RALPAR - RAL Mesh Partitioning Program
Version 2.0

RF Fowler and C Greenough

December 1997

Abstract

This report describes the second release of the Rutherford Appleton Laboratory Mesh Partitioning Program *ralpar*. *Ralpar* is a software tool to split up unstructured meshes into subdomains for parallel processing on MIMD type architectures. A wide range of basic decomposition methods have been implemented within this tool. This document briefly describes these methods and how to use *ralpar* to partition meshes. It also gives details of the measures of partition quality that are provided.

A copy of this report can be found at the Department's web site (*http://www.dci.clrc.ac.uk/*) under page *Group.asp?DCICSEMSW* or anonymous ftp server *www.inf.rl.ac.uk* under the directory *pub/mathsoft/publications*

Mathematical Software Group
Department for Computation and Information
Rutherford Appleton Laboratory
Chilton, DIDCOT
Oxfordshire OX11 0QX

Contents

1 Introduction

This document describes Version 2.0 of the *ralpar* mesh partitioning tool. This tool implements a wide range of basic mesh partitioning algorithms for unstructured finite element type meshes in two and three dimensions using direct and graph based methods. The tool provides some simple display options and commands for storing information about the partitions generated.

Mesh partitioning for unstructured meshes is important in the area of parallel processing for grid based calculations. *ralpar* is a tool to help in the generation of these partitions. The initial sections of this report review the requirements of mesh partitioning for parallel and distributed computing. A short description of the partitioning methods contained in *ralpar* is included together with some thoughts on measuring the quality of the partitions generated.

Much of the report is given over to the detailed use of *ralpar* which is illustrated using some two and three-dimensional examples.

Appendix A is a command reference Section for *ralpar,*. The interface to RALBIC neutral files and memory management control are discussed in Appendices B and C respectively.

2 Parallel processing and mesh decomposition

The use of computing systems with some type of parallel architecture has grown significantly over the past few years. These systems are seen as the path by which sufficient computing power can be provided for accurate three dimensional simulations of complex phenomenon in areas such as fluid dynamics, stress analysis and electromagnetism. Such simulations typically use meshes or grids containing 10^5 to 10^6 nodes and may employ automatic grid refinement during the solution. Unstructured grids are often necessary due to complex geometries and the need to place nodes in an intelligent manner, to minimise the overall error in the solution.

Many of the parallel machines now being used are based on the MIMD form of parallelism where the memory of the machine is distributed over a network of processors. A consequence of this is that the program and its associated data must also be distributed between these processors. In finite volume and finite element methods this leads to the problem of how to distribute large unstructured grids and meshes initially, and how to redistribute them subsequently, if refinement is made. This tool only addresses the problem of the initial partitioning and assumes that a single processor with sufficient memory is available to partition the complete mesh.

When moving a large grid based computation to a distributed memory machine there are two main approaches:

- Direct parallelisation of the existing solution algorithm. In this case data exchanges are made between processors whenever required by the algorithm.

- True Domain Decomposition (DD) methods. In this case separate problems are solved on each of the subdomains independently and then some method is used to “patch” together these solutions, which again involves interprocessor communication. Two subclasses within DD methods are:

  1. Overlapping subdomains. In this case a layer of mesh elements on the interface between two subdomains is assigned to both of them. These allow methods based on the Schwarz alternating procedure [1] to be used.

2. Non-overlapping subdomains. In this case it is necessary to solve a separate global problem for the interface nodes between domains. One approach is to form the Schur complement matrix for the interface.

The current version of *ralpar* only produces non-overlapping mesh decompositions, though the `HALO` command allows the identification of elements in an overlap region.

Three of the major requirements for efficient parallel processing are:

1. To minimise the sequential part of the computation.

2. To ensure that all processors have the same amount of work to do.

3. To minimise interprocessor communication.

The fact that *ralpar* is a sequential program is not optimal with regard to the first point, but the cost of partitioning is often quite small compared to the actual computations performed on the mesh.

Load balancing is often achievable by assigning the same number of elements to each processor. However in many cases, such as with a mixed mesh containing several different element types, it can be desirable to attach a computational weight w_i to each element. Partitioning of the mesh into N parts should then aim to assign each domain a set of elements with weight W given by

$$W = \frac{\sum_i w_i}{N} \tag{1}$$

The current version of the tool only deals with element partitioning, though there can be cases where a partitioning of the nodes is more appropriate.

Another extension that is becoming important for computations on heterogeneous workstation clusters is that of domain weighting. It may be that one machine is more powerful than the others and hence should have a larger share of the work. This can also be included in the above scheme and is available in the current version of *ralpar*.

Minimisation of interprocessor communication time is very important in distributed grid based calculations. An ideal case would be where the subdomains could be completely decoupled and no exchange of information was required. This is rarely the case, so the aim of a mesh partitioning method is to find a splitting that minimises the communication cost. The NP-hard nature of this minimisation means that all methods are heuristic, and may not find the global minimum. The actual communication time will also depend on the characteristics of the parallel machine, the solution algorithm and the nature of the problem being solved.

3 Measuring partition quality

The true measure of interest for any mesh partitioning method is the effect it has on the run time for the computations that are to be performed on a real parallel system. Generally, this effect is too sensitive to the particular combination of algorithm and parallel hardware being used to be practical. Instead simpler models are used which measure the size of the interface between subdomains and related quantities.

Load balancing, by assigning the same amount of work to each processor, is built into all the decomposition schemes discussed here. Hence it is not a good measure to compare decomposition methods with. Multi-level methods which do not include some form of smoothing on the first level can produce poorer load balance than expected. As a result in most implementations of multi-level techniques smoothing in provided between expansions.

As was noted in the previous section, the number of interface nodes that a mesh decomposition produces can effect the size of the interface problem to be solved. This quantity is taken as our main measure of interface size. For all methods *ralpar* reports the total number of interface nodes that are generated.

Another measure that may be of importance is the number of neighbouring subdomains about each subdomain. This can be important in matrix assembly operations for node based quantities where each processor will have to exchange interface data with all its neighbours. While the total amount of data to be exchanged will be proportional to the number of interface nodes, there is always a certain amount of time involved in initiating a communication with another processor. Under some circumstances this time can be a significant amount of the total communication time. In *ralpar* the user is provided with the average number of neighbouring subdomains for the partition along with the minimum and maximum number of neighbours for any one subdomain.

Many partitioning methods can be cast in terms of an undirected graph which describes the way in which elements are connected together. Within such a graph, each element is represented as a vertex of the graph. The need for communication between neighbouring elements is then indicated by an "edge" of the graph, joining the two vertices. This approach is more fully explained in [11]. For such methods a convenient measure of the quality of a partition is the number of *cut edges* generated. A cut edge occurs when the vertices (elements) at either end of the edge are assigned to different subdomains. *Ralpar* calculates the sum of these cut edges for all graph based methods. This measure is not identical to the number of interface nodes, but they are roughly proportional to each other. The number of edges in a graph, and hence the number of cut edges, depends on the definition used for elements to be connected. The three options are available within *ralpar* are:

> *EDDG* - elements are linked if they have a common face with a neighbour *TRUE* - elements are linked if they share one or more nodes with a neighbour *WEIGHT* - elements are linked if they share one or more nodes with a neighbour and the edge is weighted by the total number of common nodes.

The number of cut edges reported will depend on which definition has been used. For the first two options, all edges are assigned a weight of one.

4 Mesh partitioning techniques

All mesh partitioning methods generate a parameter or measure to separate elements into partitions. These *separators* can be based on some geometric property of the mesh or based on some attribute of the graph associated with the topology of the mesh. Some methods have more heuristic origins and base the separation on some type of cost measure. *ralpar* is a pre-processor of finite element meshes prior to their use in parallel or distributed computation. It provides the analysis program with a list of elements along with the partition number that they have been assigned to. The following mesh partitioning techniques are currently available within *ralpar*:

Geometric Methods

Geometric bisection (`geo-bis`):

Cost Optimised geometric bisection (`costgeo`)

Geometric bisection on axes of inertia (`inertia`)

Recursive version on Inertia method (`r-inertia`)

Graph Based Methods

Graph bisection method (`graph`)

Malone bandwidth minimisation algorithm (`bandwdt`)

Variation of Malone method with profile minimisation (`profile`)

Greedy algorithm of Farhat (`greedy`)

Cost optimised version of Greedy algorithm (`glutton`)

Kernighan and Lin method with a Greedy start (`kl-greedy`)

Kernighan and Lin method with a random start (`kl-rand`)

Kernighan and Lin with a graph bisection start (`kl-rgb`)

Spectral bisection (may use KL refinement) (`spect`)

Multi-Level Methods

A more detailed descriptions of these methods is given in [11].

5 Multilevel partitioning methods

Some partitioning methods can be very expensive in terms of CPU time and memory requirements for large problems. The spectral method is a prime example of this as the eigenvector solution can be very expensive when dealing with the order of a million elements. Multilevel methods [12] have been proposed as way of reducing the costs of such techniques while still generating high quality partitions. These work on the connectivity graph of the mesh, but instead of trying to split this directly, the graph is first “condensed” through a number of levels. The condensation is achieved through clustering together vertices that are “close together” to produce a graph with fewer vertices. New edges between the clusters are weighted to reflect the number of edges that existed in the larger graph.

By using several levels of condensation a much smaller graph can be obtained that is easily partitioned by a method such as spectral bisection. This partitioning information can then be transfered up through the levels to the original graph. In practice it is found that some refinement of the partition is required on the higher graph levels, and this can be done using the Kernighan and Lin (KL) technique.

The multilevel methods can be accessed using the `MLPART` command. This gives a number of options as to the number of levels of condensation to be used and how vertices are to be clustered at each level. The set of methods that can be used to partition the smallest graph is as follows:

Graph bisection (`graph`)

Dual graph variation of graph bisection (`dgraph`)

Malone bandwidth minimisation algorithm (on vertices) (`malone`)

Greedy algorithm of Farhat (`greedy`)

Cost optimised version of Greedy algorithm (`glutton`)

Random partitioning (`rand`)

Spectral method (`spec`)

All these methods can be combined with KL refinement at just the lowest graph level or at all levels. In addition the methods can be used in three different ways:

Recursive bisection: splitting the graph in half each time (`bisect`)

Recursive section: splitting one subdomain at a time from the remaining graph (`recsect`)

Linear section: where the method is applied just once to get a single ordering of the vertices. This is then split into the required number of subdomains. (`linsect`)

In the current release KL refinement can not be used with linear sections.

The `mlpart` command allows an additional parameters to control the KL refinement process.

6 Modelling parallel system performance

For a given algorithm and data distribution it is often possible to specify how much data needs to be exchanged and between which processors this has to occur. The time to send n bytes between two processors in a parallel machine can often be approximated by the equation

$$t = t_{start} + nt_{send} \tag{2}$$

where t_{start} is the time to initialise communications and t_{send} is the time to send one byte. This formula ignores many factors, such as contention (due to other messages in the system), multihop costs, message length dependent buffering strategies and so on. However it is a reasonable first approximation and the parameters have been measured for most parallel systems in common use.

If the knowledge of the algorithm communication requirements is combined with the data distribution of a given partition and a communication cost model, like (2), then it is possible to estimate the actual communication time.

As a first step in this direction, two simple cost modelling schemes have been built into *ralpar*. There are two commands associated with these:

- `Machine` - this command allows machine communication parameters to be stored and viewed. The parameters used are the t_{start} and t_{send} values defined above, expressed in units of microseconds. R_∞ and $N_{1/2}$, the maximum communication rate and half performance message length respectively, are calculated from t_{start} and t_{send}. The `machine` command also selects which machine type is to be used by the next `table` calculation.

- `Table` - this command calculates or displays the results for a given mesh using a range of partitioning methods. Results can also be written to file using this command.

The cost models currently implemented assume a simple contribution assembly calculation where each processor will have to transmit or receive data proportional to the number of interface nodes that it has. We assume that one 8 byte value will be associated with every interface node and must be swapped with a neighbouring subdomain. For the first model we assume communications on a bus type architecture, such as a network of workstations connected via Ethernet. This is referred to as SEQCOMM as in this case only one message can be active at a time and the total time spent in communication is just the sum of all the communication times for individual subdomains. Thus we calculate the communication time T_{comm} as,

$$\begin{aligned} T_{comm} &= \sum_{i=1}^{p}(N_i t_{start} + 8 t_{send} I_i) \\ &= N_T t_{start} + 8 t_{send} \sum_{i=1}^{p} I_i \qquad (3) \end{aligned}$$

where I_i is the number of interface nodes associated with subdomain i and N_i is the number of neighbour subdomains for i. N_T is the total number of neighbours, given by summing N_i. Clearly, there are a number of variations one could make in this formula, depending on the number of variables to be evaluated and which processors need to know the result. However, (3) is one possible form and we can make comparisons between methods using it, as long as we accept the results are qualitative rather than quantitative.

The second model that has been implemented is referred to as PARACOMM. This model assumes that all processors can perform their communication in parallel. It is again assumed that each processor will have to send or receive 8 bytes of data for each interface node that belongs to its domain and also to start communication with as many other processors as it has neighbours. Hence the total communication time for one processor is just

$$T_{comm}(i) = 8 t_{send} I_i + N_i t_{start} \qquad (4)$$

Thus communication tasks within a single processor will be done sequentially, but each processor will do this in parallel with all the others. On some systems, such as the Intel iPSC/860, it is possible to have concurrent communication on one processor using all the links available at once, though this may result in increased network contention. Such contention is ignored in the PARACOMM model. Note that in this model we use the I_i to represent *all* the interface nodes on subdomain i, where as in the sequential model interface nodes were assigned to the first subdomain they appeared in. Hence we have a larger amount of data interchange in this case.

Assuming that all the processors can operate in parallel without any contention, then the total communication time will be given by the largest value of $T_{comm}(i)$. The PARACOMM model reports both the maximum and average values of $T_{comm}(i)$. Details of how the `Machine` and `Table` commands can be found in the Command Appenix.

7 Using *ralpar*

7.1 Starting execution

To start the standard version of *ralpar*, you need to be in a directory that contains the command definition file `ralpar.cmd`. If this file is not present, *ralpar* will stop and issue an error message.

The partitioner is started by issuing the executable name, if it has been placed in your search path, or the full pathname otherwise, e.g. `../ralpar`. This will cause a short introductory message to be printed followed by the prompt `Ralpar:`. Any valid command can than be issued and these commands are fully described in Appendix A.

7.2 *Ralpar* Commands

Ralpar has a comprehensive set of commands to control partition creation, method selection and information output. To get an online list of command names you just type `help`. The commands currently defined are:

```
Applications Commands

 INPUT        -  to read in mesh file
 PARTITION    -  to partition the data
 DISPLAY      -  to display partition results
 PLIST        -  to list element numbers in a partition
 OUTPUT       -  to write output file with partition numbers
 WEIGHT       -  Define element weight
 MACHINE      -  to specify machine constants
 TABLE        -  to evaluate or display table of costs
 INFORMATION -  control message output
 QUIT         -  to quit program
 VIEW         -  to change viewing angle for current plot
 HALO         -  to construct element halo regions
 LOADPAR      -  to read in partition data
 MLPART       -  multilevel partitioning

Internal Commands

 CHANGE       -  to change working directory
 RENAME       -  to rename a file
 COPY         -  to copy a file
 DELETE       -  to delete (remove) a file
 LIST         -  to provide directory listing
 WRITE        -  to provide monitoring of a session
 READ         -  to redirect the input stream to read from a file
 SYNTAX       -  to provide the syntax of a command
 HELP         -  to access HELP system

 For further information type: HELP <command name> [<option>],
 where <option> is BRIEF or FULL
```

All the commands can be typed in upper or lower case. The syntax of each command can be obtained by using the `syntax` command. For example

```
 syntax partition
```

produces

```
Partition [Processors=<integer>] [,Method=<choice>] [,LEvel=<integer>]
          [,CGraph=<choice>] [,KLBISC=<choice>] [,KLREF=<choice>]
          [,PWeight=<real_list>] [,FIlepw=<string>]
```

To get full details on a command and its parameters, such as the `PARTITION` command, you can use `HELP PARTITION`.

```
Name    : PARTITION
Purpose : to partition the data
Syntax  : Partition [Processors=<integer>] [,Method=<choice>]
                    [,LEvel=<integer>] [,CGraph=<choice>]
                    [,KLBISC=<choice>] [,KLREF=<choice>]
                    [,PWeight=<real_list>] [,FIlepw=<string>]

Keyword        Type          Status       Current Value
----------------------------------------------------------------------

PROCESSORS     integer    retained      4
METHOD         choice     retained      GEO-BIS,costgeo,greedy,glutton,
                                        bandwdt,profile,inertia,r-iner,
                                        kl-greedy,kl-rand,spec,graph,
                                        kl-rgb,
LEVEL          integer    retained      5
CGRAPH         choice     retained      EDGE,truecomm
KLBISC         choice     retained      TRUE,false
KLREF          choice     retained      FALSE,true
PWEIGHT        real_list  reset         0
FILEPW         string     reset
```

A command can be abbreviated, the shortest value being indicated by the uppercase letters in the syntax section, e.g. the `Partition` command may be shortened to just `p`. The system is reasonably simple to use and working through one or two of the examples below should enable one to get to grips with it.

7.3 Internal Commands

Ralpar provides a number of *internal* commands. These commands, such as `HELP` and `COPY`, provide standard information and file handling from within *ralpar*. As with all commands, details of their usage can be obtained through the `HELP` command. A summary of these commands is given in the Command Appendix.

Those commands that access the file store do so by invoking the appropriate system command of the operating system being used. This means that in general if an report or error on an action is produced, these will be those of the host operating system.

For example, on UNIX systems, the `RENAME` command will use the UNIX command `mv`. Similarly `LIST` uses the UNIX command `ls`. Although the parameter types for these commands is *string*, the appropriate host systems file expression can be used. An example of this is:

```
LIST *.MSH
```

On a UNIX system this command will list all files with extension `.MSH` in the current working directory.

7.4 Data Input and Output

Ralpar currently supports three main formats for data input and output. These are:

RALBIC - A formal format defined for finite element meshes, full details of which are given in [10].

ASCII - A simple format which should be easier to use than the RALBIC one.

BINARY - A binary version of the ASCII format, mainly used to save and restore results quickly.

A full mesh is required as input, including the nodal coordinates and the element topology. There are three commands related to input and output operations. `Input` is used to read a mesh in one of the above formats. The `output` command will, by default, write a file with the mapping of elements to partitions in the selected format. Finally, `loadpar` can be used to read in the results of a previously saved partitioning operation. This requires that the mesh has already been read via `input`. The details of these formats is given in Appendix B.

7.5 Partitioning a two dimensional mesh

The file `CMPLX1.MSH` contains the RALBIC neutral file description of a two dimensional mesh with 165 quadrilateral elements and 200 nodes. Note that all RALBIC mesh files have the extension `.MSH` and that the filename is in uppercase. The `input` command will automatically add the extension and convert the name to upper case. Thus to read in this file we can use the command

```
Ralpar: input cmplx1
```

which is the same as

```
Ralpar: input file=cmplx1 type=ralbic
```

since the last parameter is optional and defaults to `ralbic`.

You can than view the mesh using the command

```
Ralpar: display
```

This should give a plot of the mesh similar to that shown in Figure 1.

To partition the mesh use either the `partition` or the `mlpart` commands. Thus, after having read the mesh with the `input` command, one can type

```
Ralpar: partition 4 geo
Ralpar: display
Ralpar: part 4 band
Ralpar: disp
Ralpar: mlpart 4 spec maxlvl=1 klref=full
Ralpar: disp
```

to compare the results of three different methods for the case of dividing the mesh into four parts. Typical results are shown in Figures 2, 3 and 4.

Note that after each `partition` command has been completed, the tool provides some information on the number of interface nodes that have been generated, such as:

```
 Inform: Interface node cost=    45
         Neighbour domains: Ave.=   2.500  Max.=   3  Min.=   2
 Inform: CPU time =    0.010 s
```

Also reported are the average number of domains about a domain and the minimum and maximum value of this quantity. For methods which are graph based, we also give the number of cut edges generated:

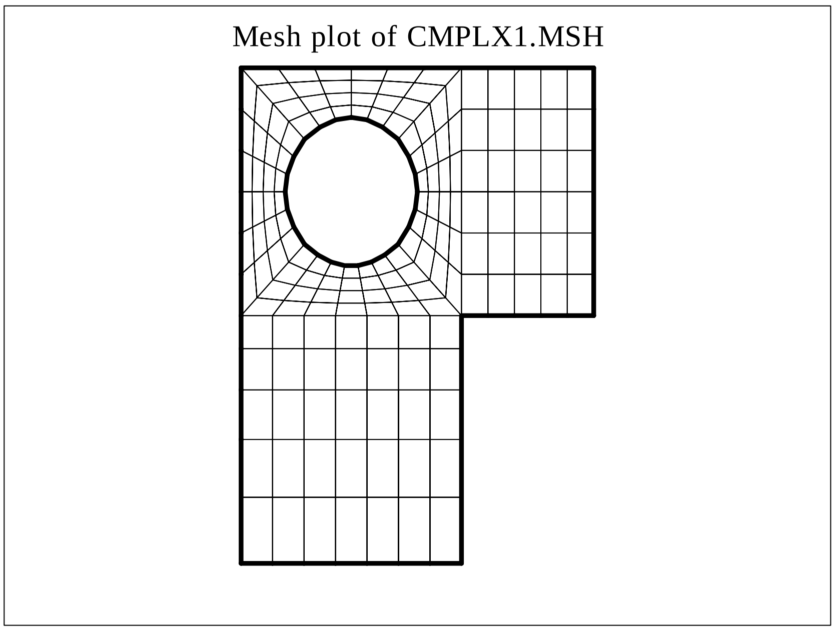

Figure 1: *The mesh* `CMPLX1.MSH` *as displayed using the* `display` *command.*

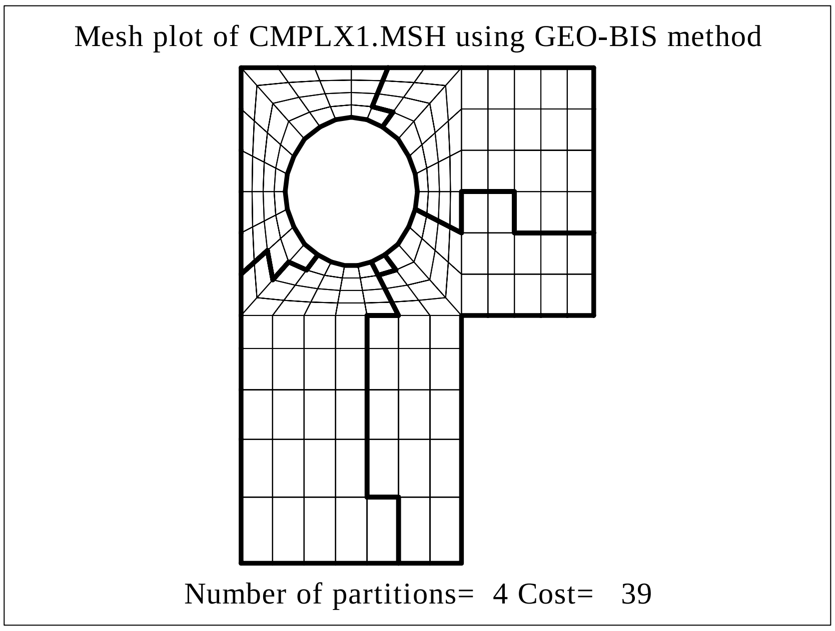
Mesh plot of CMPLX1.MSH using GEO-BIS method
Number of partitions= 4 Cost= 39

Figure 2: *The mesh* `CMPLX1.MSH` *partitioned into 4 using the geometric bisection method.*

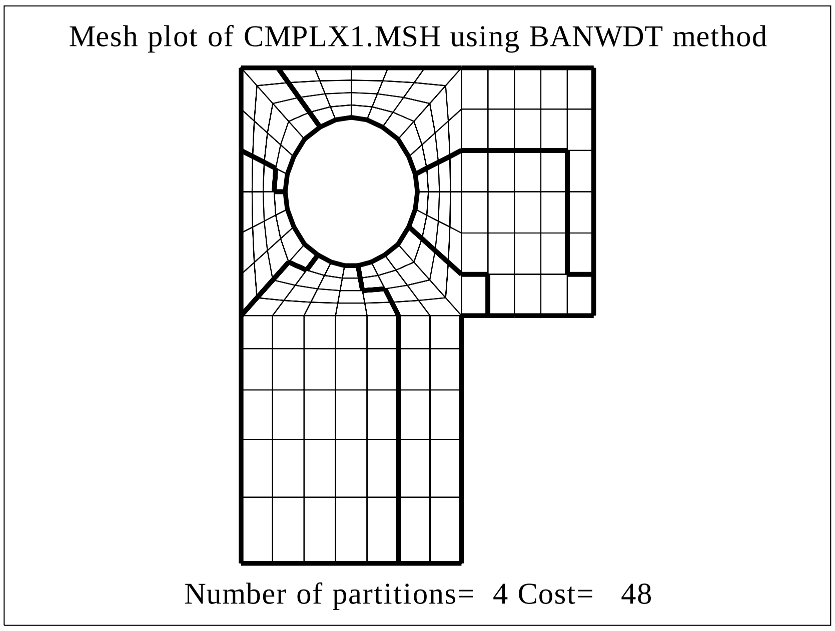
Mesh plot of CMPLX1.MSH using BANWDT method
Number of partitions= 4 Cost= 48

Figure 3: *The mesh* `CMPLX1.MSH` *partitioned into 4 using the bandwidth (Malone) method.*

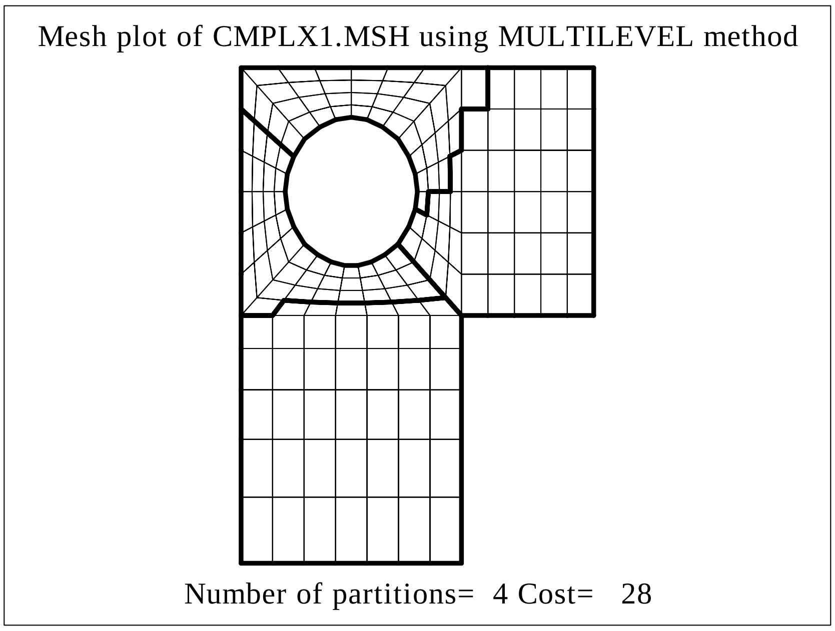
Mesh plot of CMPLX1.MSH using MULTILEVEL method
Number of partitions= 4 Cost= 28

Figure 4: *The mesh* `CMPLX1.MSH` *partitioned into 4 using a multi-level method.*

```
Ralpar: part 4 kl-gre
 Inform: Edge cut cost=     26 for    4 domains
 Inform: Interface node cost=    29
         Neighbour domains: Ave.=   2.500  Max.=   3  Min.=   2
 Inform: CPU time =    0.080 s
```

Note that the cut edge cost depends on the definition of the communication graph that is used. The default is to use the edge communication graph (described in the section on methods). Using the true communication graph with the same method, we get:

```
Ralpar: part 4 kl-gre cg=true
 Inform: Edge cut cost=     62 for    4 domains
 Inform: Interface node cost=    28
         Neighbour domains: Ave.=   2.500  Max.=   3  Min.=   2
 Inform: CPU time =    0.130 s
```

It should be noted that most optional parameters are *retained*, that is if you change the value in one command, it will become the new default. So for example, if the command `part 5` was issued after the above partition command, it would use the `kl-gre` method with the true communication graph.

Having made a partition of the mesh, the results can be written out to a file using the `output` command. By default this command will create a RALBIC file with the extension ".PAR". This will contain just a list of the mapping of element numbers to the partition they have been assigned to. ASCII or BINARY formats can be selected if required. If the parameter FULLMESH is set to TRUE, then the complete mesh will be written as well as the partition information. In this case the ".MSH" extension will be added to the file in the RALBIC case. For example, to write just the partition data to the file `CMPLX2.PAR`, one could issue the command:

```
Ralpar: output cmplx2
```

This partition could then be read back in at a later time with the command:

```
Ralpar: loadpar cmplx2
```

Appendix C gives details of the RALBIC and ASCII formats. This should allow conversion of existing data to a form that can be read by *ralpar*.

7.6 Partitioning a three dimensional mesh

All the same commands used for two dimensional meshes can be used the three dimensional ones. A simple three dimensional mesh, just consisting of one plane of hexahedral elements in the form of a "T", is given in the neutral file `T.MSH` which is provided with the standard *ralpar* release. Figure 5 shows the basic mesh.

The first method used is `geocost` and the result is shown in Figure 6. Other methods shown in Figures 7-9 are: `kl-rgb`, `profile` and `mlp` with a *spectral bisection* root.

The commands used to generate these partitions are:

```
Ralpar: input t
 Inform: Data file read: Nodes=  3402 Elements=  1600  Periodic nodes=    0
Ralpar: display sty=line
```

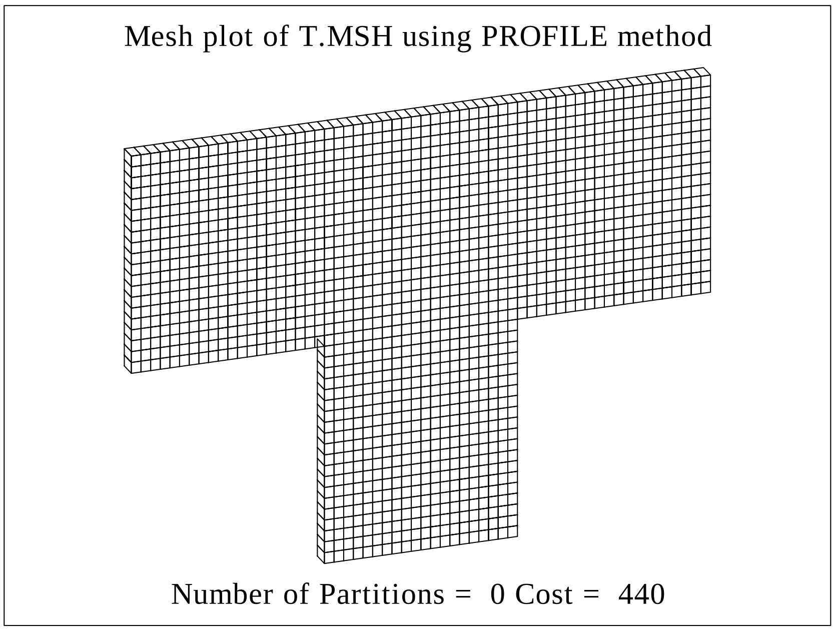
Mesh plot of T.MSH using PROFILE method
Number of Partitions = 0 Cost = 440

Figure 5: *The mesh* `T.MSH` *as displayed using the* `display` *command.*

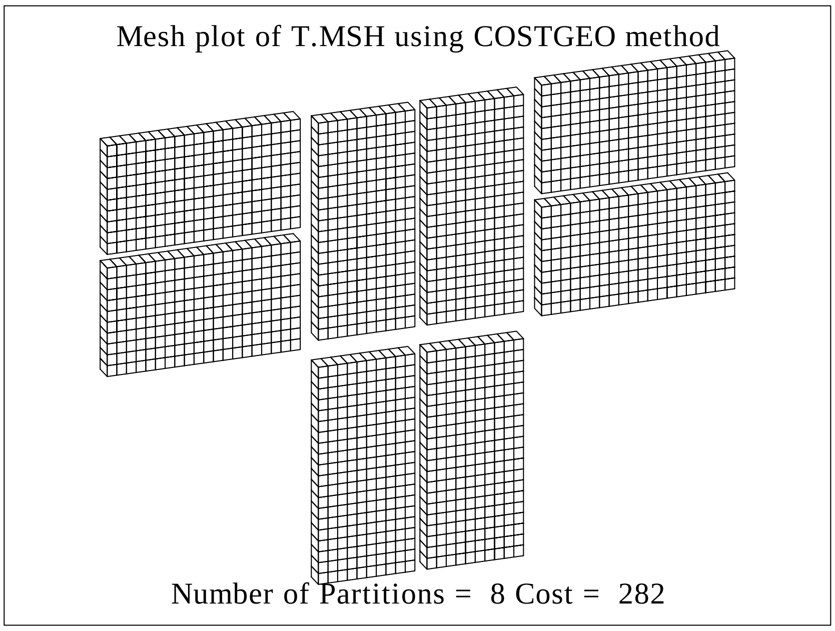
Mesh plot of T.MSH using COSTGEO method
Number of Partitions = 8 Cost = 282

Figure 6: *The mesh* `T.MSH` *partitioned into 4 using the cost variation of the geometric bisection method.*

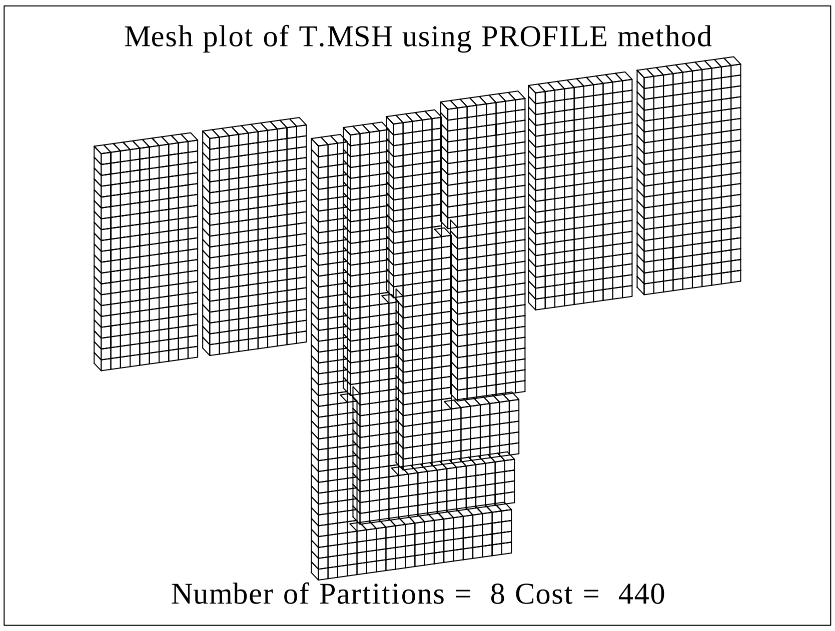
Mesh plot of T.MSH using PROFILE method
Number of Partitions = 8 Cost = 440

Figure 7: *The mesh* `T.MSH` *partitioned into 4 using Malone's method.*

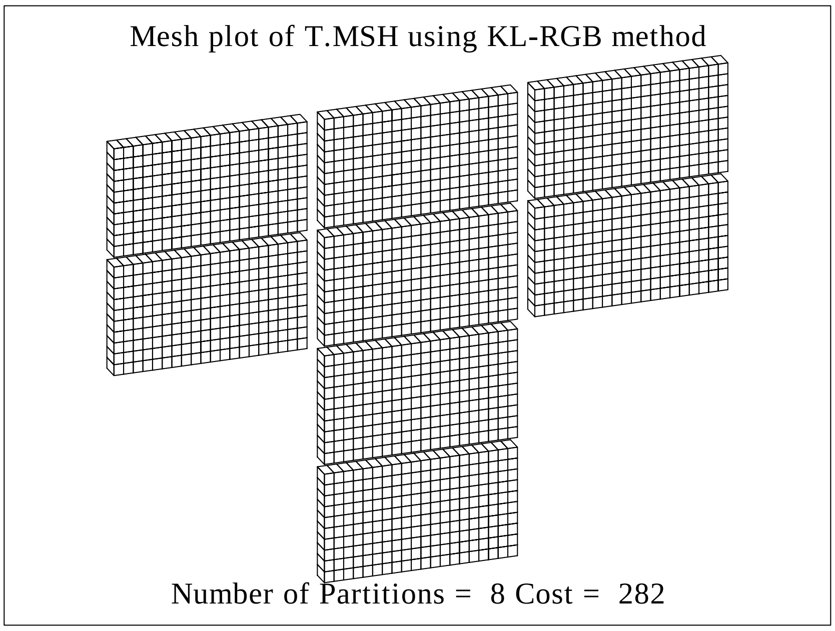
Mesh plot of T.MSH using KL-RGB method
Number of Partitions = 8 Cost = 282

Figure 8: *The mesh* `T.MSH` *partitioned into 4 using a Kernighan and Lin method with a rgb start.*

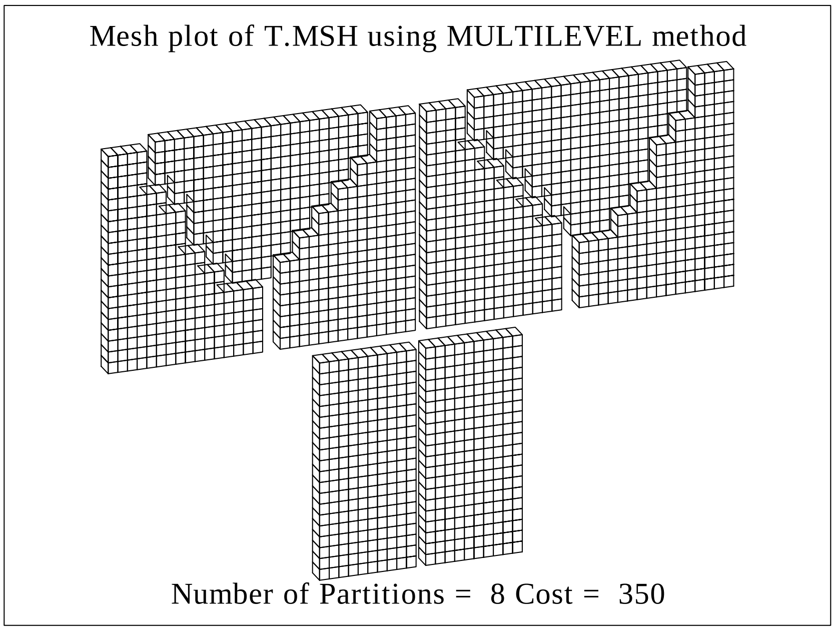
Mesh plot of T.MSH using MULTILEVEL method
Number of Partitions = 8 Cost = 350

Figure 9: *The mesh* `T.MSH` *partitioned into 4 using a multi-level method.*

```
Ralpar: part 8 costgeo
 Inform: Interface node cost=   282
         Neighbour domains: Ave.=   3.250  Max.=   5  Min.=   2
 Inform: CPU time =    0.672 s
Ralpar: display exp=0.25 sty=line
Ralpar: part 8 kl-rgb
 Inform: Edge cut cost=    432 for    8 domains
 Inform: Interface node cost=   326
         Neighbour domains: Ave.=   2.500  Max.=   4  Min.=   1
 Inform: CPU time =    1.920 s
Ralpar: disp
Ralpar: part 8 profile
 Inform: Interface node cost=   440
         Neighbour domains: Ave.=   1.750  Max.=   2  Min.=   1
 Inform: CPU time =    1.515 s
Ralpar: mlp 8 spec
 Inform: CPU time ML part =    5.884 secs
  Cut edges=  466
 Worst balance factor =  0.000000

 MINCON=     84   MAXCON=    166   AVECON=    116.500
 MINNEI=      2   MAXNEI=      5   AVENEI=      3.000
 Interface node cost =     350
 Inform: Interface nodes=    350    Cut edges=      0    Imbalance= 0.000000
         CPU time (secs)=       5.884
```

```
Ralpar: displ
Ralpar:
```

For this highly regular case the cost-geometric method happens to give the best result in terms of interface nodes.

7.7 Partitioning Elements of Different Weights

If we wish to assign non-uniform weights to each element we can do this with the `weight` command. In the mesh `T.MSH` all the elements are of the same type, so using the weighting on number of nodes (`weight nodal`) will have no effect. Instead we can use a file to specify the weights. This can be a normal text file with one number to a line which gives the weights in sequence for each element. If we assume such a file (`wfile`) has been written, then it can be used in the following way:

```
Ralpar: weight file wfile
 Using element weights from file:wfile
Ralpar: info high
Ralpar: part 2 glutton
 Inform: Target weight per partition =   0.120025E+04
         Min. weight =  0.119931E+04   Max. weight =   0.120119E+04
         Ratio (max. weight)/(ave. weight) =   0.100078E+01
 Inform: Using ISTART=  5
 Inform: Interface node cost=   100
         Neighbour domains: Ave.=   1.000  Max.=   1  Min.=   1
 Inform: Workspace used     26210 out of   2800000 I*4 words
         Minimum value of RALPAR_MEMORY_SF =  15
 Inform: CPU time =    0.730 s
```

Note that we have used the `information` command to increase the amount of detail that is reported in this case. Much of the extra detail is not usually wanted, though in this case is allows us to see how well the load balance has worked in the case with weighting. If highly non-uniform weighting are used it is possible to get poor load balancing results. An element weight is a real value greater than zero.

7.8 Partition Halos

In many computational methods the partition halo is required to enable a subdomain iteration process. Often this involves the iterative update of an overlap region. The *ralpar* command `HALO` generates a single depth element halo on all partition interfaces.

Those elements contained in the halo are all those that are connected to the interface nodes. Other types of halo or overlap are possible but a single level halo of this type seems currently the most used.

Figure 10 and Figure 11 show two examples of the halos generated by *ralpar*.

7.9 Comparing methods using the `table` command

The `machine` and `table` commands can be used to compare partitioning methods. Currently the `table` command implements the simple model for bus type communication architectures and the

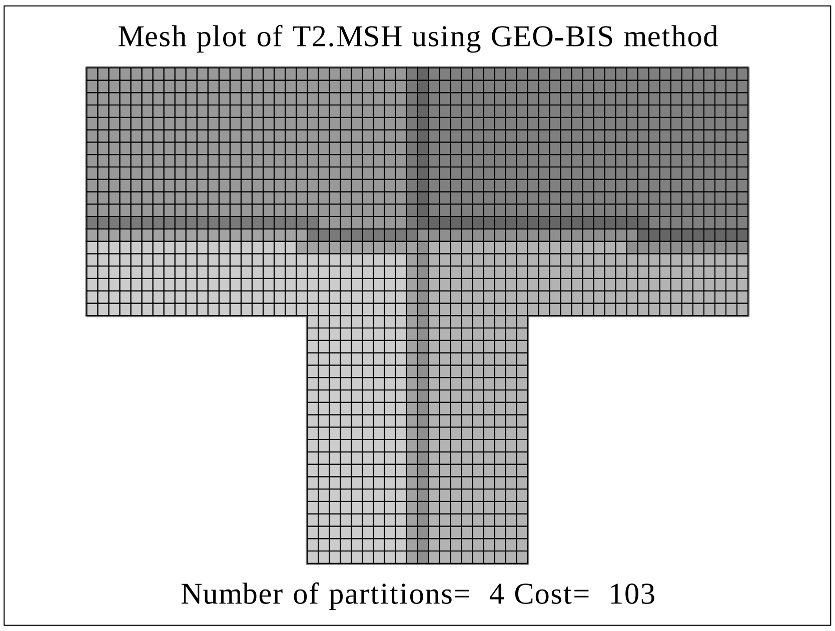
Mesh plot of T2.MSH using GEO-BIS method
Number of partitions= 4 Cost= 103

Figure 10: *The halo generate for the* `T.MSH` *mesh.*

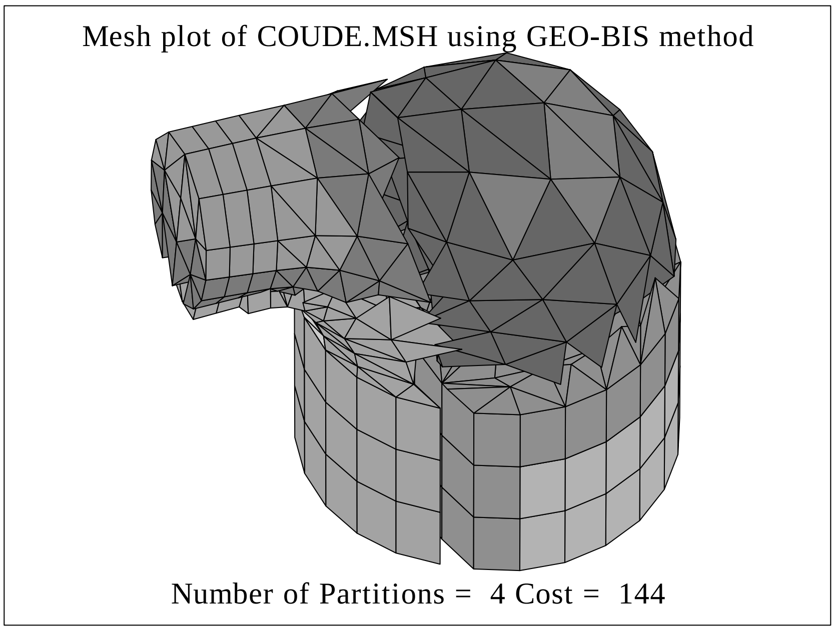
Mesh plot of COUDE.MSH using GEO-BIS method
Number of Partitions = 4 Cost = 144

Figure 11: *The halo generate for the* `COUDE.MSH` *mesh.*

parallel model, as described in the previous section. This command may be extended in future versions of *ralpar*.

Having read in a mesh file, such as `T.MSH`, it is necessary to first select the machine parameters to be used in the calculation. This is done with the `machine` command. To see what machines are available you can use the command:

```
Ralpar: machine action=display

 Machine                  Tstart    Tsend     n-half     R-inf


i860                     175.000     0.360   486.000     2.800
ipsc/2                   612.000     0.360  1750.000     2.800
SuperNode               1200.000     1.340   895.000     0.710
Transputer                 8.730     1.130     7.844     0.898
```

This gives names and parameters of the available machines.

It is possible to extend this list within the current run of *ralpar*. For example to add the parameters for PVM running over a Ethernet at RAL the commands would be:

```
Ralpar: mach add new 1500 1.5 name=pvm
```

This will cause `pvm` to be added to the list of available machines. Times are in microseconds and communication rate in Mbytes/s. New machine names and parameters are not automatically saved between runs of *ralpar*, though it is possible to use the `write` option of the `machine` command to save the data and reload it in another run with the `read` option.

To compare selected methods over a range of partition numbers, one can then use the `select` option of the `machine` command followed by the `table` command, e.g.

```
Ralpar: mach select pvm
Ralpar: table compute methods=(kl-rgb,prof) part=(2,4,8)
....
Ralpar: table disp data=seqcomm

Table for: Sequ. Comm. model
--------------------------------------
                 2             4             8
kl-rgb      3.984000E+03  1.646400E+04  4.838400E+04
prof        4.104000E+03  1.111200E+04  2.628000E+04
```

The table gives the total time in microseconds for communication of one value per interface node, based on the models discussed previously. In this case the high cost of the start up times means that the `profile` method, which minimises the number of neighbour domains, gives better results than the `kl-rgb` method which gives fewer interface nodes, when for 4 or 8 processors are used.

References

[1] HA Schwarz: “Uber einige Abbildungsauf-gaben”, *Ges. Math. Abh.* **11** 65–83 (1869).

[2] YF Hu and R Blake, “Numerical experiences with partitioning unstructured grids”, Daresbury Laboratory Report, DL/SCI/P865T, March 1993.

[3] C Farhat, W Wilson and G Powell: “Solution of Finite Element Systems on Concurrent Processing Computers” *Engineering with Computers*, **2**, 157–165, (1987).

[4] RF Fowler, BW Henderson, and C Greenough, “Initial Experiences in Porting a Three-Dimensional Semiconductor Device Modelling Program to the Intel iPSC/860”, Rutherford Appleton Laboratory Report, RAL-92-090 (1992).

[5] YF Hu and RJ Blake: “Numerical Experiences with Partitioning Unstructured Meshes”, Daresbury Laboratory Report, DL/SCI/P865T, March 1993.

[6] B Kernighan and S Lin: “An efficient heuristic procedure for partitioning graphs”, Bell System Technical Journal, 29 (1970), pp. 291-307.

[7] JG Malone: “Automatic Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessor Computer”, Comp. Meth. in Applied Mechanical Eng., **70**, 27–58 (1988).

[8] Algorithm 582, Collected algorithms from ACM, ACM-Trans. Math. Software, Vol. 8, No. 2, p. 190, June 1982.

[9] C Greenough and RF Fowler: “Partitioning Methods for Unstructured Finite Element Meshes”, Rutherford Appleton Laboratory Report, RAL-94-092.

[10] CRI Emson, C Greenough, NJ Diserens and KP Duffy, “RALBIC - A Simple Neutral File for Finite Element Data: File Definition”, RAL Report RAL-87-102, 1987.

[11] C Greenough and RF Fowler: “A Review of Partitioning Methods for Unstructured Finite Element Meshes”, Rutherford Appleton Laboratory Report, (To be published).

[12] ST Barnard and HD Simon, “A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems”, Proceedings of the 6th SIAM conf. on parallel processing for scientific computing”, p711-718, 1993.

A The *ralpar* Command Summary

This Appendix provides a summary of all the commands available within *ralpar*. These commands are listed below in alphabetic order, with full details of each in the corresponding section.

Application commands

A.1	DISPLAY	- to display partition results
A.2	HALO	- to construct element halo regions
A.3	INFORMATION	- to control message output
A.4	INPUT	- to read in a mesh file
A.5	LOADPAR	- to read in partition data
A.6	MACHINE	- to specify machine constants
A.7	MLPART	- to partition using a multilevel method
A.8	OUTPUT	- to write out partition data
A.9	PARTITION	- to partition the mesh
A.10	PLIST	- to list element numbers in a partition
A.11	QUIT	- to quit program
A.12	TABLE	- to evaluate or display table of costs
A.13	VIEW	- to set viewing angle for the next plot
A.14	WEIGHT	- to define element weights

Internal commands

A.15	CHANGE	- to change working directory
A.16	COPY	- to copy a file
A.17	DELETE	- to delete a file
A.18	HELP	- to access HELP system
A.19	LIST	- to provide directory listing
A.20	READ	- to redirect the input stream to read from a file
A.21	RENAME	- to rename a file
A.22	SYNTAX	- to provide the syntax of a command
A.23	WRITE	- to provide monitoring of a session

A.1 DISPLAY (Application) - to display partition results

Syntax

Display [Boundary=<*choice*>] [,Mesh=<*choice*>] [,Label=<*choice*>]
[,DEvice = <*choice*>] [, STyle = <*choice*>] [,SHrink = <*real*>]
[,EXplode = <*real*>] [,DOmain =<*integer*>]

Description

The DISPLAY command plots the current mesh and partition on the selected output device. The finite elements in the mesh can be shrunk to help see the partition boundaries. The partitions are shown in different colours and the partition boundaries in heavy lines (2D only). The command provides parameters to control the style of graphical output.

Parameters

BOUNDARY retained *choice* : initial = YES

The BOUNDARY parameter controls the display of the partition boundaries. Its values are: NO (no boundaries) and YES (partition boundaries are displayed). Only applies to 2D meshes.

MESH retained *choice* : initial = YES

The MESH parameter controls the display of the mesh. Its values are: NO (no mesh) and YES (mesh displayed). Only applies to 2D meshes.

LABEL retained *choice* : initial = NO

The LABEL parameter controls the display of the node and element numbers.Its values are: NO (no labels) and YES (node and element numbers displayed). Only applies to 2D meshes.

DEVICE retained *choice* : initial = SCREEN

The DEVICE parameter controls the destination of the graphical output. This can be either to the screen or to a file (e.g. if a PostScript driver is available). Its values are: SCREEN (output to screen) and FILE. Note: the the action of option FILE is dependent of the host graphics system.

STYLE retained *choice* : initial = SOLID

The STYLE parameter controls how elements are drawn, either as a SOLID (filled area) or LINE (outline only).

COLMAP retained *string* : initial = ’ ’

The COLMAP parameter allows the user to control the colours used by display. COLMAP should be a valid file name in which the RGB values of the colour defined.

The file structure is:

n - the number of colours defined in the file
m r-val g-val b-val - the colour number followed by RGB values
. . . .
. . . .

The RGB values are given in the range [0-1].

SHRINK retained *real* : initial = 1.0

The SHRINK parameter controls the size of elements. A value of 1.0 gives full size while smaller values shrink each element.

EXPLODE retained *real* : initial = 0.0

The explode factor only applies to 3D meshes. If a value greater than zero is given, domains are moved apart to reveal internal interfaces.

DOMAIN retained *integer* : initial = 0

If greater than zero this selects a single domain to be displayed. Otherwise all domains are shown. Applies to 3D meshes only.

Examples

```
Ralpar: DISPLAY
Ralpar: dis yes yes yes
Ralpar: Disp shrink=.6 explod=.5
```

A.2 HALO (Application) - to construct element halo regions

Syntax

HALo [DOMAIN=<*integer*>]

Description

The HALO command allows the user to control the generation of halo information by the program. A *halo* is defined to be a region along the boundary of a partition which has a one-element thickness. These elements are indicated by negative partition numbers in output lists and darker shading in DISPLAY.

Parameters

DOMAIN retained *integer* : initial = 0

If greater than 0, this parameter is the single domain in which the halo elements should be marked.

Examples

```
Ralpar: HALO
Ralpar: hal 3
```

A.3 INFORMATION (Application) - control message output

Syntax

INFormation [LEvel=<*choice*>]

Description

The INFORMATION command allows the user to control the amount of information provided by the program during its execution.

Parameters

LEVEL retained *choice* : initial = QUIET

The LEVEL parameter controls the level of information output. Its values are: QUIET, MEDIUM, HIGH and VERBOSE.

Examples

```
Ralpar: INFORMATION VERBOSE
Ralpar: info q
```

A.4 INPUT (Application) - to read in mesh file

Syntax

Input [File=<*string*>] [,Type=<*choice*>] [,Access=<*choice*>]

Description

The INPUT commands reads a complete mesh from the specified file. The description contains the nodal positions and the element topologies. The command allows for the a number of input formats: RALBIC, ASCII, BINARY, TEST. If the file was generated by a previous OUTPUT command from RALPAR with partition information, this data will also be read.

Parameters

FILE required *string*

The FILE parameter specifies the file name in which the data is held.

TYPE retained *choice* : initial = RALBIC

The TYPE parameter specifies the type of format in which the data is stored. It has values: RALBIC, ASCII, BINARY and TEST.

TESTSIZE retained *integer_list* : initial = ()

This parameter is only used when generating test meshes, (TYPE=TEST). Test meshes are regular hexahedral grids and the integer list must give the number of nodes in X, Y and Z directions. If only one node plane is specified in Z, a 2D mesh results.

Examples

```
Ralpar: INPUT FILE=DATA, TYPE=FELIB, ACCESS=FORMATTED
Ralpar: i data
Ralpar: Inp type=test testsize=(9 17 1)
```

A.5 LOADPAR (Application) - to read in partition data

Syntax

LOadpar [File=<*string*>] [,Type=<*choice*>]

Description

The LOADPAR command reads a file containing a previously calculated partition for the current mesh. It checks that a mesh has been read and that the number of elements in it and the data file are consistent. A number of different file formats are allowed, as described for the OUTPUT command. To read a file that has been written with `OUT FULLMESH=T`, the INPUT command must be used instead.

Parameters

FILE required *string*

The FILE parameter specifies the file name in which the data is held. For RALBIC files, do not include the ".PAR" extension, and note that these file names are automatically converted to uppercase.

TYPE retained *choice* : initial = RALBIC

The TYPE parameter specifies the type of format in which the data is stored. It has values: RALBIC, ASCII, BINARY or BERTIN.

Examples

```
Ralpar: LOADPAR FILE=DATA, TYPE=ASCII
Ralpar: load oldpart
```

A.6 MACHINE (Application) - to specify machine constants

Syntax

MAChine [ACtion=<*choice*>] [,Type=<*choice*>] [,TSTART=<*real*>]
[,TSEND=<*real*>] [,NAme=<*string*>] [,FILename=<*string*>]

Description

The MACHINE command allows the user to select and modify the machine constant table. These constants are used by the TABLE command for simple estimates of parallel communication times. New machines can be added into the list of names along with known values of communication parameters. The list of machine details can be saved to (or restored from) a file.

Parameters

ACTION retained *choice* : initial = SELECT

The ACTION parameter control the process of the command. The current actions are:

SELECT - To select a particular set of machine values.
READ - To read a table of values from FILENAME.
WRITE - To write the machine constant table to FILENAME.
DISPLAY - To display the current table values.
ADD - To add a new machine to the table.

The TABLE command will use the machine parameters that have been SELECT'ed.

TYPE retained *choice* : initial = ipsc/860

The TYPE parameter specifies which set of machine parameters are to be used. Parameters are included for iPSC/860, iPSC/2, SUPERNODE and TRANSPUTER systems, but this list can easily be extended.

TSTART retained *real* : initial = 175.0

The TSTART parameter specifies the communication start-up time in microseconds. This parameter is used with ACTION=ADD.

TSEND retained *real* : initial = 0.36

The TSEND parameter specifies the communication rate in Mbytes per second. This parameter is used with ACTION=ADD.

NAME reset *string* : initial =

The NAME parameter specifies the name of the new machine to be added to the machine constants table. This parameter is used with ACTION=ADD.

FILENAME retained *string* : initial = machine.cst

The FILENAME parameter specifies the filename where the machine constants table is stored (ACTION=WRITE) or read from (ACTION=READ).

Examples

```
 Ralpar: MACHINE TYPE=SUPERNODE ACTION=SELECT
 Ralpar: mach add new 1500 1.5 pvm
 Ralpar: machine action=dis

 Machine                Tstart    Tsend     n-half     R-inf

ipsc/860               175.000     0.360   486.000     2.800
ipsc/2                 612.000     0.360  1750.000     2.800
SuperNode             1200.000     1.340   895.000     0.710
Transputer               8.730     1.130     7.844     0.898
paragon                175.000     0.360   486.000     2.800
```

A.7 MLPART (Application) - to use a multilevel partitioning method

Syntax

MLPart [NPart=<*integer*>] [,METHod=<*choice*>]
  [,SECtion=<*choice*>] [,CGraph=<*choice*>]
  [,CLUst=<*choice*>] [,MAXLvl=<*integer*>]
  [,MINSiz=<*integer*>] [,KLRef=<*choice*>]
  [,KLLim=<*integer*>] [,PWeight=<*real_list*>]
  [,FILEPW=<*string*>]

Description

The MLPART command provides access to a set of partitioning routines that can make use of multilevel techniques. These methods include some of those available in the PARTITION command, but with greater flexibility.

Parameters

NPART retained *integer* : initial = 4

The NPART parameter specifies the number of partitions into which the mesh is to be partitioned. The number of partitions must be less than the number of elements in the mesh.

METHOD retained *choice* : initial = SPECT

This parameter specifies the partitioning method to be used. The current methods are:

MALONE - Element re-ordering for minimum profile width.
SPEC - Spectral bisection.
GRAPH - Graph bisection.
RAND - Random partitioning.
DGRAPH - Variation on graph bisection.
GREED - Greedy algorithm based on that of Farhat.
GLUT - Extended Greedy algorithm.

SECTION retained *choice* : initial = BISECT

The SECTION parameter specifies the way in which the chosen method should be applied to partition the mesh.

LINSEC - The selected method is called once to get a single ordering of the elements. This list is then split into the required number of partitions.
BISECT - The selected method is used recursively to split the mesh into two parts until the required number of partitions is generated.
RECSECT - Each required partition is split away from the remaining mesh, one at a time.

CGRAPH retained *choice* : initial = EDGE

This value controls how edges in the communication graph will be generated from the mesh. Possible values are:

EDGE - Any two elements with a common edge (2D) or face (3D) will be connected.

TRUE - Any elements with one or more nodes in common are connected.

WEIGHT - Any elements with one or more nodes in common are connected, and the connection is weighted by the number of such common nodes.

CLUST retained *choice* : initial = A

This parameter controls how clusters are formed. At each level of graph reduction, vertices are joined together. The default method (CLUST=A) is a greedy one where non-clustered vertices are joined to all neighbours that are not already in a cluster. CLUST=B merges each free vertex to the free neighbour with greatest edge weight.

MAXLVL retained *integer* : initial = 1

This parameter controls how many levels the graph will be condensed through before being partitioned.

MINSIZ retained *integer* : initial = 20

When a graph contains less than MINSIZ vertices it will not be reduced any further, even in the value of MAXLVL would allow this.

KLREF retained *choice* : initial = NONE

Any of the partitioning methods available to MLPART can be combined with Kernighan and Lin style refinement. This can be applied to either just the lowest graph level (KLREF=MINLVL) or at all levels of the graph (KLREF=FULL). The latter gives better results but is slower. Currently, refinement can not be used with LINSECT option.

KLLIM retained *integer* : initial = 0

The Kernighan and Lin refinement process involves repeated passes through all vertices, searching for a better configuration. When set to a value greater than zero, KLLIM causes these loops to terminate early if the reduction in cut edges falls below the current best value less KLLIM. This speeds up the method, at the cost of sometimes giving poorer results. Typical values to try are 100 to 5000 depending on mesh size.

PWEIGHT reset *real_list* : initial = 0

The weights of each partition may be set by the PWEIGHT parameter. All weights must be positive. If neither PWEIGHT or FILEPW are specified, all partitions will have the same weight.

FILEPW reset *string* : initial = “”

Partition weights will be read from the file named FILEPW if this parameter is set. The file is read in free format and should contain one number per line giving the weight of each partition in sequence. All values must be greater than 0.

Examples

```
 Ralpar: MLPART 2 SPEC BIS MAXLVL=2
 Ralpar: mlp 32 klref=full
 Ralpar: mlp 20 spec lin klref=none
 Ralpar: MLP 6 mal pweight=(4,2,2,1,1,1)
```

A.8 OUTPUT (Application) - to write neutral file with partition numbers

Syntax

Output [FIle=<*string*>] [,Type=<*choice*>] [,FULLmesh==<*choice*>]

Description

The OUTPUT command writes the current partition data to a file in one of the available formats. With the FULLMESH=TRUE option the whole mesh can be written to output file as well as the partition data.

Parameters

FILE retained *string* : initial = RALPAR

The FILE parameter specifies the file name in which the data is held.

TYPE retained *choice* : initial = RALBIC

The TYPE parameter specifies the format in which the data is to be stored. It can take the values: RALBIC, ASCII or BINARY. RALBIC files will have the extension ".PAR" appended to the name for partition information, or ".MSH" if the full mesh is written. RALBIC filenames are always translated to uppercase.

FULLMESH retained *choice* : initial = FALSE

The FULLMESH parameter specifies whether the whole mesh should be written to the file, or just the partition data (the default).

Examples

```
Ralpar: OUTPUT FILE=PART01 TYPE=RALBIC
Ralpar: out data900 ascii true
```

A.9 PARTITION (Application) - to partition the data

Syntax

Partition [Processors=<*integer*>] [,Method=<*choice*>]
[,LEvel=<*integer*>] [,CGraph=<*choice*>]
[,KLBISC=<*choice*>] [,PWeight=<*real_list*>]
[,FIlepw=<*string*>]

Description

The PARTITION command controls the partitioning of the meshes read into the program. The command specifies the number of partitions, the method to be used and the values of some control parameters. For multilevel methods, see the MLPART command.

Parameters

PROCESSORS retained *integer* : initial = 4

The PROCESSORS parameter specifies the number of partitions into which the mesh is to be split. The number of partitions must be compatible with the method being used.

METHOD retained *choice* : initial = GEO-BIS

The METHOD parameter specifies the partitioning method to be used. The current methods are:

GEO-BIS - geometric bisection
COSTGEO - geometric bisection using lowest cost directions
GREEDY - greedy algorithm due to Farhat
GLUTTON - greedy algorithm with multiple seed point searching for minimum cost
BANDWTH - nodal re-ordering for minimum bandwidth (Malone)
PROFILE - nodal re-ordering for minimum profile width
INERTIA - bisection based on axes of inertia
R-INER - recursive inertial bisection
KL-GREEDY - Kernighan & Lin with GREEDY starting point
KL-RAND - Kernighan & Lin with random starting point
SPEC - recursive spectral bisection
GRAPH - recursive graph bisection
KL-RGB - Kernighan & Lin with RGB starting point

LEVEL retained *integer* : initial = 5

The LEVEL parameter specifies the number of levels to be searched in the GLUTTON algorithm.

CGRAPH retained *choice* : initial = EDGE

The CGRAPH parameter specifies the type of communication to be used in generating the communication graph. Possible values are EDGE or TRUE.

KLBISC retained *choice* : initial = TRUE

The KLBISC parameter specifies whether the Kernighan & Lin methods are to be recursive bisection methods. If FALSE, then one subdomain at a time is split from the remaining mesh. This can be expensive for large numbers of partitions.

KLREF retained *choice* : initial = FALSE

The KLREF parameter specifies whether Kernighan & Lin refinement should be used with the SPECTRAL method. All other methods are unaffected by this parameter.

PWEIGHT reset *real_list* : initial = 0

The weights of each partition may be set by the PWEIGHT parameter. All weights must be positive. If neither PWEIGHT or PFILE are specified, all partitions will have the same weight.

FILEPW reset *string* : initial =

Partition weights will be read from the file named FILEPW if this parameter is set. The file is read in free format and should contain one number per line giving the weight of each partition in sequence. All values must be greater than 0.

Examples

```
Ralpar: PARTITION PROC=16 METHOD=KL-RAND CGRAPH=TRUE
Ralpar: par 32 geo
Ralpar: part 6 prof pweight=(4,2,2,1,1,1)
```

A.10 PLIST (Application) - to list element numbers in a partition

Syntax

PList [Partition=<*integer*>] [,Limit=<*integer*>]

Description

The PLIST command allows the user to list the elements allocated to one or all partitions. A limit on the number of element listed per partition is useful for large meshes.

Parameters

PARTITION retained *integer* : initial = 0

The PARTITION parameter specifies which partition is to be listed. If this is set to zero, all partitions will be listed.

LIMIT retained *integer* : initial = 50

No more than LIMIT elements will be listed for each subdomain.

Examples

```
Ralpar: PLIST PARTITION=2 LIMIT=10
Ralpar: pl 0 50
Ralpar: plist
```

A.11 QUIT (Application) - to quit program

Syntax

```
Quit
```

Description

The QUIT command closes all files and terminates the program.

Parameters

This command has no parameters.

Examples

```
Ralpar: quit
```

A.12 TABLE (Application) - to evaluate or display table of costs

Syntax

TABle [ACtion=<*choice*>] [,MEthods=<*string*>]
[,PARtitions=<*string_list* >] [,FILename=<*integer_list* >]
[,DAta=<*choice*>]

Description

The TABLE command allows the user to compare a range of methods over a number of partitions. The user has to define the methods to be used, and the number of partitions which are used in the `COMPUTE` action. Results can then be viewed with the `DISPLAY` action or written to file with `WRITE`.

Parameters

ACTION retained *choice* : initial = COMPUTE

The ACTION parameter specifies the action to be taken by the TABLE command. Currently values are:

COMPUTE - to perform the cost table calculations

DISPLAY - to display the calculated cost table

WRITE - to write a calculated cost table to file

FILE retained *string* : initial = ralpar.ctab

The FILE parameter specifies the file name to write the cost table to, if ACTION=WRITE.

METHODS retained *string_list* : initial = (GEO-BIS,INERTIA)

The METHODS parameter specifies the methods to be used in the table calculations. Available methods are: GEO-BIS, COSTGEO, GREEDY, GLUTTON, BANDWDT, PROFILE, INERTIA, R-INER, KL-GREEDY, KL-RAND, SPEC, GRAPH and KL-RGB. Used when ACTION=COMPUTE.

PARTITIONS retained *integer_list* : initial = (2,4)

The PARTITIONS parameter specifies the partitions that are to be used for each method in calculating the cost table. Used when ACTION=COMPUTE.

DATA retained *choice* : initial = SEQCOMM

The DATA parameter specifies which results are to be shown when ACTION=DISPLAY or WRITE is used. Options available are: ALL, SEQCOMM, PARACOMM, INTERFACE, NEIGHBOURS. Note that all the data is calculated when ACTION=COMPUTE, irrespective of the setting of this option.

Examples

```
Ralpar: table methods=(geo-bis,costgeo,greedy) part=(2,4,8,16)
Ralpar: TABLE ACTION=WRITE FILE=COST-TABLE data=all
Ralpar: TABLE display data=seqcomm
```

A.13 VIEW (Application) - to set viewing angle for the next plot

Syntax

VEIw [XROT=<*real*>] [, YROT=<*real*>] [,ZROT=<*real*>]
[,SCALE=<*real*> >]

Description

The VIEW command allows the user to change the orientation of 3D meshes and set the scaling. 2D meshes are not altered by this command. The effect of the new view is only seen in subsequent display commands.

Parameters

XROT retained *real* : initial = 30

The XROT parameter specifies the rotation about the x axis in degrees.

YROT retained *real* : initial = 20

The YROT parameter specifies the rotation about the y axis in degrees.

ZROT retained *real* : initial = 0

The ZROT parameter specifies the rotation about the z axis in degrees.

SCALE retained *real* : initial = 1

The SCALE parameter specifies a scaling factor to be applied to the display.
Not currently implemented.

Examples

```
Ralpar: VIEW XROT=30 YROT=30
Ralpar: view 40 0 10
```

A.14 WEIGHT (Application) - to define element weights

Syntax

WEight [MEthod=<*choice*>] [,File=<*string*>]

Description

The WEIGHT command allows the user to give weighting to the elements of a mesh being partitioned. The weight of elements can be used to improve load balance in applications where mixed element types are present, or other factors are known to alter the computational work per element. Weighting can be set to the number of nodes in each element or to values read from a file.

Parameters

METHOD retained *choice* : initial = UNIFORM

The METHOD parameter specifies the weighting method to be used. Possible values are:

UNIFORM - all elements have the same weight

NODAL - elements are weighted according to the number of nodes they contain

FILE - the weighting information is to be read from a file

FILE retained *string* : initial = ”

The FILE parameter specifies the file from which weighting information is to be taken. Only used if ACTION=FILE. If use, the file must exist and be a text file.

Examples

```
Ralpar: WEIGHT METHOD=UNIFORM
Ralpar: we file weighting-info
Ralpar: weight nodal
```

A.15 CHANGE (Internal) - to change working directory

Syntax

CHAnge [DIRectory=<*string*>]

Description

The CHANGE command allows the user to change the current working directory without leaving the program.

Note that the CHANGE command is an INTERNAL command and is system dependent.

Parameters

DIRECTORY reset *string* : initial = '.'

DIRECTORY specifies the directory name to which the context should be changed. This must be a valid name for the host system.

Examples

Some examples on UNIX systems are:

```
Ralpar: change /u/cg/ralpar/tests
Ralpar: CHA ../tests
```

A.16 COPY (Internal) - to copy a file

Syntax

COPy [FILe1=<*string*>] [,FILe2=<*string*>]

Description

The COPY command allows the user to copy one file to another without leaving the program.

Note that the COPY command is an INTERNAL command and is system dependent.

Parameters

FILE1 required *string*

The FILE1 parameter specifies the source file. FILE1 must be a valid file name or expression on the host system.

FILE2 required *string*

The FILE2 parameter specifies the target file or directory. FILE2 must be a valid file name or expression on the host system.

Examples

Some examples on UNIX systems are:

```
Ralpar: COPY /u/cg/ralpar/tests/test1 data
Ralpar: cop ../tests/example .
```

A.17 DELETE (Internal) - to delete (remove) a file

Syntax

DELete [FILe=<*string*>]

Description

The DELETE command allows the user to delete files from the file system without leaving the program.

Note that the DELETE command is an INTERNAL command and is system dependent.

Parameters

FILE required *string*

FILE specifies the file(s) to be deleted. FILE must be a valid file name or expression on the host system.

Examples

Some examples on UNIX systems are:

```
Ralpar: DELETE data
Ralpar: del ../tests/example
```

A.18 HELP (Internal) - to access HELP system

Syntax

Help [KEY=<*string*>] [,OPTion=<*choice*>]

Description

Gives access to the inbuilt HELP system within the command decoder. HELP is one of the internal commands of the command processor and has a companion command SYNTAX.

HELP has two parameters allowing the selection of help on a specific command and the level of help required (SUMMARY, BRIEF and SYNTAX). If no command name is given summary help is given on all the commands currently defined.

If an ambiguous or invalid command name is given a warning or error message is given.

BRIEF help gives information on the purpose, syntax and the current state of the selected command. A table of command keywords, their type, status and current value (if applicable) is printed.

Parameters

KEY reset *string* : initial =

Either the global command name SUMMARY, or the specific command name on which help is sought.

OPTION reset *choice* : initial = BRIEF

The level of help required. This can be SUMMARY, BRIEF or SYNTAX.

Examples

```
 Ralpar: help plist

Name    : PLIST

Purpose : to list element numbers in a partition
Syntax  : PList [Partition=<integer>] [,Limit=<integer>]


Keyword       Type            Status       Current Value
----------------------------------------------------------

PARTITION     integer         retained      0
LIMIT         integer         retained      50
```

A.19 LIST (Internal) - to provide directory listing

Syntax

LISt [FILe=<*string*>]

Description

The LIST allows the user to list the files available in the in the system file store.

Note that the LIST command is an INTERNAL command and is system dependent.

Parameters

FILE reset *string* : initial =

The FILE parameter specifies a file name or file mask over which the listing is to search. (*The file mask will be system dependent*).

Examples

Some examples on UNIX systems are:

```
Ralpar: LIST FILE=data
Ralpar: lis "*.f*"
```

A.20 READ (Internal) - to redirect the input stream to read from a file

Syntax

REad [FIle=<*string*>] [,ECHO=<*choice*>]

Description

The READ allows the user to execute a set of commands that have placed in a file. Lines is the file are processed as normal commands and information and error reports are directed to the terminal.

Note that the READ command is an INTERNAL command and is system dependent.

Parameters

FILE required *string*

Input file name containing program commands.

ECHO reset *choice* : initial = OFF

Echo control option. If set to OFF, then commands read from the file are not echoed to the terminal. Echo is enabled by the value ON.

Examples

```
Ralpar: READ FILE=script, ECHO=ON
Ralpar: read long-run off
```

A.21 RENAME (Internal) - to rename a file

Syntax

REName [FILe1=<*string*>] [,FILe2=<*string*>]

Description

The RENAME command allows the user to rename files from the file system without leaving the program.

Note that the RENAME command is an INTERNAL command and is system dependent.

Parameters

FILE1 required *string*

The FILE1 parameter specifies the file to be renamed. FILE1 must be a valid file name or expression on the host system.

FILE2 required *string*

The FILE parameter specifies the target file. FILE2 must be a valid file name or expression on the host system.

Examples

```
Ralpar: rename xyz.msh XYZ.MSH
```

A.22 SYNTAX (Internal) - to provide the syntax of a command

Syntax

SYNtax [COMmand=<*string*>]

Description

Displays the formal syntax of all the currently defined commands. If the syntax of a specific command name is required then that name is given as a parameter to the command.

Parameters

COMMAND retained *string* : initial = ALL

Specifies the commands name for which the syntax is required. If the syntax of all the currently defined commands is required, then the special command name ALL should be used.

Examples

```
Ralpar: syntax machine

 MAChine [ACtion=<choice>] [,Type=<choice>] [,TSTART=<real>]
         [,TSEND=<real>] [,FILename=<string>] [,NAme=<string>]
```

A.23 WRITE (Internal) - to provide monitoring of a session

Syntax

WRIte [STAte=<*choice*>] [,FIle=<*string*>] [,PROmpt=<*choice*>]

Description

Redirects the command decoder echo output to the file specified by the FILE parameter. The information flow is controlled by the STATE parameter. This command can enable the constructions of command files to drive the program in a background mode.

The echoing of the command prompt can be controlled using the PROMPT parameter.

Parameters

STATE required *choice*

Controls the flow of information to the monitoring file It has values ON, OFF or CLOSE. ON switches on monitoring. OFF suspends it but does not close the file and CLOSE ends monitoring and closes the file.

FILE retained *string* : initial = MONITOR

Output file name to receive the monitoring stream.

PROMPT reset *choice* : initial = OFF

Allows you to select whether the command prompt is echoed in the monitoring file. It has values ON or OFF.

Examples

In this example commands are written to the file MONITOR without the prompt:

```
Ralpar: WRITE STATE=ON FILE=MONITOR PROMPT=OFF
```

B File formats

B.1 RALBIC file format

The RALBIC neutral file format [10] is a text file format for storing details of finite element meshes and results. This file format permits many different data fields to be stored in a single file, but for *ralpar* only the element topology and nodal coordinates of the mesh are required.

By convention all RALBIC file names are in upper case and the extension `.MSH` is used. A simple example of the RALBIC neutral file for a two dimensional mesh with just two elements is shown below:

```
$HEAD
 VER_03_87
$CASE
  Ralpar
$NODE
    6    2
    1  0.0000000D+00  0.0000000D+00
    2  0.5000000D+00  0.0000000D+00
    3  0.0000000D+00  0.5000000D+00
    4  0.5000000D+00  0.5000000D+00
    5  0.0000000D+00  0.1000000D+01
    6  0.5000000D+00  0.1000000D+01
$END-NODE
$ELEM
    2    8
    1 QU04           AIR    4    1    2    4    3
    2 QU04           AIR    4    3    4    6    5
$END-ELEM
$END-CASE
$END-HEAD
```

The node section of the file is quite simple and just contains x and y coordinates for each node. The element topology section (`$ELEM`) contains one record for each element. This gives the element number, the element type, the material type, the number of nodes in the element and then the node numbers. The material type field is ignored on input into *ralpar*. On output, an integer is written into this field indicating the subdomain to which this element has been assigned, if the FULLMESH option is set to true. If just the partition data is required, the OUTPUT command will produce a file with the extension ".PAR". For the above mesh split into 2 the file would look like this:

```
$HEAD
 VER_03_87
$CASE
  Ralpar
$ETYP
    2
    1    2    2    1
$END-ETYP
$END-CASE
$END-HEAD
```

The partitioning of each element is given in the section labelled ETYP. The first number here is the number of elements in the mesh. Then each element number is listed along with the partition it has been assigned to. In this case element 1 is in partition 2 and element 2 in partition 1.

To actually create such files from another format, two example programs are provided with the standard release of *ralpar*. The first is `loc2ral.f` which reads a mesh file in a simple format and writes out a corresponding RALBIC neutral file. The source code for this program is provided along with the RALBIC library so that it may be easily modified to read the data according to the preferred format of the user. This will then provide a tool to convert from any desired format to RALBIC format.

A similar program to convert the output RALBIC file from *ralpar* into a local format is also supplied as `ral2loc.f`. This can be modified to write the specific information that will be required for the separate subdomains. The element types that are supported by the RALBIC interface include those listed in Table 1.

Element	No. of nodes	Name	Type No.
Hexahedra	8	BR08	1
Prism	6	WG06	2
Tetrahedra	4	TE04	3
Hexahedra	20	BR20	4
Prism	15	WG15	5
Tetrahedra	10	TE10	6
Triangle	3	TR03	7
Triangle	6	TR06	8
Quadrilateral	4	QU04	9
Quadrilateral	8	QU08	10
Line	2	LI02	11
Line	3	LI03	12
Pyramid	5	PY05	13
Hexahedra	16	BR16	14

Table 1: Some RALBIC element types.

For partitioning the ordering of nodes within an element is not important, even though the RALBIC format does specify the required order. The display command does require the correct ordering of nodes to enable the results to be viewed. The assumed scheme is to number the nodes anti-clockwise. For 3D elements, the lower plane is numbered first, then the subsequent planes, starting at the same point as the first plane. Details of the numbering for more complex elements is given in [10].

B.2 ASCII file format

The ASCII file format is another text format, similar to that used in the RALBIC format, though slightly simpler. This format may be easier to write directly from an existing program since it does not make use of an interface library as is the case with the RALBIC neutral files. For the same two element mesh that was discussed in the previous section, the ASCII format file would look like this:

```
6      2      1       Nodal coordinates
1 0.0000000E+00 0.0000000E+00
2 5.0000000E-01 0.0000000E+00
3 0.0000000E+00 5.0000000E-01
4 5.0000000E-01 5.0000000E-01
```

```
5 0.0000000E+00 1.0000000E+00
6 5.0000000E-01 1.0000000E+00
2      4      Element topology
1  9  4      1      2      4      3
2  9  4      3      4      6      5
```

As before, the nodal coordinates are listed first. The initial line gives the number of nodes, the dimension of the mesh (2 or 3) and a version number of the format (=1). Then the x and y coordinates of each node are given. The exact formats used can be seen in the routine `wrasci.f` in the directory `dataio`. The first line of the element topology section gives the number of elements in the mesh and the maximum number of nodes in an element. this is followed by a list of element descriptions. This consists of the element number, the element type (as in table 1), the number of nodes in the element and finally the node numbers themselves.

If the ASCII format is used in the OUTPUT command then the file is of the form shown here:

```
  2      2      1      Element partition numbers
2    1
```

The three values on the first line are the number of elements, the number of partitions used and a value indicating which method was used. This is followed by the partition number each element has been assigned to. Note that the element numbers are not listed in this section.

C Control of memory allocation

Ralpar is written in standard Fortran 77 as far as possible. As the standard does not allow dynamic memory allocation, most arrays are treated as having fixed size. To allow you to adjust the memory use according to that available on your machine (and size of mesh), dynamic allocation of work arrays is made at the top level. By default, space is allocated for about 30000 nodes/elements. To partition larger meshes you need to set the environment variables `RALPAR_NODES` and/or `RALPAR_ELEMENTS`. For example, if you have 100000 nodes and 120000 elements you could issue the command (csh):

```
setenv RALPAR_ELEMENTS 130000
```

before running *ralpar*. In the Bourne shell the command would be:

```
RALPAR_ELEMENTS=130000
export RALPAR_ELEMENTS
```

Note that if you only set one of nodes or elements, the other defaults to the same value.

The maximum number of nodes in an element defaults to 8. This is sufficient for most simple finite elements. However, if elements with greater number of nodes are needed, such as 20 noded hexahedra, this limit can be increased by;

```
setenv RALPAR_NODES_PER_ELEM 20
```

Conversely, if a mesh only contains tetrahedra for example, setting this parameter to 4 would save some memory.

Of course, some methods require much more memory than other methods and the partitioner allocates enough for all methods by default (assuming “reasonable” limits on connectivity). You can change the amount of workspace allocated by setting the environment variable `RALPAR_MEMORY_SF` if your machine cannot allocate sufficient memory for a given number of nodes/elements. The default value of this is 100, so to half the workspace, you could issue the command (in csh):

```
setenv RALPAR_MEMORY_SF 50
```

Note that some methods may not run with this reduced workspace, and you should only try this if the partitioner will not run because it failed to allocate enough memory for the requested mesh size.

To find out how much of the available workspace is actually being used by each method, you can use the command `information high`. Any subsequent `partition` commands will then also print out some details of how much space was actually used.