping is always more expensive). However, we don't consider these figures particularly relevant as it is intended to parallelise the partition optimisation algorithm. Far more interesting are the parallel solution times and figure 2

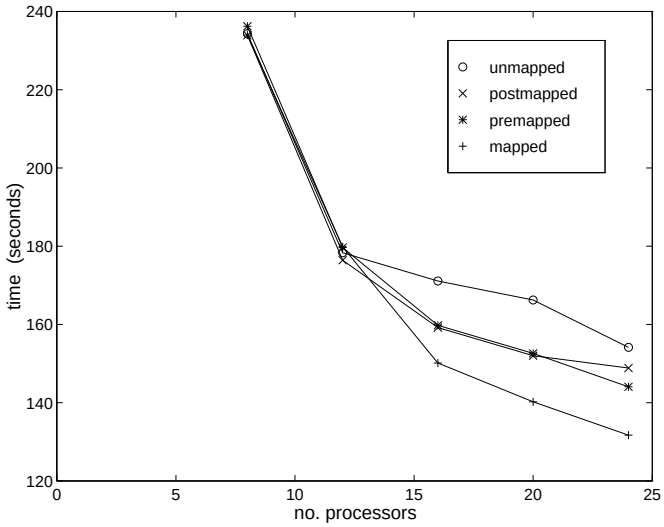

Fig. 2. Solution times

shows clearly how much difference a good mapping can make. Although the results are indistinguishable for small numbers of processors, as might be expected, they rapidly separate out for 16 or more.

6 Conclusion

The preliminary results of our investigation show that a good mapping can make a considerable difference to the communication overhead in the parallel solution time for unstructured meshes, particularly as the number of processors increases. Future work on this technique will include the parallelisation of the optimisation algorithms and an attempt to derive a meaningful cost function.

References

1. C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Comp. & Struct.*, 28:579–602, 1988.
2. C. Farhat and H. D. Simon. TOP/DOMDEC – a Software Tool for Mesh Partitioning and Parallel Processing. Tech. Rep. RNR-93-011, NASA Ames, Moffat Field, CA, 1993.
3. K. McManus, M. Cross, and S. Johnson. Integrated Flow and Stress using an Unstructured Mesh on Distributed Memory Parallel Systems. In *Parallel CFD'94*. Elsevier, 1995.
4. D. Roose and R. Van Driessche. Distributed Memory Parallel Computers and Computational Fluid Dynamics. Rep. TW 186, Dept. Comp. Sci., Katholieke Universiteit Leuven, 1993.
5. D. Vanderstraeten and R. Keunings. Optimized Partitioning of Unstructured Computational Grids. *Int. J. Num. Meth. Engng.*, 38:433–450, 1995.
6. C. Walshaw, M. Cross, and M. Everett. A Parallelisable Algorithm for Optimising Unstructured Mesh Partitions. Tech. Rep. 95/IM/03, University of Greenwich, London SE18 6PF, UK, 1995. (submitted for publication).