Software for mesh partitioning

电磁仿真中心

北京理工大学信息科学技术学院

E-mail：danibarrero@hotmail.com　电话: 13520078012

http://www.cems.bj.cn

Contents

1. Introduction

Graph partitioning is a fundamental problem in many scientific contexts. Algorithms that find a good partitioning of highly unstructured graphs are critical for developing efficient solutions for a wide range of problems in many application areas on both serial and parallel computers. For example, large-scale numerical simulations on parallel computers, such as those based on finite element methods; require the distribution of the finite element mesh to the processors. This distribution must be done so that the number of elements assigned to each processor is the same, and the number of adjacent elements assigned to different processors is minimized.

The goal of the first condition is to balance the computations among the processors. The goal of the second condition is to minimize the communication resulting from the placement of adjacent elements to different processors. Graph partitioning can be used to successfully satisfy these conditions by first modeling the finite element mesh by a graph, and then partitioning it into equal parts.

Software for graph partitioning is widely available. This document contains a brief summary of the most famous ones and a detailed explanation of MeTis Graph Partitioning Software and a Matlab toolbox, justified in the following section the choice of this software.

2. Software options

Among all the algorithms and tools that can be found, in this section there is a brief explanation of the most important ones:

- **Metis** (Karypis and Kumar): Metis is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in Metis are based on the multilevel recursive-bisection, multilevel *k*-way, and multi-constraint partitioning schemes.

- **Party** (Preis): The Party partitioning library serves a variety of different partitioning methods in a very simple and easy way. Instead of implementing the methods directly, the user may take advantage of the ready implemented methods of the library. All implementations increases the performance of the partitioning heuristics. Two kinds of interfaces allow the use as stand-alone tool as well as the inclusion into application codes. The data-structures used as interface to Party are simple and easy to generate.

- **Jostle** (Walshaw): Jostle is a software package designed to partition unstructed meshes (for example, finite element or finite volume meshes) for use on distributed memory parallel computers. It can also be used to repartition and load-balance existing partitions (such as those deriving from adaptive refined meshes). It achieves this by modeling the mesh as an undirected graph and then using state-of-the-art graph partitioning techniques.

- **Scotch** (Pellegrini): The Scotch distribution is a set of programs and libraries which implement the static mapping and sparse matrix reordering algorithm.

- **Chaco** (Bruce Hendrickson) a variety of algorithms for graph partitioning and implemented them into a package. The code is being used to simplify the development of parallel applications, and to ensure that high performance is obtained.

- **Zoltan** (Karen D.): the Zoltan library is a collection of data management services for parallel, unstructured, adaptive, and dynamic applications. It simplifies the load-balancing, data movement, unstructured communication, and memory usage difficulties that arise in dynamic applications such as adaptive finite-element methods, particle methods, and crash simulations. Zoltan's data-structure neutral design also lets a wide range of applications use it without imposing restrictions on application data structures.

Election justification

Since there are not clear advantages between the different algorithm/software and each one has a better performance in different fields, it has been chosen the METIS software because is the one that has a free distribution and it provides high quality partitions in finite element methods domain (aim of the study):

The advantages of METIS compared to other similar packages are the following:

+ **Provides high quality partitions in finite element methods domain**
Experiments on a large number of graphs arising in various domains including finite element methods, linear programming, VLSI, and transportation show that METIS produces partitions that are consistently better than those produced by other widely used algorithms.

+ **Faster than other algorithms**
Experiments on a wide range of graphs have shown that METIS is one to two orders of magnitude faster than other widely used partitioning algorithms.

3. Metis

Metis is a software package for partitioning large irregular graphs, partitioning large meshes, and computing fillreducing orderings of sparse matrices. The algorithms in Metis are based on multilevel graph partitioning. These algorithms reduce the size of the graph by collapsing vertices and edges, partition the smaller graph, and then uncoarsen it to construct a partition for the original graph. These highly tuned algorithms allow METIS to quickly produce high-quality partitions for a large variety of graphs.

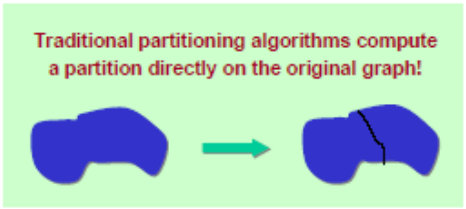
Metis provides a variety of programs that can be used to partition graphs, partition meshes, compute fill-reducing orderings of sparse matrices, as well as programs to convert meshes into graphs appropriate for Metis's graph partitioning programs.

But among all the programs that Metis has, this document just give an explanation of partition meshes programs, since is the aim of the study.

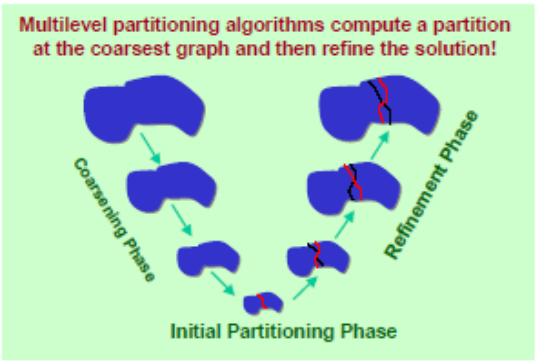
Metis algorithms

The algorithms in METIS are based on multilevel graph partitioning. Traditional graph partitioning algorithms compute a partition of a graph by operating directly on the original graph as illustrated in the next figure (a). These algorithms are often too slow and/or produce poor quality partitions.

Multilevel partitioning algorithms, on the other hand, take a completely different approach. These algorithms, as illustrated in the next figure (b), reduce the size of the graph by collapsing vertices and edges, partition the smaller graph, and then uncoarsen it to construct a partition for the original graph. METIS uses novel approaches to successively reduce the size of the graph as well as to further refine the partition during the uncoarsening phase. During
coarsening, METIS employs algorithms that make it easier to find a high-quality partition at the coarsest graph. During refinement, METIS focuses primarily on the portion of the graph that is close to the partition boundary. These highly tuned algorithms allow METIS to quickly produce high-quality partitions for a large variety of graphs.



(a)



(b)

Mesh partitioning programs

Metis provides two programs *partnmesh* and *partdmesh* for partitioning meshes (*e.g.*, those arising in finite element or finite volume methods) into *k* equal size parts. These programs take as input the element node array of the mesh and compute a partitioning for both its elements and its nodes. Metis currently supports four different types of mesh elements which are triangles, tetrahedra, hexahedra (bricks), and quadrilaterals.

The difference between these two programs is that *partnmesh* converts the mesh into a nodal graph (*i.e.*, each node of the mesh becomes a vertex of the graph), whereas *partdmesh* converts the mesh into a dual graph (*i.e.*, each element becomes a vertex of the graph). In the case of *partnmesh*, the partitioning of the nodal graph is used to derive a partitioning of the elements. In the case of *partdmesh*, the partitioning of the dual graph is used to derive a partitioning of the nodes.

Both of these programs produce partitioning of comparable quality, with *partnmesh* being considerably faster than *partdmesh*. However, in some cases, *partnmesh* may produce partitions that have higher load imbalance than *partdmesh*.

Both partnmesh and partdmesh are invoked by providing two arguments at the command line as follows:

partnmesh *MeshFile Nparts*
partdmesh *MeshFile Nparts*

The first argument *MeshFile*, is the name of the file that stores the mesh (whose format is described in the following section), while the second argument*Nparts*, is the number of partitions that is desired. Both *partnmesh* and *partdmesh* can partition a mesh into an arbitrary number of partitions. Upon successful execution, both programs display statistics regarding the quality of the computed partitioning and the amount of time taken to perform the partitioning.

The actual partitioning is stored in two files named: ***MeshFile.npart.Nparts*** which stores the partitioning of the nodes, and ***MeshFile.epart.Nparts*** which stores the partitioning of the

elements. The format of the partitioning files is described in the following section.

The output of *partnmesh* and *partdmesh* for partitioning a mesh print information about the mesh, such as its name, the number of elements (*#Elements*), the number of nodes (*#Nodes*), and the type of elements (*e.g*.*TET*). Next, they print some information regarding the quality of the partitioning. Specifically, they report the number of edges being cut (*Edge-Cut*) by the partitioning, as well as the balance of the partitioning. For both *partnmesh* and *partdmesh*, the balance is computed with respect to the number of elements. The balance with respect to the number of nodes is not shown, but it is in general similar to the element balance.

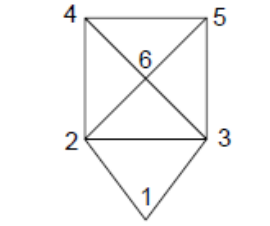
Finally, both *partnmesh* and *partdmesh* show the time that was taken by the various phases of the algorithm.

Input files format

The primary input of the mesh partitioning programs in METIS is the mesh to be partitioned. This mesh is stored in a file in the form of the element node array. A mesh with *n* elements is stored in a plain text file that contains *n* +1. lines. The first line contains information about the size and the type of the mesh, while the remaining *n* lines contain the nodes that make up each element.

The first line contains two integers. The first integer is the number of elements *n* in the mesh. The second integer *etype* is used to denote the type of elements that the mesh is made off. *Etype* can either take the values of 1, 2, 3, or 4, indicating that the mesh consists of either triangles, tetrahedra, hexahedra (bricks), or quadrilaterals, respectively.

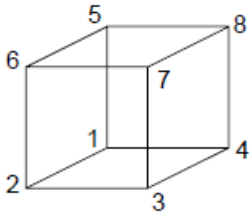
After the first line, the remaining *n* lines store the element node array. In particular for element *i* , line *i* +1 stores the nodes that this element is made off. Depending on *etype*, each line can either have three integers (in the case of triangles), four integers (in the case of tetrahedra and quadrilaterals), or eight integers (in the case of hexahedra). In the case of triangles and tetrahedra, the ordering of the nodes for each element does not matter. However, in the case of hexahedra and quadrilaterals, the nodes for each element should be ordered according to the numbering illustrated in next figure(b). Note that the node numbering starts from 1. Next figure illustrates this format for a small mesh with triangular elements. Note that the *etype* field of the mesh file is set to 1 indicating that the mesh consists of triangular elements.

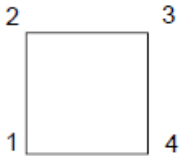


Mesh File:

```
5 1
1 2 3
2 4 6
2 6 3
4 5 6
5 6 3
```

(a) Sample Mesh File





(b) Ordering of nodes

Output File Formats

The partition file of a mesh with *n* vertices consists of *n* lines with a single number per line. The *i* th line of the file contains the partition number that the *i* th vertex belongs to. Partition numbers start from 0 up to the number of partitions minus one.

Mesh Data Structure

All of the mesh partitioning and mesh conversion routines in METISlib take as input the element node array of a mesh. This element node array is stored using an array called elmnts. For a mesh with *n* elements and *k* nodes per element, the size of the elmnts array is *n*k*. Note that since the supported elements in Metis are only triangles, tetrahedra, hexahedra, and quadrilaterals, the possible values for *k* are 3, 4, 8, and 4, respectively.

The element node array of the mesh is stored in elmnts as follows. Assuming that the element numbering starts from 0 (C style), then the *k* nodes that make up element *i* are stored in array elmnts starting at index *i*k* and ending (but not including) index *(i +1)*k*. The ordering of the nodes is not important for triangle and tetrahedra elements. However, in the case of hexahedra, the nodes for each element must be ordered.

The array that describes the element node array of the mesh is defined in METISlib to be of type idxtype, which by default is equivalent to int (*i.e.*, integers).

Mesh partitioning test

86400 elements mesh

Next figure shows the output of partnmesh and partdmesh for partitioning a mesh with tetrahedron elements into 100 parts. From this figure we see that both programs initially print information about the mesh, such as its name, the number of elements (*#Elements*), the number of nodes (*#Nodes*), and the type of elements (*e.g.*, *TET*). Next, they print some information regarding the quality of the partitioning. Specifically, they report the number of edges being cut (*Edge-Cut*) by the partitioning, as well as the balance of the partitioning. For both partnmesh and partdmesh, the balance is computed with respect to the number of elements. The balance with respect to the number of nodes is not shown, but it is in general similar to the element balance. Finally, both partnmesh and partdmesh show the time that was taken by the various phases of the algorithm. All times are in seconds. This time includes the time required both to construct the dual graph and to partition it. As can be seen from this example, partnmesh is considerably faster than partdmesh. This is because of two reasons: the time required to construct the nodal graph is smaller than the time required to construct the dual graph; the nodal graph is smaller than the dual graph.

```
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partdmesh big_mesh.mesh 100
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: big_mesh.mesh, #Elements: 86400, #Nodes: 20449, Etype: TET

Partitioning Dual Graph... ------------------------------------------
  100-way Edge-Cut:   11091, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                          0.130
  Partitioning:                 0.370
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh big_mesh.mesh 100
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: big_mesh.mesh, #Elements: 86400, #Nodes: 20449, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
  100-way Edge-Cut:   22029, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                          0.150
  Partitioning:                 0.170
**********************************************************************
```

2560000 elements mesh

Next figure shows the output of partnmesh and pardmesh for partitioning a big mesh of 2560000 elements with tetrahedron elements into 2 to 20 parts. In this case it is compared

the time that was taken by the various phases of the algorithm using both algorithms. As can be seen the difference is not very big and the time taken doesn't depend on the number of partitions.

```
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 2
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
  2-way Edge-Cut:  233223, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                    2.960
  Partitioning:           59.610
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 4
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
  4-way Edge-Cut:  359716, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                    2.970
  Partitioning:           60.100
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 6
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
  Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
  6-way Edge-Cut:  416717, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                    2.990
```

```
 Partitioning:              62.730
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 8
**********************************************************************
 METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
 Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
 8-way Edge-Cut:  449047, Balance:  1.03

Timing Information --------------------------------------------------
 I/O:                       3.010
 Partitioning:              62.420
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 10
**********************************************************************
 METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
 Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
 10-way Edge-Cut:  472797, Balance:  1.03

Timing Information --------------------------------------------------
 I/O:                       2.950
 Partitioning:              60.420
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 15
**********************************************************************
 METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
 Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

Partitioning Nodal Graph... -----------------------------------------
 15-way Edge-Cut:  512481, Balance:  1.03

Timing Information --------------------------------------------------
 I/O:                       3.070
 Partitioning:              60.910
**********************************************************************
```

```
[Daniel@ZUCHONGZHI metis-4.0]$ ./partnmesh superbig.mesh 20
**********************************************************************
 METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ----------------------------------------------------
 Name: superbig.mesh, #Elements: 2560000, #Nodes: 426465, Etype: TET

 20-way Edge-Cut:  542762, Balance:  1.05

Timing Information --------------------------------------------------
 I/O:                    3.070
 Partitioning:           65.030
**********************************************************************
[Daniel@ZUCHONGZHI metis-4.0]$
```

As a summary the next table shows the time taken for each partition:

partnmesh			
Nparts	Timing I/O	Timing part	Balance
2	2.96	59.61	1.03
4	2.97	60.1	1.03
6	2.99	61.73	1.03
8	3.01	61.87	1.03
10	2.95	62.42	1.03
15	3.07	62.91	1.03
20	3.07	65.03	1.05

partdmesh			
Nparts	Timing I/O	Timing part	Balance
2	2.49	82.37	1.03
4	2.51	83.12	1.03
6	2.59	83.78	1.03
8	2.58	83.4	1.03
10	2.61	84.01	1.03
15	2.61	84.13	1.03
20	2.65	86.8	1.03

As can be seen partdmesh requieres more time to partitoning the mesh for the reasons mentioned before.

The quality of the partition is measured in terms of the edge-cut, the total number of edges between nodes belonging to different partitions, and the balance in the number of nodes assigned to each part.

In this case the upper bound on the imbalance between the weights of the partitions produced is 3%. It means that the number of nodes in any partition does not exceed 1.03 x n/k, where n is the total number of nodes and k is the number of parts. This small load

imbalance may result due to the logk levels of recursive bisection.

Mesh Partitioning Routines

Metis also includes some routines that can be used to partition a mesh. Those routines are PartMeshNodal and PartMeshDual, detailed the parameters in this section.

METIS PartMeshNodal (int *ne, int *nn, idxtype *elmnts, int *etype, int *numflag, int *nparts, int * edgecut, idxtype *epart, idxtype *npart)

Description
This function is used to partition a mesh into *k* equal-size parts. It provides the functionality of the partnmesh program.

Parameters

ne The number of elements in the mesh.

nn The number of nodes in the mesh.

elmnts The element node array storing the mesh as described in Mesh Data Strcuture.

etype Indicates the type of the elements in the mesh. *etype* can take the following values:
1 The elements are triangles.
2 The elements are tetrahedra.
3 The elements are hexahedra (bricks).
4 The elements are quadrilaterals.

numflag Used to indicate which numbering scheme is used for the element node array. *numflag* can take the following two values:
0 C-style numbering is assumed that starts from 0
1 Fortran-style numbering is assumed that starts from 1

nparts The number of parts to partition the mesh.

edgecut Upon successful completion, this variable stores the number of edges that are cut by the partition in the nodal graph.

epart This is a vector of size *ne* that upon successful completion stores the partition vector for the elements of the mesh. The numbering of this vector starts from either 0 or 1, depending on the value of *numflag*.

npart This is a vector of size *nn* that upon successful completion stores the partition vector for the nodes of the mesh. The numbering of this vector starts from either 0 or 1, depending on the value of *numflag*.

METIS PartMeshDual (int *ne, int *nn, idxtype *elmnts, int *etype, int *numflag, int *nparts, int *edgecut, idxtype *epart, idxtype *npart)

Description

This function is used to partition a mesh into *k* equal-size parts. It provides the functionality of the partdmesh program.

Parameters

ne The number of elements in the mesh.

nn The number of nodes in the mesh.

elmnts The element node array storing the mesh as described in Mesh Data Strcuture.

etype Indicates the type of the elements in the mesh. *etype* can take the following values:
1 The elements are triangles.
2 The elements are tetrahedra.
3 The elements are hexahedra (bricks).
4 The elements are quadrilaterals.

numflag Used to indicate which numbering scheme is used for the element node array. *numflag* can take the following two values:
0 C-style numbering is assumed that starts from 0
1 Fortran-style numbering is assumed that starts from 1

nparts The number of parts to partition the mesh.

edgecut Upon successful completion, this variable stores the number of edges that are cut by the partition in the dual graph.

epart This is a vector of size *ne* that upon successful completion stores the partition vector for the elements of the mesh. The numbering of this vector starts from either 0 or 1, depending on the value of *numflag*.

npart This is a vector of size *nn* that upon successful completion stores the partition vector for the nodes of the mesh. The numbering of this vector starts from either 0 or 1, depending on the value of *numflag*.

4. Matlab Mesh Partitioning and Graph Separator Toolbox

This toolbox contains Matlab code for several graph and mesh partitioning methods, including geometric, spectral, geometric spectral, and coordinate bisection. It also has routines to generate recursive multiway partitions, vertex separators, and nested dissection orderings; and it has some sample meshes and mesh generators.

The toolbox contains a Matlab interface to Karypis et al.'s Metis partitioning package, using Robert Bridson's "metismex" code. This interface is going to be used to plot the meshes partitioning.

Toolbox Contents

Partitioning methods.

geopart	- Geometric.
specpart	- Spectral.
gspart	- Geometric spectral.
coordpart	- Coordinate bisection.
Inertpart	- Inertial bisection.
metispart	- Multilevel method from Metis.
metismex	- Interface to more options of Metis.

Multiway partitions.

dice	- Use any 2-way partitioner to get a multiway partition.
geodice	- Recursive geometric partitioning.
specdice	- Recursive spectral partitioning.
gsdice	- Recursive geometric spectral partitioning.
metisdice	- Multiway partitioning from Metis.

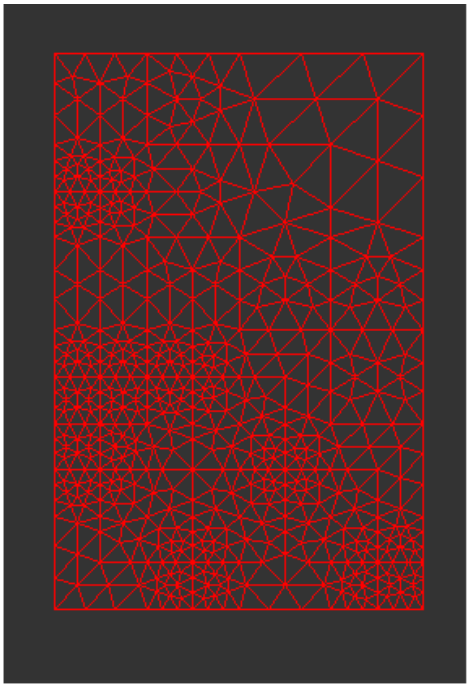
Visualization and graphics.

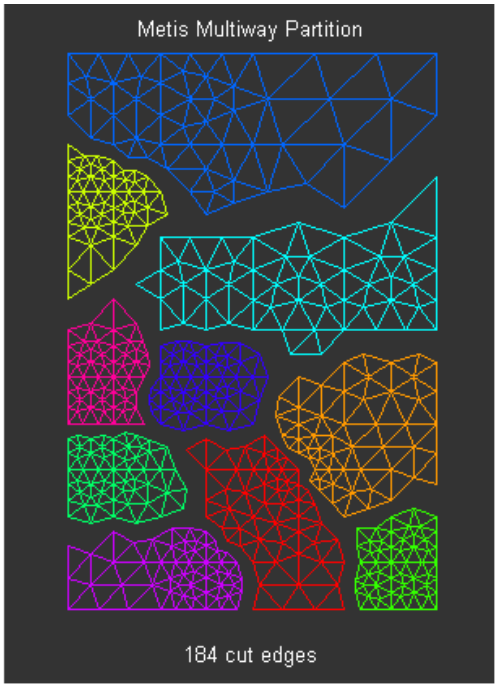
gplotpart	- Draw a 2-way partition.
gplotmap	- Draw a multiway partition.
highlight	- Draw a mesh with some vertices highlighted.
gplotg	- Draw a 2D or 3D mesh
etreeplotg	- Draw an elimination tree
spypart	- Matrix spy plot with partition boundaries.
dmspy	- Spy plot of matrix in block triangular form.

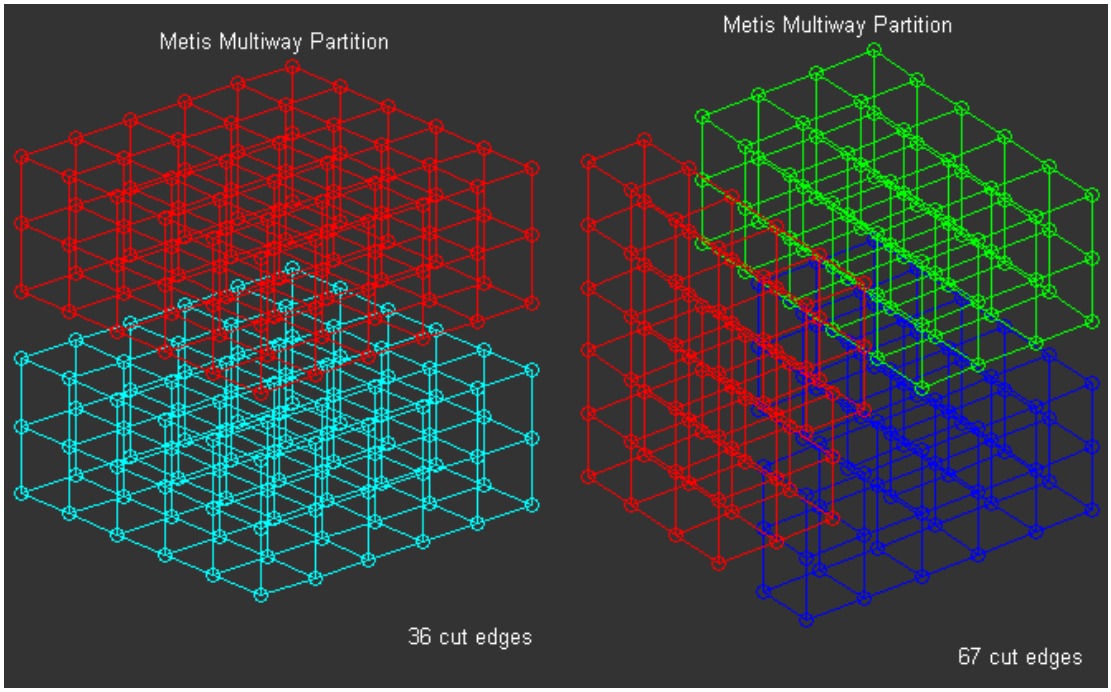
Matlab graph/mesh partitioning toolbox tests

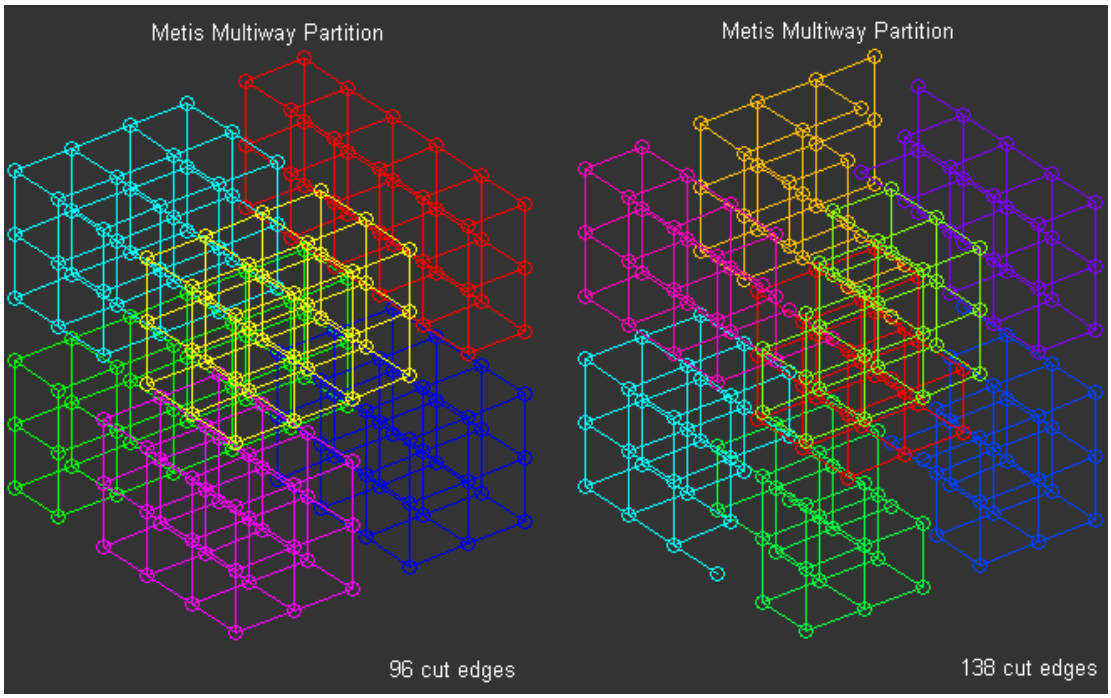
In this section a mesh has been generated with the Matlab toolbox as example. It has been used the metisdice function for partitioning this mesh since it uses the Metis partitioning routine described before.

The mesh used is a 2D finite-element mesh with 547 nodes, and it has been partitioned into 10 parts, as can be seen in the following pictures obtaining 184 cut edges.






Metis Multiway Partition
36 cut edges
Metis Multiway Partition
67 cut edges


Metis Multiway Partition
96 cut edges
Metis Multiway Partition
138 cut edges

5. Bibliography

C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview. In F. Magoules, editor, *Mesh Partitioning Techniques and Domain Decomposition Techniques*. Civil-Comp Ltd., 2007.

G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998

John R. Gilbert, Gary L. Miller, and Shang-Hua Teng. Geometric mesh partitioning: Implementation and experiments. SIAM J. Scientific Computing 19:2091-2110, 1998.

Tony F. Chan, John R. Gilbert, and Shang-Hua Teng. Geometric spectral partitioning. Xerox PARC Technical Report CSL-94-15, 1995.

Bruce Hendrickson and Robert Leland. The Chaco user's guide, version 2.0. Sandia National Laboratories Technical Report SAND94-2692, 1994.

George Karypis et al. METIS, Serial graph partitioning, version 4.0.1, November 1998. http://www.cs.umn.edu/~karypis/metis

Robert Bridson. A MATLAB CMEX interface to the Metis library. http://www.stanford.edu/~rbridson/download/metismex.c