Implementation parallel computing for finite element method of Poisson equations

Jichao Zhao *

March 6, 2005

Abstract

In this project, we talk about the issue of implementation parallel computing for finite element method of poisson equation. We show how to use the GRUMMP to generate mesh data, and then use Metis to partition the mesh data into several pieces which are the input files for parallel finite element method solver. In Section(1), we introduce the two dimension of poisson equation and finite element method for it; We give the parallel algorithm for finite element method in Section(2); In Section(3), we use several examples to show how to use the two software GRUMMP and Metis; Then we use GRUMMP and Metis to generate the mesh for several examples and solve the poisson equation on these irregular domains, all the results are given in Section(4) or in the Appendix; Conclusion is drawn in the last section.

1 Introduction

In this project we consider how to solve the following two dimensions of poisson equations using parallel computing:

$$-(\frac{\partial^2\Phi(x,y)}{\partial x^2}+\frac{\partial^2\Phi(x,y)}{\partial y^2})=f(x,y),(x,y)\in\Omega, \tag{1}$$

To make life easier, we only deal with Dirichlet boundary condition for eqn(1) in my project, i.e.

$$u|_{\partial\Omega}=g_0(x,y), \tag{2}$$

In finite element course, we have applied finite element method on the two dimensions of poisson equations, we seek an approximate solution of the form:

$$\Psi_h=\Sigma_{j=1}^{M}v_jN_j(x,y), \tag{3}$$

*AM562, a course project, supervised by Weijiu Liu

where $N_j(x,y)$ are the usual piecewise linear basis functions on one triangulation partition Γ. And Ψ_h satisfies:

$$\int_\Omega (\frac{\partial \Psi_h(x,y)}{\partial x}\frac{\partial N_j(x,y)}{\partial x} + \frac{\partial \Psi_h(x,y)}{\partial y}\frac{\partial N_j(x,y)}{\partial y})dxdy$$
$$= \int_\Omega f(x,y)N_j(x,y)dxdy, j = 1..M. \tag{4}$$

Then we calculate the local element stiffness matrix and vector, assemble the local element stiffness matrix and vector to obtain the global stiffness matrix K and Q, then solve the following system of linear equation for v_j:

$$KV = Q, \tag{5}$$

where

$$V := \begin{pmatrix} v_1 \\ \vdots \\ v_M \end{pmatrix},$$

and K and Q are the global stiffness matrix and global element force vector respectively.

Through this project, we test $f(x,y) = (a^2+b^2)\sin(ax)\cos(bx)$, and $a = 2, b = 3$ in the equation (1). And it is easy to know that for this example, the accurate solution is $\Phi_{true} = \sin(ax)\cos(bx)$. To measure the accuracy of finite element method, we use the max absolution error:

$$Error = max_j \mid \Phi(x_j, y_j) - v_j(x_j, y_j) \mid, \tag{6}$$

2 Parallel algorithm for finite element method

In this project, we use Message Passing Interface(MPI) library to implementation parallel computing.

Suppose that we have p processes together solve the poisson equations, first we need to partition the triangulation Γ evenly into $\Gamma_1, ..., \Gamma_p($ we will talk about on this issue in next section), then solve the poisson equation on the subpartition Γ_j on jth process respective. Note when we compute the local stiffness matrix for the interior vertices in subpartition Γ_j, we can obtain all the information it needs on this subpartition Γ_j. But for the each of the vertices lying on the partition boundary, it may need some information from other subpartition. So we write the global stiffness matrix eqn(5) into the following block matrix form eqn which separate the interior vertices from boundary vertices :

$$K_p\Psi_p = Q_p, \tag{7}$$

where

$$K_p := \begin{pmatrix} A_1 & 0 & \cdots & 0 & B_1 \\ 0 & A_2 & 0 & 0 & B_2 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & A_p & B_p \\ B_1' & B_2' & \cdots & B_p' & A_s \end{pmatrix}$$

and

$$\Psi_p := \begin{pmatrix} v_1 \\ \vdots \\ v_p \\ v_s \end{pmatrix} \text{ and } Q_p := \begin{pmatrix} q_1 \\ \vdots \\ q_p \\ q_s \end{pmatrix},$$

the A_j, B_j, v_j, and q_j $(j = 1..p)$ correspond to the interior vertices on the subpartition Γ_j respectively, and A_j, B_j, v_j, and q_j correspond to the boundary vertices on two subpartitions. Then we solve the eqn (7) stored in a distributed manner by the parallel conjugate gradient algorithm.

3 Implementation mesh generation

In this part, we introduce two computer grograms which are very good at mesh generation[3],[2]. First we briefly show how GRUMMP works. For example, we have the following Fig.(1) of the domain:

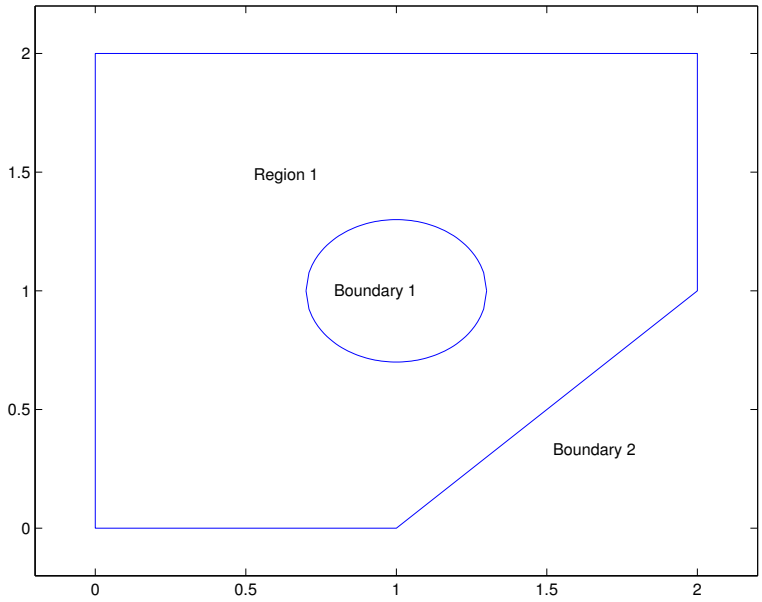

Figure 1: Sample domain 1

And our input(simple2.bdry) is very simple:

8 8
2 1
2 2
1 2
0 2
0 1
0 0
1 0
1 1
polyline r 1 b 2 2 0 1
polyline r 1 b 2 2 1 2
polyline r 1 b 2 2 2 3

polyline r 1 b 2 2 3 4
polyline r 1 b 2 2 4 5
polyline r 1 b 2 2 5 6
polyline r 1 b 2 2 6 0
circle b 1 r 1 0.3 7

we just brief show how to set this input file, please refer to the manual of GRUMMP[3]for details. On the first line is the number of vertices and patches; From line 2 to 9 is the coordinates of these eight vertices; from line 10 to 16 describes 7 polylines repectively, for example, one line 10, it means that this polyline has the region 1 on its left side, this polyline is part of right side boundary 2, and this polyline has two points which connects vertice 0 and 1; On the last line of input file, it defines a circle, whose centered at the location of vertice 8 and radius is 0.3.

Using the command under GRUMMP:

./tri -i simple2

then, we get one output file(please refer to Appendix A). And the Fig.(2) which is plotted by matlab codes in Appendix B for the triangle partition. GRUMMP is very powerful software, which does almost everything for us. In the Appendix C, we give an more complex example by GRUMMP.

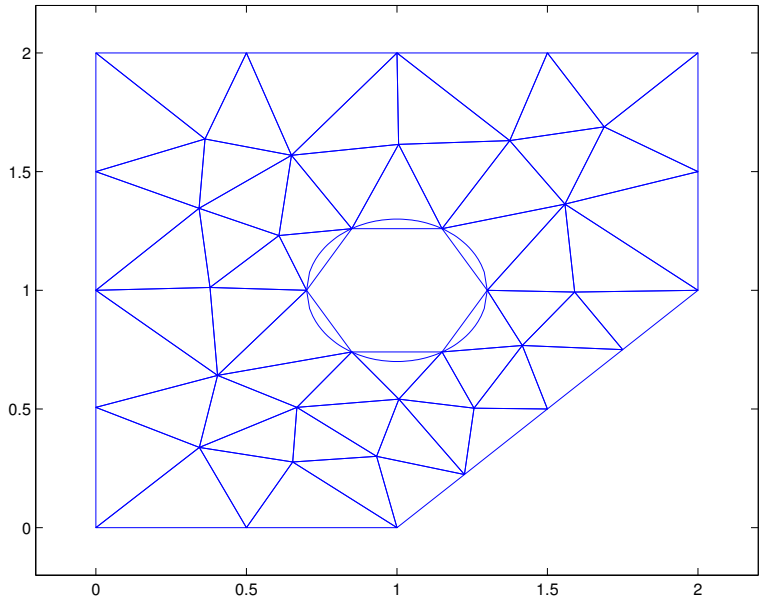

Figure 2: Triangle partition by GRUMMP

Next we introduce the program METIS. "METIS is a software package for partitioning large irregular graphs, partitioning large meshes, and computing fillreducing orderings of sparse matrices." METIS is very powerful software and provides many executable programs, we mainly use three of them here, the "mesh2nodal, mesh2dual and partdmesh".

In METIS' manual, it explains the two programs in a very clear way: "mesh2nodal and mesh2dual for converting a mesh into the graph format used by METIS. In particular, mesh2nodal converts the element node array of a mesh into a nodal graph; i.e., each node of the mesh corresponds to a vertex

in the graph and two vertices are connected by an edge if the corresponding nodes are connected by lines in the mesh. Similarly, mesh2dual converts the element node array of a mesh into a dual graph; i.e., each element of the mesh corresponds to a vertex in the graph and two vertices are connected if the corresponding elements in the mesh share a face. These mesh-to-graph conversion programs support meshes with triangular, tetrahedra, and hexahedra (bricks) elements. Both mesh2nodal and mesh2dual are invoked by providing one argument at the command line as follows: mesh2nodal MeshFile mesh2dual MeshFile The only argument of these programs MeshFile, is the name of the file that stores the mesh. Upon successful execution, both programs display information about the generated graphs, and the amount of time taken to perform the conversion. The actual graph is stored in a file named: MeshFile.ngraph in the case of mesh2nodal and MeshFile.dgraph in the case of mesh2dual.

While partdmesh for partitioning meshes (e.g., those arising in finite element or finite volume methods) into k equal size parts, it converts the mesh into a dual graph (i.e., each element becomes a vertex of the graph, and invoked by providing two arguments at the command line as follows: partdmesh MeshFile Nparts."

And I will give an exmaple to explain how to use it. See the Fig.(3)

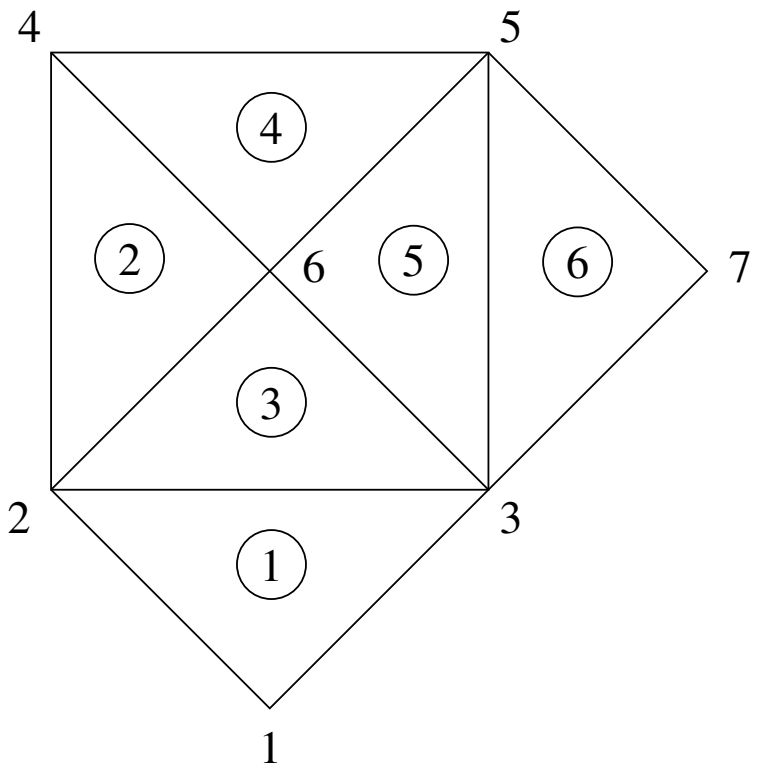

Figure 3: Sample domain 2

The input mesh file is(inputMesh.txt):

61
123
246
236
465
365
357

On he first line: the first integer is the number of elements in the mesh; the second integer is the Etype of partition, value of 1 means that is triangle partition. Use the command:

./mesh2nodal inputMesh.txt

return some information on the screen and the output file(inputMesh.txt.ngraph):

7 12
2 3
1 3 4 6
1 2 6 5 7
2 6 5
4 6 3 7
2 4 3 5
3 5

On the first line, it means that has 7 vertices and 12 lines altogether. The jth $(j > 1)$ line lists all $j - 1$ vertice's neighbors.

./mesh2dual inputMesh.txt

return some information on the screen and the output file(inputMesh.txt.dgraph):

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

which means that the dual graph has 6 element and 6 lines, and The jth $(j > 1)$ line lists all $j - 1$ vertice's neighbors of dual graph.

./partdmesh inputMesh.txt 2

return some information on the screen and the output file(inputMesh.txt.epart.2):

1
0
0
0
1
1

6

It means that subpartition 1 includes element 1, 5 and 6, while subpartition 1 includes element 2, 3 and 4. This file will be used for partition the mesh evenly for parallel computing.

4 Implementation parallel computing with sharcent

In [5], it gives a parallel finite element software using MPI. I Modify a little for our poisson problem.

In this part, we will use the mesh data generated by section (3), then use the codes in Appendix D to convert these data into the input files for parallel computing. We use the deeppruple of charcnet to implementation our program, and the command is as the follows:

mpicc -o fem.exe fem.c
bsub -q devel -n 4 -o outPutFile prun ./fem.exe -lm

For the our Sample domain 2, we use two processes and get the following results:

0.000000 2.732000 1.110653 0 1.110653
the max error is 0.000000 on partition 0 part
0.000000 2.732000 1.110653 1 1.110653
the max error is 0.000000 on partition 1 part
the max error is 0.000000

And for our Sample domain 1, we still use two processes to compute. We make some codes see Appendix D to transfer the data generated by GRUMMP into the input files of Metis and the Parallel FEM solver. And we get the following results:

ProcID = 0, IntNodes = 8, IBNodes = 2
ProcID = 1, IntNodes = 8, IBNodes = 2
Gamma = 17.191345
Gamma = 1.500610 (K = 0)
Gamma = 0.175526 (K = 1)
Gamma = 0.024888 (K = 2)
Gamma = 0.001797 (K = 3)
Gamma = 0.000174 (K = 4)
Gamma = 0.000019 (K = 5)
Gamma = 0.000003 (K = 6)
Gamma = 0.000000 (K = 7)
Gamma = 0.000000 (K = 8)
Gamma = 0.000000 (K = 9)
0.654000 0.277000 0.635924 0 0.015068
0.932600 0.300600 0.552167 0 0.041348
0.343900 0.338500 0.331203 0 0.003481
0.667700 0.506800 0.050245 0 0.001260
0.404100 0.641800 -0.290166 0 0.039111
1.006600 0.541800 -0.039003 0 0.010319
1.416300 0.767700 -0.190380 0 0.012936

1.256000 0.503600 0.025303 0 0.010002
1.590400 0.992700 0.046609 0 0.007934
1.558000 1.362900 -0.015733 0 0.000787
the max error is 0.041348 on partition 0 part
0.649800 1.568900 0.049485 1 0.054966
0.342900 1.345600 -0.371696 1 0.024338
0.608300 1.230600 -0.752705 1 0.046798
0.362500 1.637500 0.164523 1 0.032706
1.005700 1.614700 0.082982 1 0.035806
1.374900 1.630600 0.066590 1 0.001550
1.688400 1.688400 -0.081414 1 0.000889
1.558000 1.362900 -0.015733 1 0.000787
0.379300 1.012100 -0.618675 1 0.065421
0.404100 0.641800 -0.290166 1 0.039111
the max error is 0.065421 on partition 1 part
the max error is 0.065421
Resource usage summary:
CPU time : 0.61 sec.
Max Memory : 24 MB

To show parallel finite element works very well, we test another mesh data generated by GRUMMP, please see the Appendix C for the graph of mesh. And the results(use four processes):

ProcID = 0, IntNodes = 404, IBNodes = 18
ProcID = 1, IntNodes = 404, IBNodes = 42
ProcID = 2, IntNodes = 401, IBNodes = 45
ProcID = 3, IntNodes = 367, IBNodes = 21
the max error is 0.021664 on partition 0 part
the max error is 0.022051 on partition 1 part
the max error is 0.022270
the max error is 0.020357 on partition 3 part
the max error is 0.022270 on partition 2 part
Resource usage summary:
CPU time : 5.00 sec.
Max Memory : 54 MB

It seems that finite element method for this case can obtain the error $O(.01)$.

5 Conclusions

In our project, we use GRUMMP and Metis to help us for mesh generation and partition the region, You can see the two softwares are very powerful. Then we make some codes convert these mesh data into the input files for parallel computing solver. you can see the results by parallel finite element method are accurate and fast.

Appendix A: Output file of (1) by GRUMMP

```
40 58 22
```

```
2 1
2 2
1 2
0 2
0 1
0 0
1 0
1.3 1
1.15 1.25981
0.85 1.25981
0.7 1
0.85 0.740192
1.15 0.740192
1.5 0.5
1.00573 1.61471
1.22433 0.224331
0 0.506763
0 1.5
2 1.5
1.75 0.75
1.41634 0.76772
1.5 2
0.5 2
1.25603 0.503607
0.932555 0.300553
0.40407 0.641782
0.342897 1.34559
1.55797 1.36293
1.59039 0.992697
1.00663 0.54178
0.5 0
0.667693 0.506752
0.379263 1.01214
0.608264 1.23062
0.3625 1.6375
1.68835 1.68835
1.37492 1.6306
0.343856 0.338505
0.653979 0.276978
0.649828 1.56886
39 26 33  1
39 34 26  1
39 22 34  1
39 2 22  1
39 14 2  1
39 9 14  1
39 33 9  1
38 6 24  1
38 30 6  1
38 37 30  1
```

```
38 31 37  1
38 24 31  1
37 31 25  1
37 5 30  1
14 9 8  1
37 16 5  1
37 25 16  1
36 2 14  1
36 21 2  1
36 35 21  1
36 27 35  1
36 8 27  1
36 14 8  1
35 1 21  1
35 18 1  1
35 27 18  1
34 3 17  1
34 22 3  1
34 17 26  1
33 32 10  1
33 26 32  1
33 10 9  1
32 4 25  1
32 26 4  1
32 25 10  1
31 29 11  1
20 7 12  1
31 24 29  1
20 13 19  1
31 11 25  1
29 15 23  1
29 24 15  1
29 12 11  1
29 23 12  1
28 7 20  1
28 27 7  1
28 0 27  1
28 19 0  1
23 15 13  1
23 13 20  1
23 20 12  1
28 20 19  1
27 0 18  1
24 6 15  1
27 8 7  1
26 17 4  1
25 4 16  1
25 11 10  1
7 12 1
12 11 1
```

```
11 10 1
10 9 1
0 18 2
1 21 2
2 22 2
3 17 2
4 16 2
5 30 2
6 15 2
8 7 1
9 8 1
13 19 2
15 13 2
16 5 2
17 4 2
18 1 2
19 0 2
21 2 2
22 3 2
30 6 2
```

Appendix B: Plot the mesh generated by GRUMMP

my matlab function:

```
load simple2_2.m   -ASCII
load elements_2.m  -ASCII

NVertice = simple2_2(1,1);
NElement = simple2_2(1,2);

for i = 1:NVertice;
     cordinateX(i) =  simple2_2(1 + i,1);
     cordinateY(i) =  simple2_2(1 + i,2);
end;

for i = 1:NElement;
     elementNumber1(i) =  elements_2(i,1) + 1;
     elementNumber2(i) =  elements_2(i,2) + 1;
     elementNumber3(i) =  elements_2(i,3) + 1;
end;

%%   begin to plot the partitions

     for i = 1:NElement;
          x1 = cordinateX(elementNumber1(i));
          x2 = cordinateX(elementNumber2(i));
          x3 = cordinateX(elementNumber3(i));
          y1 = cordinateY(elementNumber1(i));
```

```
          y2 = cordinateY(elementNumber2(i));
          y3 = cordinateY(elementNumber3(i));

          plot([x1,x2,x3,x1],[y1,y2,y3,y1])
          hold on;
     end

%%   plot circle
     %x = 0.7:.01:1.3;
     %plot( x,1 + sqrt(0.3^2 - (x - 1).^2))
     %hold on
     %plot( x,1 - sqrt(0.3^2 - (x - 1).^2))

     xlim([-0.2,10.2]);
     ylim([-0.2,6.2]);
```

Appendix C: a complex example by GRUMMP

Input file:

```
18 18
0 0
2 0
2 1
4 1
4 0
6 0
6 1
8 1
8 0
10 0
10 1
8 4
6 4
6 6
4 6
4 4
2 4
0 1
polyline r 1 b 1 2 0 1
polyline r 1 b 1 2 1 2
polyline r 1 b 1 2 2 3
polyline r 1 b 1 2 3 4
polyline r 1 b 1 2 4 5
polyline r 1 b 1 2 5 6
polyline r 1 b 1 2 6 7
polyline r 1 b 1 2 7 8
polyline r 1 b 1 2 8 9
polyline r 1 b 1 2 9 10
```

```
polyline r 1 b 1 2 10 11
polyline r 1 b 1 2 11 12
polyline r 1 b 1 2 12 13
polyline r 1 b 1 2 13 14
polyline r 1 b 1 2 14 15
polyline r 1 b 1 2 15 16
polyline r 1 b 1 2 16 17
polyline r 1 b 1 2 17 0
```

Output file:

```
50 71 27
0 0
2 0
2 1
4 1
4 0
6 0
6 1
8 1
8 0
10 0
10 1
8 4
6 4
6 6
4 6
4 4
2 4
0 1
1 0
3 2.64211
5 3.00003
0.94033 2.41049
5 5
7 2.64211
9 0
5 0
9.03045 2.45433
1.5 0.5
0.5 0.5
3 1
4.14464 2.08333
5.04922 1.73901
2.04482 2.08333
5.85535 2.08333
7 1
9.55788 0.442124
8.5 0.5
5.54416 0.45584
4.45584 0.45584
```

```
7.95518 2.08333
0.407933 1.6119
2.55751 1.51554
9.6684 1.49739
4.9918 0.868825
7.44249 1.51553
1.08939 1.10915
3.33904 1.7711
8.6194 1.36362
6.66096 1.7711
9.14887 0.891462
49 42 47  1
49 10 42  1
49 35 10  1
49 24 35  1
49 36 24  1
49 47 36  1
48 34 44  1
48 6 34  1
48 33 6  1
48 23 33  1
48 44 23  1
47 39 7  1
47 26 39  1
47 42 26  1
47 7 36  1
46 19 41  1
46 30 19  1
46 3 30  1
46 29 3  1
46 41 29  1
45 17 28  1
45 40 17  1
19 15 16  1
20 12 15  1
45 21 40  1
45 32 21  1
45 2 32  1
45 27 2  1
45 28 27  1
44 39 23  1
44 7 39  1
22 13 14  1
22 14 15  1
22 15 12  1
22 12 13  1
23 11 12  1
44 34 7  1
43 3 38  1
43 31 3  1
```

```
43 6 31  1
43 37 6  1
43 25 37  1
43 38 25  1
41 19 32  1
41 2 29  1
41 32 2  1
39 11 23  1
39 26 11  1
38 4 25  1
27 18 1  1
27 1 2  1
38 3 4  1
37 5 6  1
37 25 5  1
28 17 0  1
28 0 18  1
28 18 27  1
36 8 24  1
36 7 8  1
35 9 10  1
35 24 9  1
33 31 6  1
30 20 15  1
30 15 19  1
33 20 31  1
33 12 20  1
33 23 12  1
31 20 30  1
31 30 3  1
32 16 21  1
32 19 16  1
17 0 1
16 21 1
15 16 1
14 15 1
0 18 1
1 2 1
2 29 1
3 4 1
4 25 1
5 6 1
6 34 1
7 8 1
8 24 1
9 10 1
10 42 1
11 12 1
12 13 1
13 14 1
```

```
18 1 1
21 40 1
24 9 1
25 5 1
26 11 1
29 3 1
34 7 1
40 17 1
42 26 1
```

graph of it:

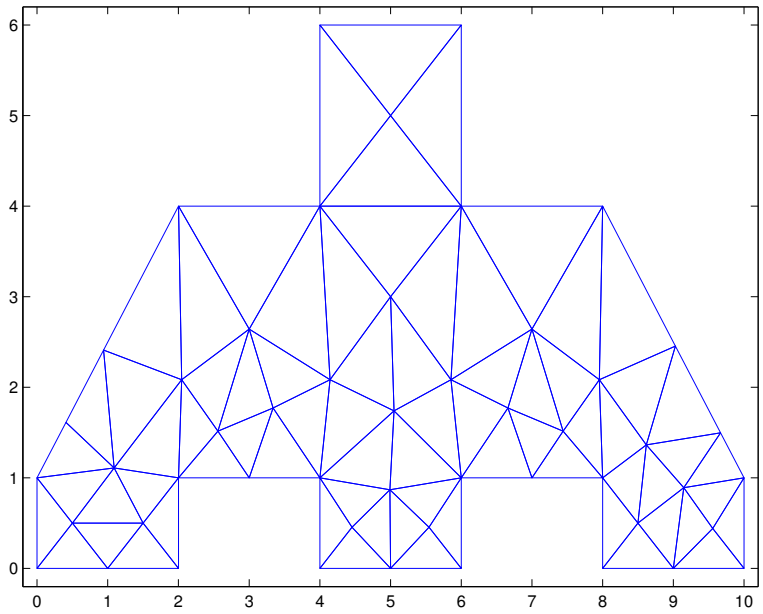

Figure 4: complex example by GRUMMP

Appendix D: Prepare data for parallel computing

my matlab function:

```
%%%  this worksheet is to convert data generated
%%%  by GRUMMP to parallel fem solver
%%%  By Jichao, March 6, 2005

%%   for the file mesh3.vtx

     clear all
     load mesh3input_ptn   -ASCII
     load mesh3input_vtx   -ASCII
     NBoundaryLine = mesh3input_ptn(1,1);
     NVertice = mesh3input_vtx(1,1);
```

```
verticeData = zeros(1 + NVertice,3);
verticeData(1,1) =  NVertice;

counter = 0;

for i = 1:NVertice;
     d = i - 1;

     for j = 1:NBoundaryLine;

          if (d == mesh3input_ptn(j + 1,1)),
               verticeData(i + 1,1) = - round(mesh3input_ptn(j + 1,3));
          end;

          if (d == mesh3input_ptn(j + 1,2)),
               verticeData(i + 1,1) = - round(mesh3input_ptn(j + 1,3));
          end;
     end;

     if (verticeData(i + 1,1) >= 0)
          verticeData(i + 1,1) = round(counter);
          counter = counter + 1;
     end;
     verticeData(i + 1,2) = mesh3input_vtx(i + 1,1);
     verticeData(i + 1,3) = mesh3input_vtx(i + 1,2);

end;

verticeData

save mesh3.vtx verticeData -ASCII

%%   for the file mesh3.el
     load mesh3input_el   -ASCII
     NElement = mesh3input_el(1,1)
     elementData(1,1) = NElement;
     for i = 1:NElement;
          elementData(i + 1,1:3) =  mesh3input_el(i + 1,1:3);
     end;

     elementData

     save mesh3.el elementData -ASCII


%%   for the file mehs3.ptn to be used by Metis

     load mesh3input_el   -ASCII

     NElement = mesh3input_el(1,1);
```

```
elementData(1,1) = NElement;
elementData(1,2) = 1; %% which means that is triangle partition
for i = 1:NElement;
     elementData(i + 1,1:3) =  round(mesh3input_el(i + 1,1:3) + 1);
end;

elementData

save mesh3ForMetis elementData -ASCII
```

References

[1] G. Karypis, METIS, http://www-users.cs.umn.edu/ karypis/metis/metis /download.html.

[2] E.G. Thompson, "Introduction to the Finite Element Method: Theory, Programming and Applications", Wiley, 2005.

[3] C. Ollivier-Gooch, GRUMMP, http://tetra.mech.ubc.ca/GRUMMP /index.html.

[4] P.K. Jimack and N. Touheed, " A General Introduction to Parallel Programming in MPI", http://www.scs.leeds.ac.uk/cpde/papers/paper1. ps.gz.

[5] P.K. Jimack and N. Touheed, " Developing parallel finite element software using MPI", http://www.scs.leeds.ac.uk/cpde/papers/paper2.ps. gz.

[6] W. Liu, Finite element method course's handout.