Evaluation of the JOSTLE mesh partitioning code for practical multiphysics applications

K. McManus[*], C. Walshaw[*], M. Cross, P. Leggett, and S. Johnson

Parallel Processing Group, Centre for Numerical Modelling & Process Analysis,
University of Greenwich, London, SE18 6PF.
email: [k.mcmanus, ..]@gre.ac.uk ; URL: http://www.gre.ac.uk/~[k.mcmanus, ..]

The use of unstructured mesh codes on parallel machines is one of the most effective ways to solve large computational mechanics problems. Completely general geometries and complex behaviour can be modelled and, in principle, the inherent sparsity of many such problems can be exploited to obtain excellent parallel efficiences. However, unlike their structured counterparts, the problem of distributing the mesh across the memory of the machine, whilst minimising the amount of interprocessor communication, must be carefully addressed. This process is an overhead that is not incurred by a serial code, but is shown to be rapidly computable at run time and tailored for the machine being used.

1. INTRODUCTION

Multiphysics simulations integrate the solution of interacting physical processes to solve complex inhomogeneous models, such as, metals casting and aeroelasticity. The University of Greenwich is developing a three dimensional unstructured mesh code, PHYSICA [3], which brings together into one toolkit the modelling of many processes such as:

turbulent multiphase fluid flow
phase changes (i.e. melting/solidification)
free surface flows
fluid-structure interaction
magnetohydrodynamics
elasto-viscoplasticity
structural dynamics
contact analysis

This code is being parallelised for Distributed Memory Multi Instruction Multi Data (DM MIMD) architectures using explicit message passing in Fortran77 [5].

Partitioning of an unstructured mesh into P partitions that are mapped to P processors is well known to be NP complete. Many methods have been developed that partition a graph corresponding to the communication requirements of the mesh. A new method for solving this graph-partitioning problem has been devised at the University of Greenwich and encapsulated in a software tool, JOSTLE [7]. It employs a combination of techniques

[*]Sponsored by the Engineering and Physical Science Research Council.

to give a rapid initial partition together with a clustering technique to further speed up the process. The resulting partitioning method is designed to work efficiently in parallel as well as sequentially and can be applied to both static and dynamically refined meshes. In this paper we present results obtained by the JOSTLE procedure for parallel multiphysics applications on unstructured meshes.

2. THE JOSTLE MESH-PARTITIONING CODE

The underlying strategy of the JOSTLE code is based on the continuing trends of research issues and computing resources. As mesh and machine sizes grow, the need for parallel load-balancing becomes increasingly acute. For small meshes (N nodes) and small machines (P processors), an order N overhead for the mesh partitioning may be considered reasonable. However, for large N and P, this order of overhead will rapidly become unacceptable if the solver is running at $O(N/P)$.

In addition, it is often the case that the mesh is already distributed across the memory of the parallel machine. For example, parallel mesh generation codes or solvers which use parallel adaptive refinement give rise to such distributed meshes, and in these cases it is extremely expensive to transfer the whole mesh back to a single processor for sequential load-balancing, if indeed the memory of that processor allows it.

To tackle these issues efficiently, the strategy developed here is to derive a partition as quickly and cheaply as possible, distribute the mesh and then *optimise* the partition *in parallel*. If the mesh is already distributed then the existing partition is used and optimisation can commence immediately. Experiments, on graphs with up to a million nodes, indicate that the JOSTLE procedure is up to an order of magnitude faster than existing state-of-the-art techniques such as Multilevel Recursive Spectral Bisection [1].

2.1. Topology mapping

A pertinent but often ignored factor in parallel processing is the underlying topology of the machine's interconnection network. 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 passing 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 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 machine 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 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.

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 produces a graph S describing sub-domain connectivity 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. Figure 1 shows three possible partitions of a mesh along with the resulting sub-domain graphs S. 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 of parallel machines such as the Intel Paragon (2D) or Cray T3D (3D).

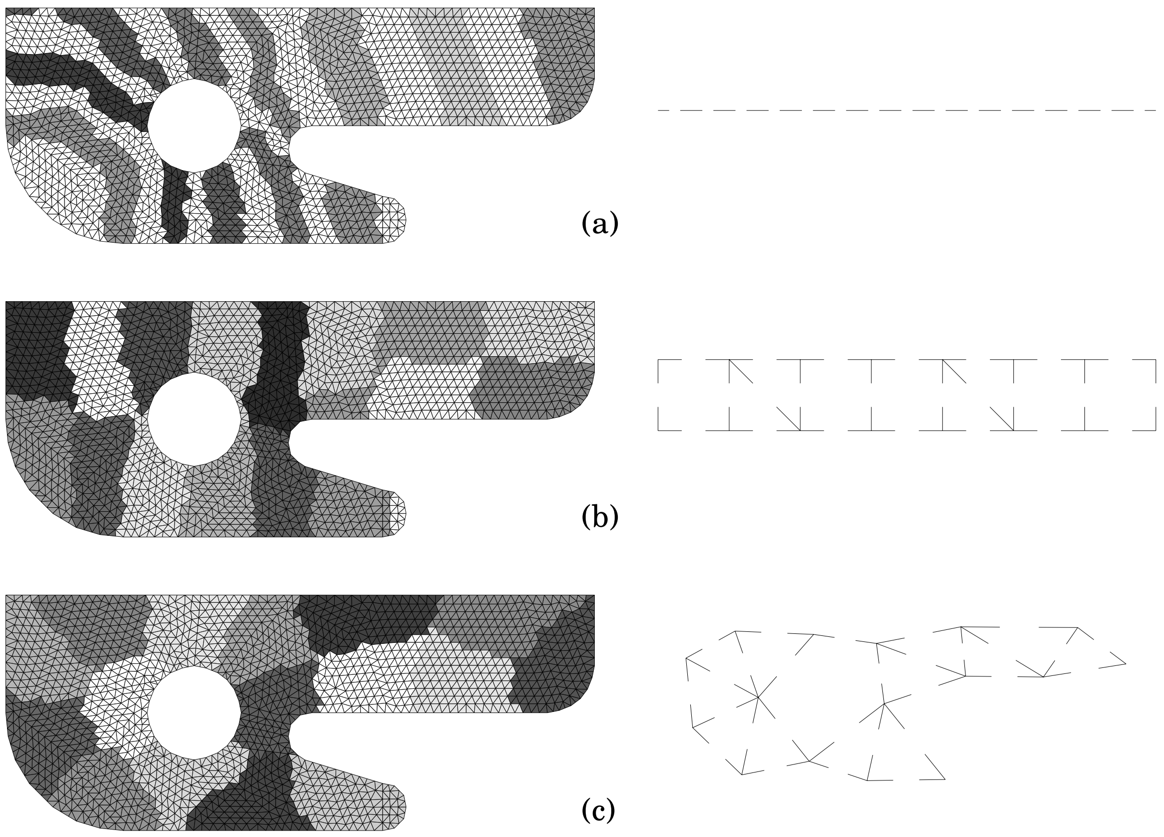

Figure 1. Partitions of a 2D mesh into (a) 1D, (b) 2D and (c) uniform topologies with the corresponding sub-domain connectivity graphs.

2.2. 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.

The Greedy algorithm used here is a simple variant of that originally proposed by Farhat and fully described in [4]. 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. The variant employed here differs from that proposed by Farhat in that it works solely with a graph rather than the nodes and elements of a finite element mesh.

Geometric sorting is a simple and intuitive algorithm which 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. However 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.

2.3. Optimisation methods

The two optimisation 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.

The uniform optimisation algorithm is fully described in [6] where it is seen that a key part of the technique is the way in which each sub-domain 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 sub-domain heuristic is to determine the centre of each sub-domain (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. The code finally decides which nodes to migrate based on a combination of radial distance, load-imbalance and the change in cut-edges.

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. After some experimentation it was found that using the radial distance as a basis for migrating nodes which are far from the sub-domain centre was simply not appropriate for achieving a grid mapping, as nodes which are relatively far away from the centre of the sub-domain may be well placed for the topology mapping. To see this, consider a partition for a 1D processor array as in Figure 1(a) where the partition preserving the topology is just a series of strips. Migrating nodes which are far away from the centre of the sub-domain (i.e. at the extremes of each strip) does not preserve the partition as a 1D array. If however, 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 sub-domain, 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 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 [6].

2.4. Mapping strategies

Table 1 describes the four mapping strategies tested. The *unmapped* partition completely ignores the processor topology to give a near optimal partition for a uniform topology as in Figure 1(c). The *postmapped* partition is the unmapped partition remapped to the processors with a processor allocation algorithm applied post-partitioning. This algorithm continually swaps sub-domains 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

Table 1
Mapping strategies

Strategy	Initial partition	Optimisation	Processor allocation
Unmapped	Greedy	Uniform	No
Postmapped	Greedy	Uniform	Yes
Premapped	Geometric sort	Uniform	No
Mapped	Geometric sort	Grid	No

optimised to minimise the number of interprocessor cut-edges. Because the final partition does not deviate far from the initial partition the resulting sub-domain 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. Finally the *partition mapping* strategy acknowledges the processor topology throughout as in Figure 1(b).

3. PARALLEL PHYSICA

Research into multiphysics modelling by the Greenwich group has led to the specification and development of PHYSICA, a modelling software framework for multiphysics phenomena. The core component of PHYSICA is a code structure which provides a three dimensional unstructured mesh framework for the solution of any set of coupled partial differential equations up to second order.

For Finite Volume (FV) procedures the evaluation of fluxes across cell/element faces, volume sources, and coefficients of the linear solvers in the iterative procedures is generic, being essentially based upon mesh geometry and material properties within a cell. As such, the code can be structured so that nodes, FV cell faces and cell volumes can all be calculated automatically and considered as software objects. Since, nodes, cell faces and volumes are all considered as objects the mesh can be conceived of as simply the tool for providing information on the connectivity of nodes, cell faces and volumes; its description may be structured as such in a memory management system which has been designed so that it makes no presuppositions on the geometric structure of the cells. Given that the representation of the mesh connectivity is described by a memory management system, it is straightforward to extend it to include a database system for the storage of all the run time information as well as for model results of any given run. All equation solvers are generic and constructed so that they may be called interchangeably by the user with consistent data structures.

PHYSICA provides a SIMPLE based solution procedure for the fully compressible Navier-Stokes equations with all variables co-located at cell centres using a modified Rhie-Chow method to estimate velocities at cell faces, plus a number of differencing methods to specify the convection terms. A range of turbulence models including k-ϵ, RNG k-ϵ and other length scale techniques together with enthalpy based solidification/melting procedures are coupled with the fluid flow solver. Cell centred elastoviscoplastic solid mechanics with contact analysis are coupled within the false time stepping.

The JOSTLE code is integrated into a PHYSICA prototype to provide at run time a partition of the mesh elements. Face-based and a node-based partitions are derived from the element partition to fully decompose the mesh into sub-domains. Each sub-domain is extended with a surface of elements, faces and nodes overlapping the neighbouring sub-domains. These overlaps carry variables required for the solution of the variables within the sub-domain. Variables in the overlaps are updated from the the processors on which the variable is calculated [5]. A consequence of this sub-domain extension is to increase the sub-domain connectivity. Sub-domains that were not connected in the original partition may become connected through the overlaps. Consider the sub-domain graph in Figure 1(b), here the maximum node degree is four. After applying overlaps to the sub domains the maximum node degree increases to five.

4. RESULTS

The test case used is a solidification problem solving flow, heat and stress over a 60,000 element mesh. This problem was run on the University's Transtech Paramid machine. This machine has 28 i860XP based processing nodes, each of which is equipped with 32 or 16Mbyte of fast DRAM and a T800 communication co-processor. The processor nodes are hard connected in pairs with Inmos C004 multi-stage crossbar switches providing interconnection between the node pairs. This configuration allows great versatility in node interconnection topology. An obvious and simple arrangement for the Paramid topology is a $p{\times}2$ grid which is the arrangement used for these results. A virtual channel router resident on each node allows message passing between all of the nodes on the machine as though the machine were a fully connected network.

In the following two graphs the solid lines refer to partitions reflecting a $p{\times}2$ processor topology and the dashed line indicates a partition reflecting a 1D pipeline or $p{\times}1$ topology.

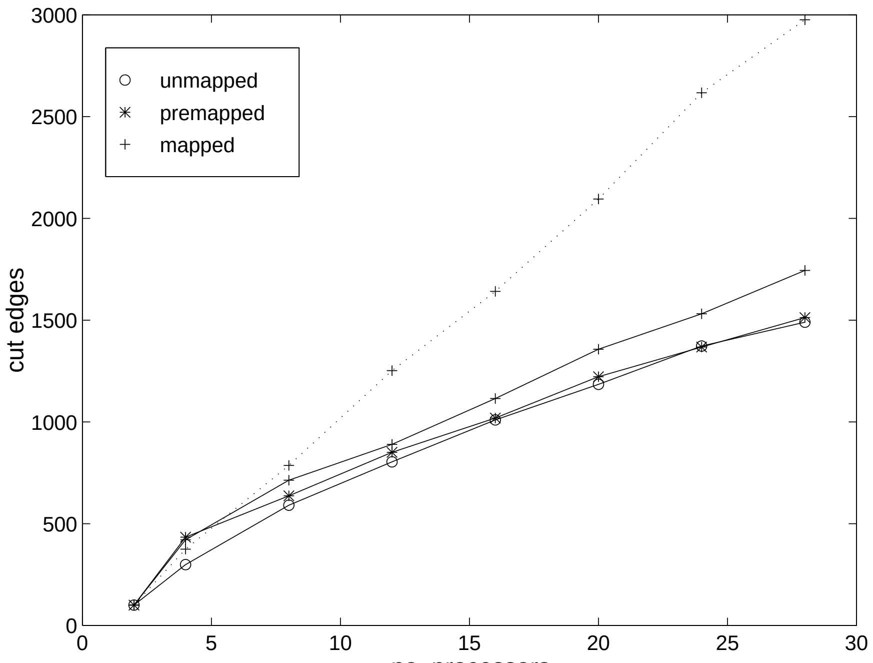

Figure 2. Number of cut edges for a range of partitions.

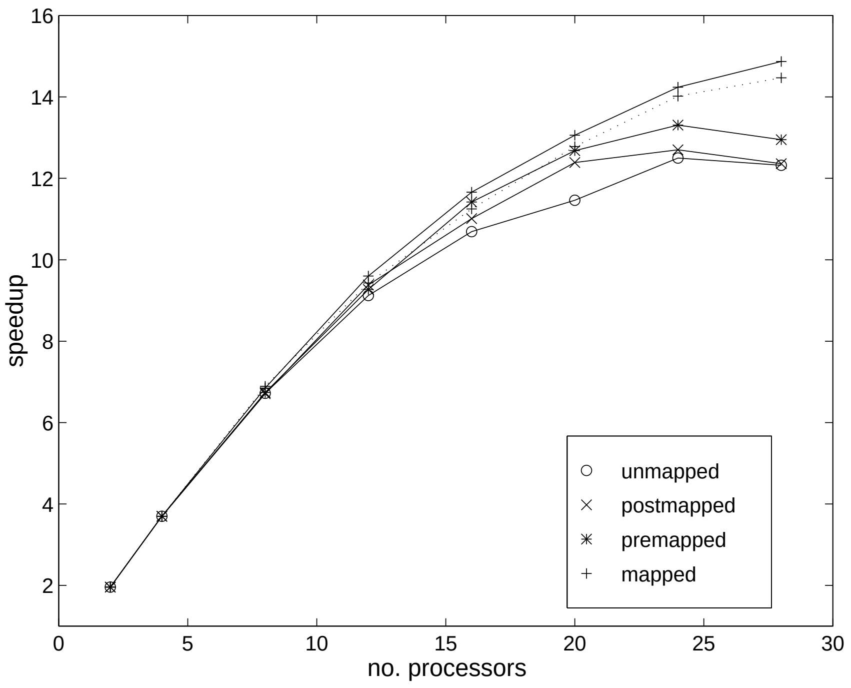

Figure 3. Speedup for a range of partitions on an i860 based Transtech Paramid.

The lowest number of cut edges is given by the unmapped (postmapped) partition but this does not give the best speedup performance. The unmapped and postmapped partitions are actually the same; however the postmapped has in addition an optimised mapping of partitions to processors applied to it. Where the two partitions give a similar speedup this reflects an unintentionally fortuitous mapping of the unmapped partition. It is possible that the unmapped and postmapped partitions may by chance be identical, it is however highly unlikely that the unmapped partition would ever give a better speedup than the postmapped partition. The best overall speedup performance is given by the mapped partitions, despite the cut edge count being higher than the other partitions. This confirms our proposition that partitioning in accordance with the machine topology will result in improved performance. Using a 1D partition leads to a significantly higher number of cut edges and consequently the message length is far greater, however fewer messages are required. In this case only two latencies are required for each overlap update which explains the unexpectedly good speedup results for the pipeline partition. Given that the imbalance of elements between the the sub-domains is less than 0.25% it is apparent from this result that the machine performance with this code is latency bound.

Start-up latency on the Paramid has been measured as 33μs with a peak bandwidth of 1.7Mbytes per second. This bandwidth is not sustained with virtual channel routing and degrades to around 1.3 for near neighbour communication and can get as low as 0.9 for non local messages. This can deteriorate further to around 0.3Mbytes per second if the communication channels are saturated. While this bandwidth is low in comparison with other parallel machines [2] the latency is reasonably good. Similar performance may therefore be expected from other platforms.

Partitioning onto a $p \times q$ processor array where $q > 2$ has yet to be tested, but is not expected to improve performance on the Paramid because of the latency bound. Whilst a $q = 2$ mapped partition is likely to incur five latencies, a $q > 2$ mapped partition can incur eight latencies, but will not significantly reduce the number of cut edges until P increases considerably.

5. MACHINE TOPOLOGY PROFILE

In spite of what parallel machine manufacturers may claim there will always be a distance related communication cost. This cost becomes more significant as the number of processors increases. To quantify the variations in latency and bandwidth we have developed a code which measures the communication performance of a parallel machine. Latency is measured by the simple method of sending a short message between each processor on the parallel machine. Similarly bandwidth is measured by sending a large message between each processor. These measurements are initially carried out with only one message being passed at any one time, and then with every node communicating simultaneously. This provides a peak and a saturated performance measure that may be expressed as a weighted graph (matrix) that describes the communication performance between each pair of processors. What is immediately apparent is the non-uniform performance described by the graph. Such a weighted graph can be obtained quickly, at run time, and then used by the partitioning code to ensure that the mesh partition produced is appropriate for the measured machine communication profile as opposed to a notional topology that may not be reflected in actual communication performance. It is anticipated that this scheme will provide improved performance across a range of parallel machines without the need to understand or specify the architecture of the machine.

REFERENCES

1. S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.
2. J. Dongarra and T. Dunigan. Message passing performance of various computers. Tr, Oak Ridge National Laboratory, University of Tenessee and Oak Ridge National Laboratory, 1995.
3. M. Cross et al. Towards an integrated control volume unstructured mesh code for the simulation of all of the macroscopic processes involved in shape casting. *Num. Meth. Industrial Forming Processes (NUMIFORM 92)*, pages 787–792, 1992.
4. C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Comp. & Struct.*, 28:579–602, 1988.
5. 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. (in press).
6. C. Walshaw, M. Cross, and M. Everett. A Localised Algorithm for Optimising Unstructured Mesh Partitions. *Int. J. Supercomputer Applications*, 1995. (in press).
7. C. Walshaw, M. Cross, S. Johnson, and M. Everett. JOSTLE: Partitioning of Unstructured Meshes for Massively Parallel Machines. In *Parallel CFD'94*. Elsevier, 1995. (in press).