Guide to Partitioning Unstructured Meshes for Parallel Computing

Phil Ridley
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, UK,
email: phil.ridley@nag.co.uk

April 17, 2010

Abstract

Unstrucutured grids are used frequently in finite element or finite volume analysis. Unlike structured grids which are mostly applicable to finite difference schemes, unstructured grids require a list of the connectivity which specifies the way that a given set of vertices form the individual elements. To implement models that use an unstructured numerical decomposition on a distributed memory computer system, careful consideration is required when partitioning the initial grid.

This Computational Science and Engineering (CSE) report discusses the process of partitioning an unstructured grid. First, we outline the algorithms behind graph partitioning and introduce the most commonly used applications for performing this work. Two widely used packages are introduced namely METIS and Scotch, together with how they may be implemented for use on HECToR and this section of the report is intended to be used as a quick start guide. Finally, we compare the efficiency of partitions produced from using these two packages by demonstrating their use in partitioning an unstructured grid for use with the CABARET finite volume code.

Contents

1 Introduction

1.1 Background

Scientific codes which use finite element or finite volume based methods e.g. from general purpose CFD and Structural Mechanics, are most likely to employ unstructured grid methods for their underlying numerical decomposition. The feature which separates the unstructured methods from the structured ones is that we also need to manage the connectivity which specifies the way the grid or mesh is constructed. This is not to say that the underlying algorithms will not scale as efficiently to as many cores as a structured grid based scheme would, but that more effort may be required to achieve this. In terms of computational consideration this involves an additional overhead for the efficiency in data manipulation arising from indirect addressing, non-contiguous memory access and optimisation of inter process communication. For optimum inter process communication an efficient partitioning algorithm is required and this is the main discussion for this report.

Algorithms for finding efficient partitions of highly unstructured graphs are critical for developing solutions for a wide range of problems in many application areas on distributed memory parallel computers. For example, large-scale numerical simulations based on finite element methods will require the finite element mesh to be distributed to the individual processors. The distribution must ensure that the number of elements assigned to each processor is roughly the same, and also that the number of adjacent elements assigned to different processors is minimized. The goal of the first condition is to achieve efficient load balancing for the computations among the processors. The second condition ensures that the communication resulting from the placement of adjacent elements to different processors is minimized. The process of graph partitioning [1] can be used to successfully satisfy these conditions by firstly converting the finite element mesh into a graph, and then partitioning it in an optimal way.

1.2 Structured and Unstructured Grids

All interior vertices of a structured mesh or grid will have an equal number of adjacent elements. Structured meshes typically contain all quadlitateral (2-D) or hexahedral(3-D) elements. Algorithms employed to create such meshes generally involve complex iterative smoothing methods which attempt to align elements with boundaries or physical domains. More importantly, each cell (element) in the grid can be addressed by an index (i, j) in two dimensions or (i, j, k) in three dimensions, and each vertex has coordinates $(i \cdot dx, j \cdot dy)$ in 2-D or $(i \cdot dx, j \cdot dy, k \cdot dz)$ in 3-D for some real numbers dx, dy, and dz which represent the grid spacing.

Unstructured meshes do not have the requirement that all interior vertices will have an equal number of adjacent elements and thus allow any number of elements to meet at a single vertex. Triangle (2-D) and Tetrahedral (3-D) meshes are most commonly thought of when referring to unstructured meshing, although quadrilateral and hexahedral meshes can also be unstructured. Unlike structured meshes or grids each cell (element) cannot be addressed simply by an index, instead a set of piecewise polynomial interpolation functions which are more specifically known as basis functions are employed. In addition we also require a list of the connectivity which specifies the way a given set of vertices make up the individual elements.

There is a large amount of both commericial and freely available software [2] that deals with structured and unstructured meshing and this is more commonly referred to as "grid generation". For the purpose of this report we shall not be concerned with the direct use of these packages. But we still need to be aware of the output that they produce, (i.e. the unstructured finite element grid) since this in itself will be a main source of input to any application that uses an unstructured grid approach.

1.3 Describing an Unstructured Grid

Any application code which uses an unstructured numerical decomposition will require some form of input which describes the mesh or grid topology. This data will always consist of a list of NVERT vertices where each V_i for $i = 1..NVERT$ is a co-ordinate in 2 or 3 dimensional cartesian space. Furthermore how these vertices form the actual grid must also be described and this will be in the form of a cell or element connectivity list. Such a list will contain NCELL cells (or elements) where each cell contains the global reference numbers of N vertices where N is the number of vertices per cell, e.g. for tetrahedral cells N = 4 and for hexahedral cells N = 8. Usually NVERT and NCELL are placed at the beginning of any data file which may be used to store the unstructured grid data and this is so that dynamic arrays can be allocated ready for reading in the data.

In a serial implementation of an unstructured grid based code there will be direct memory access to the entire mesh topology, but for the Single Process Multiple Data (SPMD) technique employed by a distributed memory version of the code, each process will have access to its own local copy of a section of the mesh i.e. a partition. In terms of keeping track of the global effects of the computation halo sections around the partition also need to be implemented. Most importantly from a software development view there are two main considerations, firstly that an efficient method of keeping track of the local to global reference of each of the individual vertices is required. Secondly, an efficient partition of the mesh is required in the first place. The first consideration is to aim for on node (intra-process) optimisation and the second consideration is to acheive optimal load balancing for the problem.

2 Partitioning the Grid for Distributed Computation

2.1 Graph Format for Representation of an Unstructured Grid

Some widely used non commercial partitioning packages are Chaco [3], Jostle [4], METIS [5] and Scotch [6]. These all require the "dual graph" format to be used for representation of the input mesh. However, METIS does have the facility to process an unstrucutured grid from its topological description. This feature is extremely useful for converting a mesh to the dual graph format.

The dual graph format of an unstructured grid may list the connectivity of each vertex or each cell (element), depending upon which option is chosen. Hence, for an unstructured grid we may calculate the dual graph of either the vertices or the elements. From a parallel processing viewpoint we are concerned with the dual graph of the elements (cells) since partitioning on the grid cells is more likely to achieve an optimum load balanced problem rather than partitioning with the vertices.

The dual graph of an unstructured grid based on the grid cells is simply a list of length NCELL where each of the entries lists the neighbouring cells for each of the individual cells $i = 1..NCELL$. Each entry can therefore have a maximum of N neighbours where N is the number of sides of the individual element e.g. for tetrahedral cells N = 4 and for hexahedral cells N = 8. To illustrate the idea of a dual graph with respect to the cells of the original mesh let us consider the trivial triangular finite element mesh in Figure 1. Here there are 3 linear triangular finite elements and a total of 5 vertices which make up the mesh. The corresponding dual graph with respect to the cells (or elements) is shown in Figure 2 or in tabular format as.

3	2	
1	2	3
2	1	
3	1	

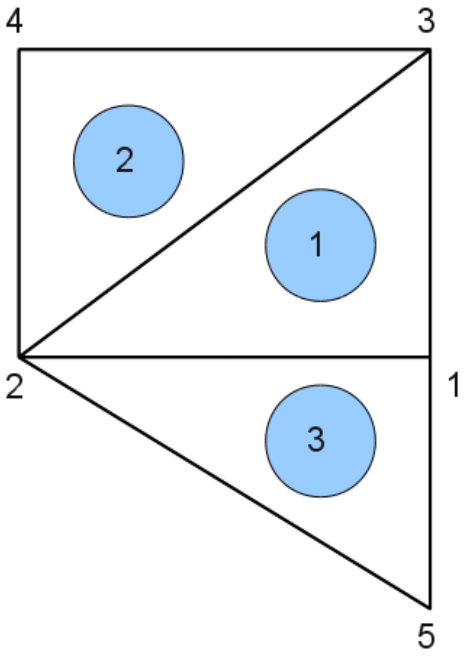

Figure 1: Trivial Triangular Mesh

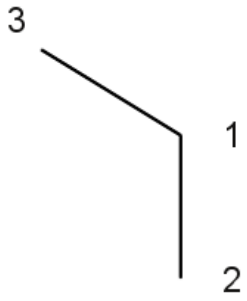

Figure 2: Dual graph for the elements

2.2 Description of the Dual Graph Format

The first line contains the number of elements of the original mesh (NCELL), which is the same as the number of vertices in the dual graph V, following is the number of edges E. The following E lines then correspond to each vertex entry and give a list of each of its neighbors. This format is adopted in Chaco, Jostle and METIS, however, Scotch adopts a slightly different one which will be described later.

2.3 The Graph Partitioning Problem

The common partitioning packages approach the graph partitioning problem in one of two ways, dual recursive bipartitioning methods [7] and multi-level methods [8] and [9]. The multi-level method is in fact a specific case of a more general dual recursive bipartitioning method. Scotch in particular adopts the more general recursive bipartitioning algorithm.

2.4 Algorithms for Finding Partitions

The dual recursive bipartitioning method uses a divide and conquer algorithm to recursively allocate cells (i.e. the V dual graph vertices), to each partition. At each step the algorithm partitions the domain into two disjoint subdomains and calls a bipartitioning routine to efficiently allocate the vertices to these two domains. The specific bipartitioning algorithm optimises the allocation of vertices by using a cost function. There are several options to choose from for the choice of specific algorithm (multi-level is one) and the cost function should be a quantitative representation of the communication for our distributed processing system.

The multi-level or k-way graph partitioning problem (GPP) can be stated as follows: given a graph G(V, E), with vertices V (which can be weighted) and edges (which can also be weighted), partition the vertices into k disjoint sets such that each set contains the same vertex weight and such that the cut-weight, i.e. the total weight of edges cut by the partition, is minimised. The GPP is usually cast as a combinatorial optimisation problem with the cut-weight as the objective function to be minimised and the balancing of vertex weight acting as a constraint. However, it is quite common to slightly relax this constraint in order to improve the partition quality.

3 The METIS Graph Partitioning Application

3.1 Background

The latest version of METIS is 4.0.1 and is available from [5]. The package has been developed as the result of a collaboration between the University of Minnesota, Army High Performance Computing Research Center, Cray Research Inc. and the Pittsburgh Supercomputing Center. Related papers are available at [5] and [10].

METIS is a set of serial programs for partitioning graphs and finite element meshes. The algorithms implemented in METIS are based on the multi-level recursive-bisection, multi-level k-way, and multi-constraint partitioning schemes. The multi-level method reduces the size of the original graph, performs a partition on this and then finally uncoarsens the graph to find a partition for the original graph. METIS can be used as a suite of stand alone partitioning applications or by linking a user's own Fortran or C application to the METIS library.

ParMETIS is an MPI-based parallel library that extends the functionality provided by METIS. It includes routines that are especially suited for large scale parallel numerical simulations and it is able calculate high quality partitions of very large meshes directly, without requiring the application to create the underlying graph. Both METIS and ParMETIS can be used in parallel distrubuted computing applications but for the purpose of this report we shall only consider the use of serial METIS for partitioning, the resulting partition will then be used to generate a parallel decompostion for use by an application which uses an unstructured grid approach to solve a CFD problem.

METIS can be distributed for both commerical and non-commerical use, subject to the conditions mentioned on [5]. Prior permission to distribute or include METIS with an application must also be obtained by sending email to metis@cs.umn.edu.

3.2 Compiling on HECToR

A gzipped tarball of METIS should first be downloaded with wget from [11]. It is probably most convenient to do this from the user's work directory and the package may be extracted with

```
tar -zxvf metis-4.0.tar.gz
```

The object code is built from a Makefile in the $\sim$ `/metis-4.0` directory. Before typing `make` the user should note that the code defines it's own log2 function but this is also defined as

part of the C99 standard. So the -c89 flag must be specified during compilation. Hence, in $\sim$ /metis − 4.0/Makefile.in the user should specify COPTIONS = −c89. The compilation can then be initiated with make
The directory structure of the METIS package is as follows :

- Doc - Contains METIS's user manual
- Graphs - Contains some small sample graphs and meshes that can be used with METIS.
- Lib - Contains the code for METIS's library
- Programs - Contains the code for METIS's stand-alone programs
- Test - Contains a comprehensive tester for METIS's partitioning routines.

However, for this report we shall only be concerned with the executables mesh2dual, kmetis, partdmesh and the METIS library - libmetis.a. These should all be present in the top level metis − 4.0 directory.

3.3 Partitioning an Unstructured Grid

Let us introduce the following unstructured grid as an example to demonstrate the partitioning features of METIS, for this example we wish to generate an efficient partition for 128 distributed meshes of the original grid. If we consider the first few lines of the following unstructured hexahedral mesh file (hexcells.mesh), on the first line at the start of the file we have the number of cells (or finite elements) followed by the element type. The type value can be 1,2,3 or 4 which represents triangular, tetrahedral, hexahedral or quadlitateral elements respectively. The type in the example is hexahedral, hence the rest of this file will contain 100000 lines, with each line consisting of 8 vertex references.

```
100000 3
     321     112     113     650   15961    5363   16133   22812
   15961    5363   16133   22812   15762    5364   16124   27493
   15762    5364   16124   27493   15563    5365   16115   32174
   15563    5365   16115   32174   15364    5366   16106   36855
   15364    5366   16106   36855   15165    5367   16097   41536
   15165    5367   16097   41536   14966    5368   16088   46217 ...
```

The above format is ready for direct use with METIS to calculate a partition with the partdmesh command. However, this application firstly converts the mesh into a dual graph before calculating the partition so it may be more convenient do this first with the mesh2dual application. This is especially useful if we need to produce several partitions of different sizes as it saves generating the dual graph each time.
The METIS instructions mesh2dual and kmetis can be used to produce 128 partitions of the original mesh in hexcells.mesh.dgraph.part.128.

```
>./mesh2dual hexcells.mesh
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: hexcells.mesh, #Elements: 100000, #Nodes: 111741, Etype: HEX

Forming Dual Graph... -----------------------------------------------
  Dual Information: #Vertices: 100000, #Edges: 288600

Timing Information --------------------------------------------------
  I/O:                          0.230
  Dual Creation:                0.060
```

```
**********************************************************************
>./kmetis hexcells.mesh.dgraph 128
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Graph Information ---------------------------------------------------
  Name: hexcells.mesh.dgraph, #Vertices: 100000, #Edges: 288600, #Parts: 128

K-way Partitioning... -----------------------------------------------
  128-way Edge-Cut:   25959, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                          0.050
  Partitioning:                 0.190   (KMETIS time)
  Total:                        0.240
**********************************************************************
```

This could have been done with `partdmesh` and the following produces the output file `hexcells.mesh.epart.128` which is identical to `hexcells.mesh.dgraph.part.128`.

```
> ./partdmesh hexcells.mesh 128
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: hexcells.mesh, #Elements: 100000, #Nodes: 111741, Etype: HEX

Partitioning Dual Graph... ------------------------------------------
  128-way Edge-Cut:   25959, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                          0.160
  Partitioning:                 0.270
**********************************************************************
```

The contents of the file `hexcells.map` will then contain the corresponding partition number from 0 to 127 for the total number of cells (i.e. 100000). In this section we have demonstrated the use of the stand alone METIS applications `mesh2dual`, `kmetis`, `partdmesh`, but it is also possible to link the METIS library to the user's own Fortran (or C) code and use the subroutines `METIS_PartGraphKway`, `METIS_MeshToDual` or `METIS_PartMeshDual` to perform the partitioning automatically.

4 Using Scotch for Unstructured Grid Partitioning

4.1 Background

The latest version of the Scotch package is 5.1.7 and is available for download from [6]. Scotch is developed at the Laboratoire Bordelais de Recherche en Informatique (LaBRI) of the Universit Bordeaux I, and now within the ScAlApplixe project of INRIA Bordeaux Sud-Ouest.

The Dual Recursive Bipartitioning (or DRB) mapping algorithm along with several graph bipartitioning heuristics which are based on a divide and conquer approach, have all been implemented in the Scotch software package. Recently, the ordering capabilities of Scotch have been extended to native mesh structures by the application of hypergraph partitioning algorithms. The parallel features of Scotch are referred to as PT-Scotch (Parallel Threaded

Scotch). Both packages share a signicant amount of code and PT-Scotch transfers control to the sequential routines of the Scotch library when the subgraphs on which it operates are located on a single processor. Both Scotch and PT-Scotch are extremely useful for partitioning grids for use in parallel distrubuted computing applications. For the purpose of this report we shall only consider the use of serial Scotch for partitioning, the resulting partition will then be used to generate a parallel decompostion for use in the example CFD application code.

Scotch is available under a dual licensing basis. It is downloadable from the Scotch web page as free software to all interested parties willing to use it as a library or to contribute to it as a testbed for new partitioning and ordering methods. It can also be distributed under other types of licenses and conditions, e.g. to parties that wish to embed it tightly into proprietary software. The free software license under which Scotch 5.1 is distributed is the CeCILL-C license [12], which has basically the same features as the GNU LGPL (Lesser General Public License): ability to link the code as a library to any free/libre or even proprietary software, ability to modify the code and to redistribute these modications. Version 4.0 of Scotch was distributed under the LGPL itself.

4.2 Compiling on HECToR

A gzipped tarball of Scotch should first be downloaded with wget from [13]. It is most probably convenient to do this from the user's work directory and the package is extracted with

```
tar -zxvf scotch_5.1.7.tar.gz
```

The object code is built from a Makefile which requires the input file Makefile.inc to set the machine specific compile options. The user should then

```
cd ~/scotch_5.1/src/
```

There is already a suitable Makefile.inc for a CrayXT which should be copied into this file before compiling Scotch as follows.

```
cp Make.inc/Makefile.inc.x86-64\_cray-xt4\_linux2} ./Makefile.inc
make
```

Check that the executables have been compiled with

```
ls ~/scotch_5.1/bin/
```

If the directory listing shows the following then Scotch has been compiled successfully.

```
acpl     amk_fft2  amk_hy  amk_p2  gbase  gmap    gmk_m2  gmk_msh  gmtst ...
```

4.3 Partitioning an Unstructured Grid

We shall be using Scotch to produce a partition for the same unstructured grid we considered in the METIS example. Scotch does not have the facility to read in a user's mesh directly and therefore the first step will be to convert a mesh from the following format into the dual graph form which Scotch requires. If we consider the first few lines of the following unstructured hexahedral mesh file (`hexcells.mesh`), at the start of the file we have the number of cells (or finite elements) followed by the element type on the first line. The type value can be 1,2,3 or 4 which represents triangular, tetrahedral, hexahedral or quadlitateral elements respectively. The type in this example is hexahedral, hence the rest of this file will contain 100000 lines, with each line consisting of 8 vertex references.

```
100000 3
     321      112      113      650    15961     5363    16133    22812
   15961     5363    16133    22812    15762     5364    16124    27493
```

```
   15762    5364   16124   27493   15563    5365   16115   32174
   15563    5365   16115   32174   15364    5366   16106   36855
   15364    5366   16106   36855   15165    5367   16097   41536
   15165    5367   16097   41536   14966    5368   16088   46217 ...
```

Before we can use Scotch to partition this mesh we need to convert it into the dual graph format. This can be conveniently performed by the use of the METIS application `mesh2dual`. From the appropriate location for the METIS executables one should issue the following command

```
mesh2dual hexcells.mesh
```

This will produce an output file `hexcells.mesh.dgraph`

```
100000 288600
 21 4001 2
 1 22 4002 3
 2 23 4003 4
 3 24 4004 5
 4 25 4005 6
 5 26 4006 7
 6 27 4007 8
 7 28 4008 9
 8 29 4009 10
 9 30 4010 11 ...
```

However, this is not quite what we require for Scotch. So we need to re-compile METIS and `mesh2dual` so that it will output the number of each neighbouring cell at the beginning of every line. This is trivial and the necessary file to alter is metis-4.0/Programs/io.c in the function `WriteGraph`. The following code snippet shows the modified version which also sets the element numbering to 0 rather than 1.

```
void WriteGraph(char *filename, int nvtxs, idxtype *xadj, idxtype *adjncy)
{
  int i, j;
  FILE *fpout;

  if ((fpout = fopen(filename, "w")) == NULL) {
    printf("Failed to open file %s\n", filename);
    exit(0);
  }

/*  fprintf(fpout, "%d %d", nvtxs, xadj[nvtxs]/2);*/
  fprintf(fpout, "%d %d", nvtxs, xadj[nvtxs]);
  for (i=0; i<nvtxs; i++) {
    fprintf(fpout, "\n");
    fprintf(fpout, "%d  ",xadj[i+1]-xadj[i]);
    for (j=xadj[i]; j<xadj[i+1]; j++)
      fprintf(fpout, "%d  ", adjncy[j]);
/*        fprintf(fpout, " %d", adjncy[j]+1); */
  }

  fclose(fpout);
}
```

After modifying metis-4.0/Programs/io.c to the above one should re-compile METIS and repeat

```
mesh2dual hexcells.mesh
```

The contents of the output file `hexcells.mesh.dgraph` should then be

```
100000 577200
3  20  4000  1
4  0  21  4001  2
4  1  22  4002  3
4  2  23  4003  4
4  3  24  4004  5
4  4  25  4005  6
4  5  26  4006  7
4  6  27  4007  8
4  7  28  4008  9
4  8  29  4009  10 ...
```

This is still not quite ready for Scotch since we just need to do the following minor ammendment. Firstly, one should copy or rename this file to `hexcells.grf`. Then edit this file so that the first line is altered as follows.

```
0
100000 577200
0      000
3  20  4000  1
4  0  21  4001  2
4  1  22  4002  3...
```

The file `hexcells.grf` is now ready for processing with Scotch. To check the consistency of this Scotch .grf file. The executable gtst can be used as follows

```
> bin/gtst grf/hexcells.grf
S       Vertex  nbr=100000
S       Vertex load     min=1   max=1   sum=100000      avg=1   dlt=0
S       Vertex degree   min=3   max=6   sum=577200      avg=5.772  dlt=0.358279
S       Edge    nbr=288600
S       Edge load       min=1   max=1   sum=288600      avg=1   dlt=0
```

To achieve our goal in obtaining a partition of the original hexahedral mesh, the following will produce 128 partitions for the mesh in the output file `hexcells.map`. This is similar to the METIS output in `hexcells.mesh.epart.128` and `hexcells.mesh.dgraph.part.128`.

```
> echo cmplt 128 | bin/gmap grf/hexcells.grf - grf/hexcells.map
```

The contents of the file `hexcells.map` will then contain the total number of cells (i.e. 100000) followed by the cell number and its corresponding partition number from 0 to 127. In this section we have demonstrated use of the stand alone Scotch applications `gtst` and `gmap` it is also possible to link the scotch library to the user's own Fortran (or C) code and use the subroutines `SCOTCH_graphCheck` or `SCOTCH_graphMap`

5 Comparison of the Efficiency of METIS and Scotch

5.1 Outline

In this section we shall compare the practical use of METIS and Scotch when applied to an unstructured CFD application, namely the CABARET Fortran 90/MPI code. We shall compare the performance of the CABARET code when using partitioned meshes produced via METIS, Scotch and also from the natural(i.e. contiguous) finite element ordering. In the remainder of this section a summary of these results will then be presented.

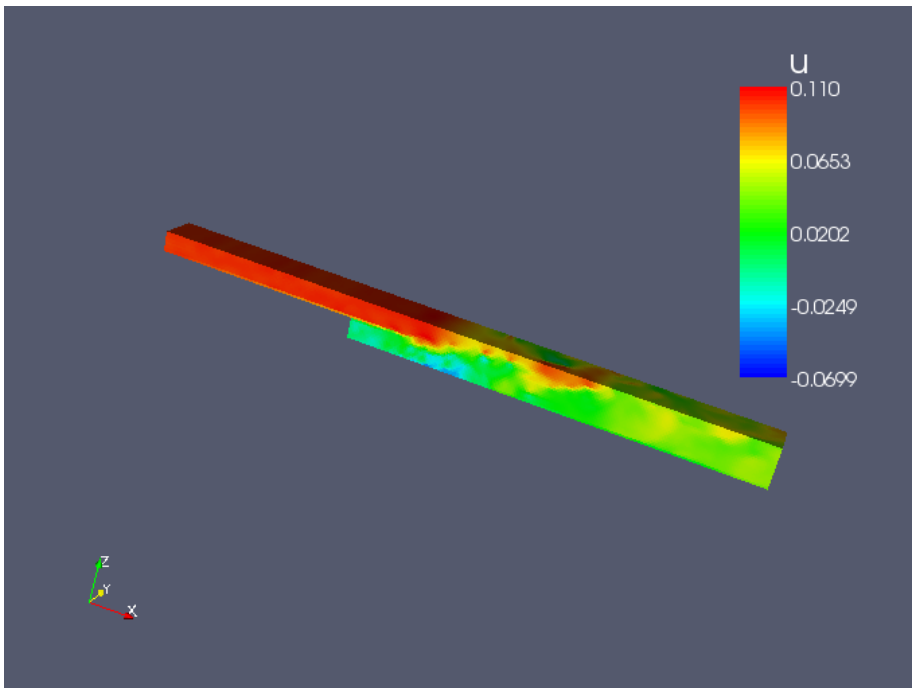

Figure 3: x direction velocity component of the flow field at 40000 time steps

5.2 CABARET

To accurately resolve turbulent flow structures high-fidelity CFD simulations require the use of millions of grid points. The Compact Accurately Boundary Adjusting high-Resolution Technique (CABARET) is capable of producing accurate results with at least 10 times more efficiency than conventional high resolution schemes. CABARET [14] is based on a local second-order finite difference scheme which lends itself extremely well to distributed memory parallel computer systems. For Reynolds numbers of 10^4 and Mach numbers as low as 0.05 the CABARET method gives rapid convergence without requiring additional preconditioning.

In this section we shall discuss the performance of the CABARET unstructured hexahedral code on HECToR by considering different partitioning methods. Performance of the code for up to 256 processing cores will be discussed in relation to the effectiveness of the load balancing for grids generated from the METIS and Scotch partitioning applications.

The CABARET code involves a 3 phase computation which currently uses pure MPI for the communications, for each iteration of the scheme there are 3 nearest neighbour type communications and a global `all_reduce`. The computational grid is constructed with 8 vertex (i.e. linear) hexahedral finite elements. The parallel decomposition is required to be unstructured to facilitate future irregularity in the discretisation scheme.

For our test case demonstration we shall consider the turbulent flow around a 3-D backward facing step using a fixed grid of 100000 finite elements. Results will be given for 276 iterations of the numerical scheme and timings will not include any I/O.

To demonstrate the method, Figure 3 shows the normalised x direction velocity component of the flow field at 40000 time steps and Figure 4 demonstrates the accuracy of the method by showing the correctly resolved streamlines around the recirculation zone after 100000 time steps.

To compare the effectiveness of the partitions produced by METIS and Scotch, the following test was carried out on 32, 64, 128 and 256 cores using the quad core processing nodes of HECToR phase 2a. A fixed grid of 100000 hexahedral finite elements was partitioned and distributed meshes were produced which were suitable to use with the parallel CABARET code. We have used METIS, Scotch and contiguous partitioning for the tests. The METIS and Scotch partitioning was performed by using both these packages with the methods described earlier in this report. The contiguous partitioning method was performed by simply dividing the grid

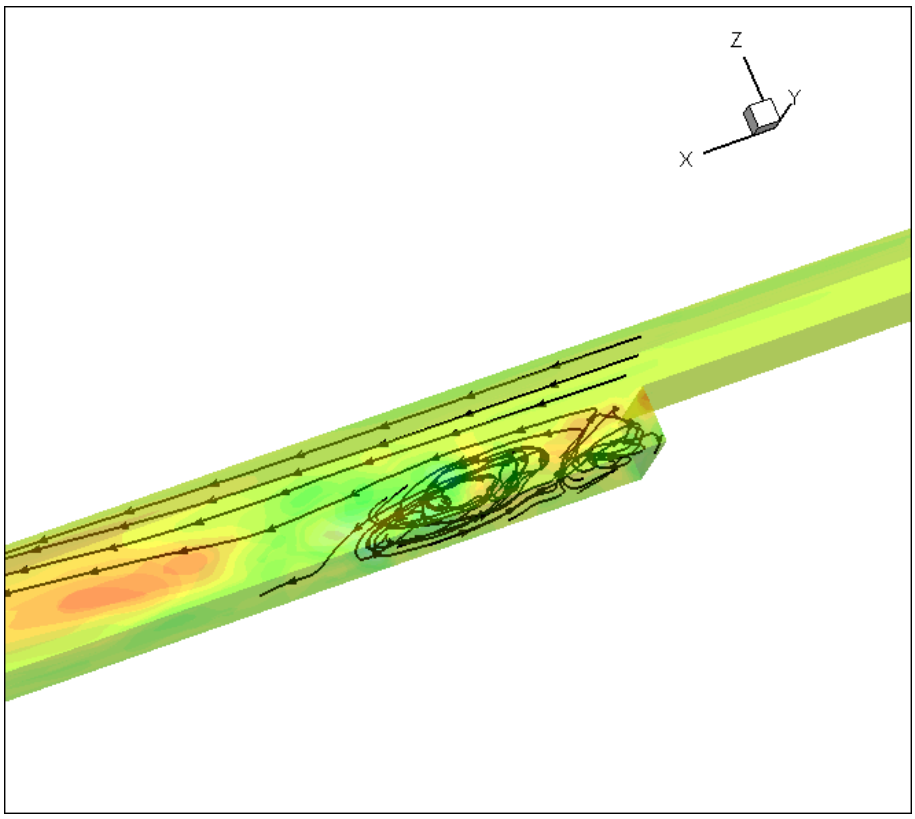

Figure 4: Streamlines around the recirculation zone after 100000 time steps

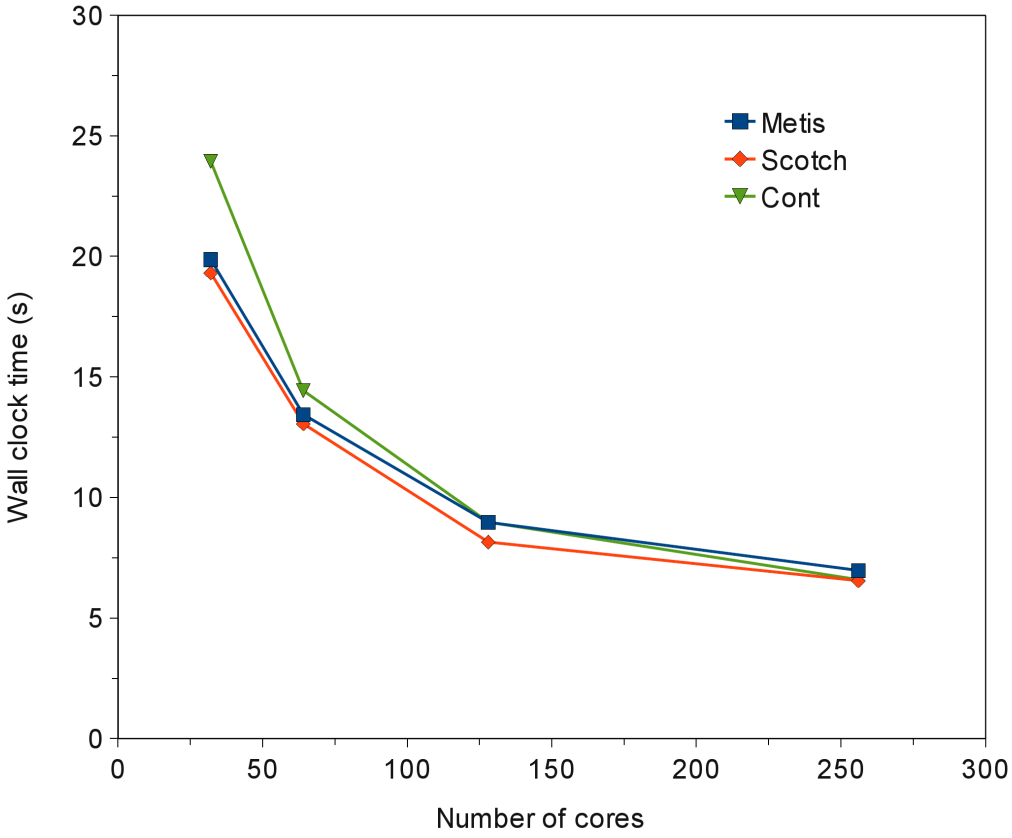

Figure 5: Timings for 256 time steps of CABARET

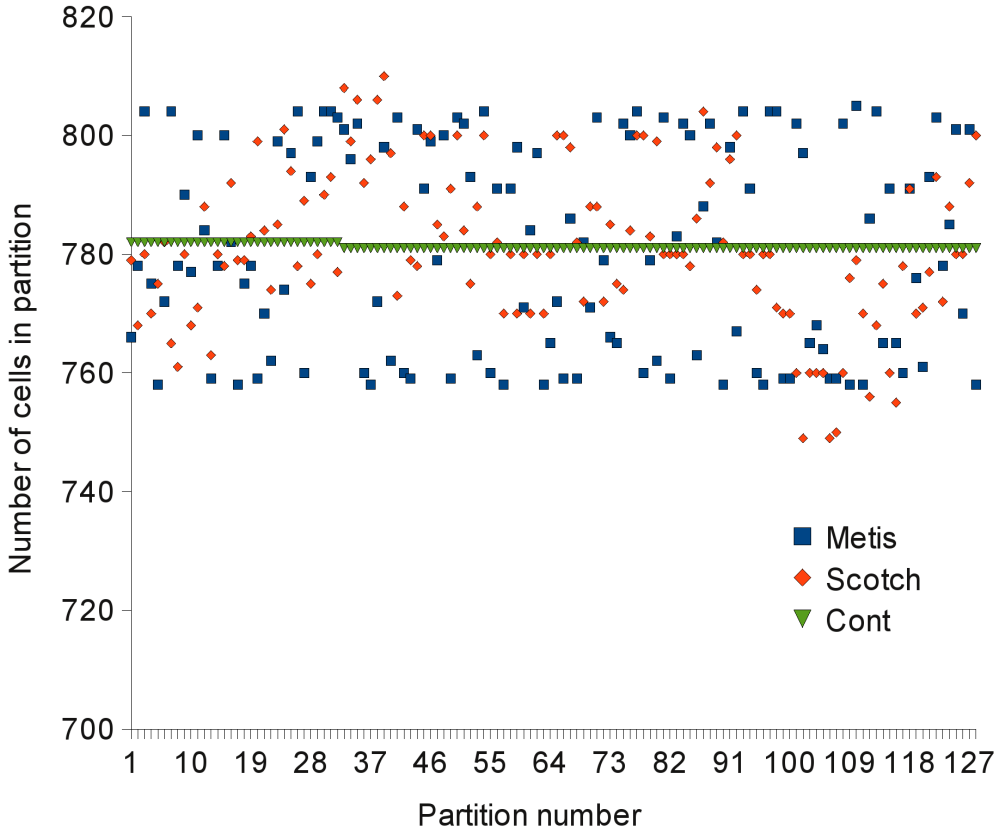

Figure 6: Number of cells in each of 128 partitions

size, i.e. 100000 by the number of cores in question p, so the first $1, 2, .., \frac{100000}{p}$ cells get assigned to process 1, the next $\left(\frac{100000}{p} + 1\right), .., 2\left(\frac{100000}{p}\right)$ to process 2, etc and then any remainder (e.g. r) is added to the first $r-1$ processes. This method does not take into account any optimisation factor for the communications and relies entirely upon the efficiency of the element ordering from the initial mesh generation. The contiguous partitioning method produces consecutively numbered elements within each local process and this approach is not likely to result in very good performance since elements that belong to different geometrical regions of the grid may get partitioned to the same process.

The timings for 256 time steps of the CABARET code using the different methods for partitioning are shown in Figure 5. Each line shows results for METIS, Scotch and contiguous partitioning for 32, 64, 128 and 256 cores. In all cases Scotch produces slightly better performance which is around a 3% reduction in wall clock time compared with METIS. However, for 128 cores there is a noticeable increase to nearly 10%. Also for this number of partitions it is noticeable that the contiguous method gives similar performance to METIS.

To give an idea of the load balancing for the 128 core case Figure 6 shows the number of cells placed in each of the 128 partitions from each of the 3 methods. Another influencing factor with the CABARET application is the number of cells in each partition which contain boundary faces. If there is an unfair distribution of the boundaries within the partitions then certain partitions may have extra communications to cope with.

For this particular example, the problem size is fixed at 100000 cells and increasing the number of cores to run the problem over will increase the communications overhead, at 32 cores the total time spent in communications is around 54% which rises to 78% for 256 cores. So for 256 cores all 3 cases are dominated by communication and there is less benefit to be gained by efficient partitioning. However, for the smaller numbers of partitions the efficiency of the particular partitioner is evidently more important.

These results demonstrate that Scotch does give slightly better performance when used to produce a partitioning for an unstructured hexahedral mesh for use with CABARET. METIS is slightly easier to use since it does allow the user to provide it with input which is close to that of the original unstructured grid whereas Scotch requires the input to be in the form of a dual graph. However, this could easily be incorporated within a pre-processing stage which could be performed prior to the partitioning phase.

6 Conclusion

In this report we have introduced the idea of an unstructured grid. Application codes using such a grid type will need an efficient partitioning method for the grid itself if the code is to be ported to a distributed memory parallel computer. We have summarised the use of the METIS and Scotch partitioning packages which may be used to perform this operation and have demonstrated their use on HECToR phase 2a. This has been achieved by comparing their use to produce partitions for a fixed sized unstructured hexahedral grid for use by the CABARET CFD code.

For a representative test case results have shown that the partitions produced by using Scotch give slightly better performance over those produced from METIS. Although, METIS is generally easier to use than Scotch for this particular application.

References

[1] "An efficient heuristic procedure for partitioning graphs", B. W. Kernighan and S. Lin, Bell Systems Technical Journal 49 (1970), pp291-307. Also available at http://www.cs.princeton.edu/∼bwk/btl.mirror/new/partitioning.pdf

[2] http://www-users.informatik.rwth-aachen.de/∼roberts/software.html

[3] http://www.cs.sandia.gov/CRF/chac.html

[4] http://www.gre.ac.uk/∼c.walshaw/jostle/

[5] http://www-users.cs.umn.edu/∼karypis/metis/

[6] http://www.labri.fr/Perso/∼pelegrin/scotch/

[7] "Static mapping by dual recursive bipartitioning of process and architecture graphs", F. Pelegrini, Proceedings of the IEEE Scalable High-Performance Computing Conference (1994), pp486-493.

[8] "A multilevel algorithm for partitioning graphs", B. Hendrickson and R. Leland, Proceedings of the IEEE/ACM SC95 Conference(1995), pp28-28. Also available at http://www.sandia.gov/ bahendr/papers/multilevel.ps

[9] "Multilevel k-way partitioning scheme for irregular graphs", G. Karypis and V. Kumar, Journal of Parallel and Distributed Computing Vol.48 (1998), pp96-129. Also available at http://www.cs.umn.edu/∼karypis

[10] "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs", G. Karypis and V. Kumar, SIAM Journal on Scientific Computing Vol. 20 (1999), pp359-392.

[11] http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/metis-4.0.tar.gz

[12] http://www.cecill.info/licenses.en.html

[13] http://gforge.inria.fr/frs/download.php/23390/scotch_5.1.7.tar.gz

[14] "A New Efficient High-Resolution Method for Nonlinear Problems in Aeroacoustics", S.A. Karabasov and V.M. Goloviznin, American Institute of Aeronautics and Astronautics Journal Vol. 45 (2007), pp2861-2871.