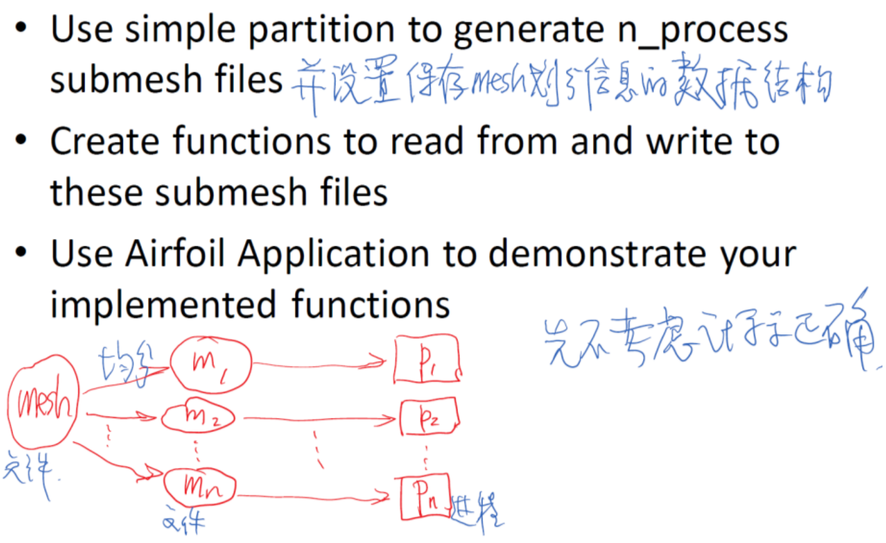
Mesh12 Distributed Mesh Files

19335015 陈恩婷

1. 项目目的

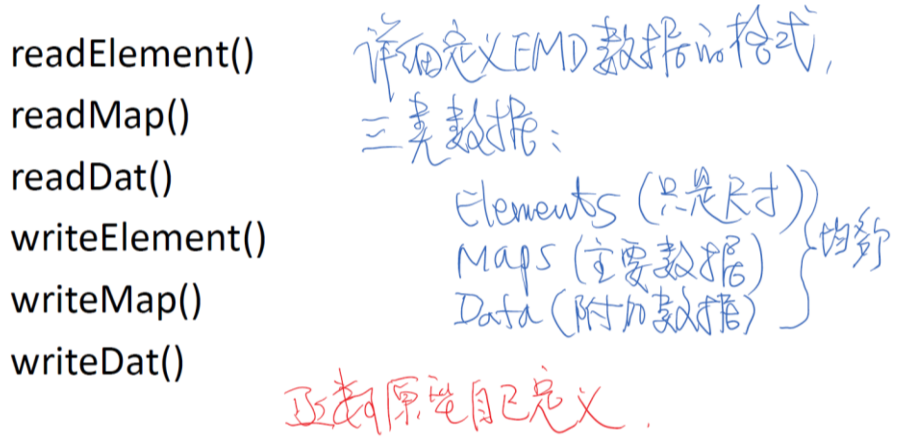
参考课件和项目代码中的提示，用简单的划分方式，实现对划分后的分布式 Mesh 从文件的读取与写入文件。相关课件截图如下：



- Use simple partition to generate n_process submesh files 并设置保存mesh划分信息的数据结构
- Create functions to read from and write to these submesh files
- Use Airfoil Application to demonstrate your implemented functions

先不考虑计算正确

本人学号是 5 结尾，所以实现的是对 ASCII EMD 文件的读取和写入：



readElement()
readMap()
readDat()
writeElement()
writeMap()
writeDat()

详细定义EMD数据的格式，
三类数据：
Elements（只是尺寸）
Maps（主要数据）
Data（附加数据）
均分

取数原样自己定义.

2. 为实现目的存在的各种技术问题

理解 mesh 的实现方法

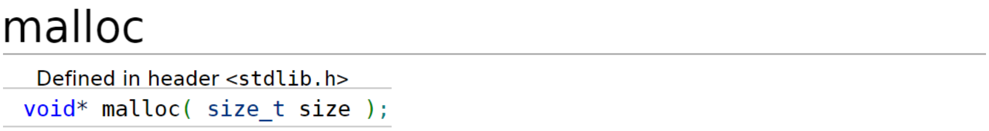
熟练掌握 MPI 并行编程模型

理解并实现 Mesh 的简单划分，与 ASCII EMD 的文件读写

3. 用什么算法、数据结构、语言机制解决这些问题

数据结构：Mesh 的结构主要由 Elements，Maps 和 Data 等结构体组成

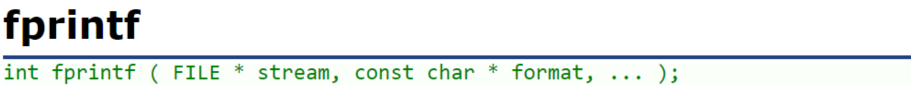
语言机制：主要使用 malloc、fprintf 和 fscanf 等库函数。



malloc

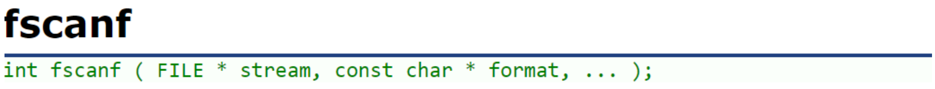
Defined in header <stdlib.h>

```
void* malloc( size_t size );
```



fprintf

```
int fprintf ( FILE * stream, const char * format, ... );
```



fscanf

```
int fscanf ( FILE * stream, const char * format, ... );
```

主要用法和 printf、scanf 类似。

算法：用 MPI_Scatter 将全局 Mesh 划分到各个进程，再在各进程中遍历 Mesh 中的所有结构体和他们所存储的数据，用 fprintf 一一写到文件。读取 Mesh 时用相同方式遍历文件并用 fscanf 读出。

4. 对应的程序框架和实现代码

1. ReadMeshFromEMDAscii 和 WriteMeshToEMDAscii

   主要就是依次写或者读文件里的数据，实现如下：

```
bool Mesh::ReadMeshFromEMDAscii(const char* fileName)
{     /////////////////////////////
      FILE* fp;
      if ((fp = fopen(fileName, "rb")) == NULL) {
          printf("can't open file\n");
          return 0;
      }

      if (element_list_index) {
          for (int i = 0; i < element_list_index; i++) {
              free(element_list[i]);
          }
          free(element_list);
      }
      if (map_list_index) {
          for (int i = 0; i < map_list_index; i++) {
              free(map_list[i]->map);
              //free(map_list[i]);
          }
          free(map_list);
      }
      if (dat_list_index) {
          for (int i = 0; i < dat_list_index; i++) {
              free(dat_list[i]->data);
              free(dat_list[i]);
          }
          free(dat_list);
      }
      readHeader(fp);
      element_list = (Elements*)malloc(sizeof(Elements) * element_list_size);
      map_list = (Map*)malloc(sizeof(Map) * map_list_size);
      dat_list = (Data*)malloc(sizeof(Data) * dat_list_size);
      readElements(fp);
      readMaps(fp);
      readData(fp);

      fclose(fp);
      return true;
    }

    bool Mesh::WriteMeshToEMDAscii(const char* fileName)
{    //////////////////////////
      printf("writing in grid \n");
      FILE* fp;
      if ((fp = fopen(fileName, "wb")) == NULL) {
          printf("can't open file\n");
          return 0;
      }

      writeHeader(fp);
      writeElements(fp);
      writeMaps(fp);
      writeData(fp);
      fclose(fp);
      return true;
    }

    bool Mesh::writeHeader(FILE * fp) {   //////////////////////
      // using fprintf
      fprintf(fp, "%d %d %d %d %d %d\n", element_list_size, map_list_size,
dat_list_size,
          element_list_index, map_list_index, dat_list_index);
      return 1;
    }

    bool Mesh::writeElements(FILE * fp) {   ////////////////////
      // using fprintf
      for ( int i = 0; i < element_list_index; i++ ){
          fprintf(fp, "%d %d %s\n", element_list[i]->index,
element_list[i]->size, element_list[i]->name);
      }
      return 1;
    }

    bool Mesh::writeMaps(FILE * fp) {    /////////////////////////
      // using fprintf
      for (int i = 0; i < map_list_index; i++) {
          fprintf(fp, "%d %d %d %d %s\n", map_list[i]->index,
map_list[i]->from->index,
              map_list[i]->to->index, map_list[i]->dim, map_list[i]->name);
          for (int j = 0; j < map_list[i]->from->size; j++) {
              for (int k = 0; k < map_list[i]->dim; k++) {
                  fprintf(fp, " %d", map_list[i]->map[j * map_list[i]->dim +
k]);
              }
              fprintf(fp, "\n");
          }
          fprintf(fp, "\n");
      }
      return 1;
    }

    bool Mesh::writeData(FILE * fp) {    ////////////////
      // using fscanf
      for ( int i = 0; i < dat_list_index; i++ ){
          fprintf(fp, "%d %d %d %d %s\n", dat_list[i]->index,
dat_list[i]->set->index,
          dat_list[i]->dim, dat_list[i]->size, dat_list[i]->name);
       if (dat_list[i]->size == 8 ){
          for ( int k = 0; k < dat_list[i]->set->size; k++ ){
             for ( int j = 0; j < dat_list[i]->dim; j++ ){
                fprintf(fp, " %f", dat_list[i]->data[k *
dat_list[i]->dim + j]);
             }
             fprintf(fp, "\n");
          }
       }
       else if (dat_list[i]->size == 4 ){
          for ( int k = 0; k < dat_list[i]->set->size; k++ ){
             for ( int j = 0; j < dat_list[i]->dim; j++ ){
                fprintf(fp, " %d", dat_list[i]->data[k *
dat_list[i]->dim + j]);
             }
             fprintf(fp, "\n");
          }
       }
       fprintf(fp, "\n");
    }
    return 1;
   }

   bool Mesh::readHeader(FILE * fp) {   //////////////////
    int header[6];
    int count = fscanf(fp, "%d %d %d %d %d %d", &element_list_size,
&map_list_size, &dat_list_size,
    &element_list_index, &map_list_index, &dat_list_index);
       return 0;
    return 1;
   }

   bool Mesh::readElements(FILE * fp) {    ///////////////////
    // using fscanf
    for ( int i = 0; i < element_list_index; i++ ){
       element_list[i] = (Elements)malloc(sizeof(elements));
       int count = fscanf(fp, "%d %d %s", &element_list[i]->index,
&element_list[i]->size, element_list[i]->name);
       if (count != 3)
          return 0;
    }
    return 1;
   }
```

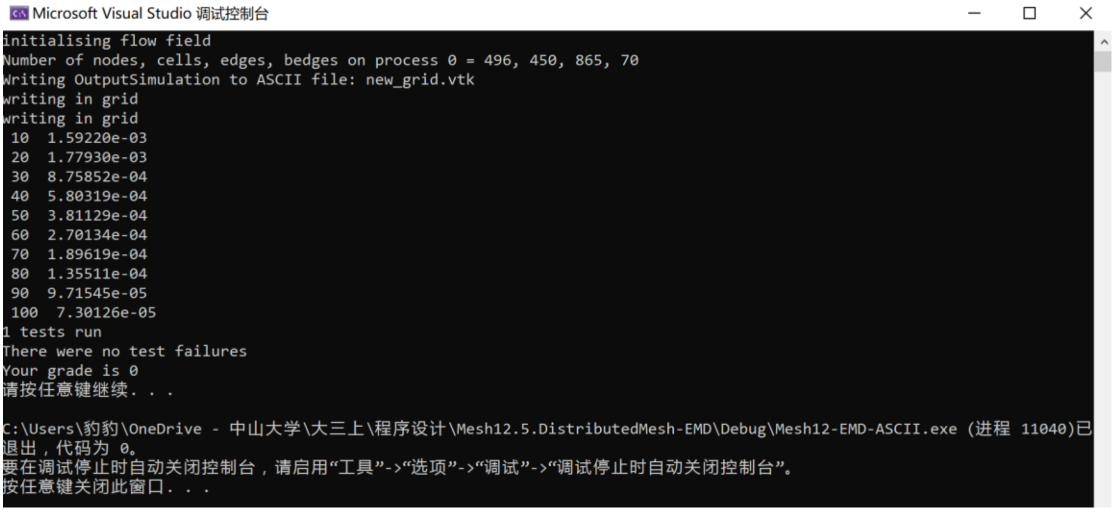
```
    bool Mesh::readMaps(FILE * fp) {    ///////////////////////////
      // using fscanf
      for ( int i = 0; i < map_list_index; i++ ){
          int temp[4];
          int count = fscanf(fp, "%d %d %d %d", &temp[0], &temp[1],
&temp[2], &temp[3]);
          if (count != 4)
              return 0;
          map_list[i] = (Map)malloc(sizeof(map));
          map_list[i]->index = temp[0];
          map_list[i]->from = element_list[temp[1]];
          map_list[i]->to = element_list[temp[2]];
          map_list[i]->dim = temp[3];
          char t[100] = { 0 };
          count = fscanf(fp, "%s", t);
          if (count != 1)
              return 0;
          strcpy((char*)(map_list[i]->name), t);
          map_list[i]->map = (int*)malloc(map_list[i]->dim *
map_list[i]->from->size * sizeof(int));
          for (int j = 0; j < map_list[i]->from->size; j++) {
              for (int k = 0; k < map_list[i]->dim; k++) {
                  count = fscanf(fp, "%d", &map_list[i]->map[j *
map_list[i]->dim + k]);
                  if (count != 1)
                      return 0;
              }
          }
      }
      return 1;
    }

    bool Mesh::readData(FILE * fp) {     ////////////////////
      // using fscanf
      for ( int i = 0; i < dat_list_index; i++ ){
          int temp[4];
          dat_list[i] = (Data)malloc(sizeof(dat));
          int count = fscanf(fp, "%d %d %d %d", &temp[0], &temp[1],
&temp[2], &temp[3]);
          if (count != 4)
              return 0;
          char nam[100] = { 0 };
          count = fscanf(fp, "%s", nam);
```

```
            if (count != 1)
                return 0;
            strcpy((char*)(dat_list[i]->name), nam);
            dat_list[i]->index = temp[0];
            dat_list[i]->set = element_list[temp[1]];
            dat_list[i]->dim = temp[2];
            dat_list[i]->size = temp[3];
            int msize = dat_list[i]->dim * dat_list[i]->set->size;
            if (temp[3] == 4) {
                int* t = (int*)malloc(sizeof(int) * msize);
                for (int j = 0; j < msize; j++)
                    fscanf(fp, "%d", &t[j]);
                dat_list[i]->data = (char*)t;
            }
            else if (temp[3] == 8) {
                double* t = (double*)malloc(sizeof(double) * msize);
                for (int j = 0; j < msize; j++)
                    fscanf(fp, "%lf", &t[j]);
                dat_list[i]->data = (char*)t;
            }
        }
        return 1;
}
```

2. 另外，本人还尝试实现了一下在简单划分的情况下，计算时进程间通信的过程：

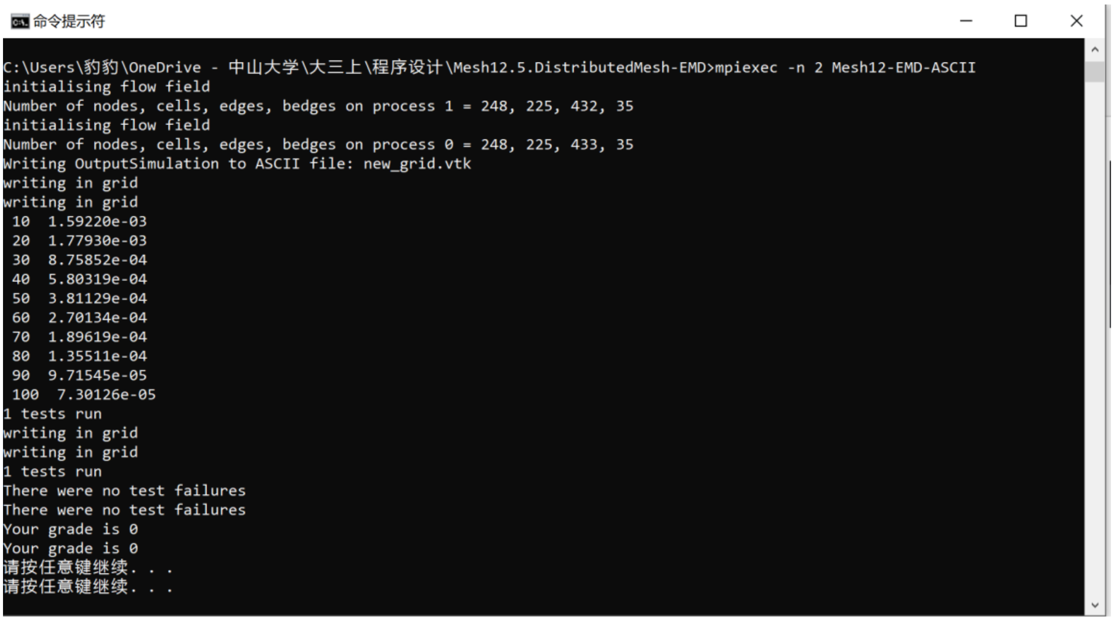
```
int calculate_source_rank(int mpi_comm_size, int g_size, int g_begin, int
elem_size) {
        int* sendcnts = (int*)malloc(mpi_comm_size * sizeof(int));
        int* displs = (int*)malloc(mpi_comm_size * sizeof(int));

        int disp = 0;
        for (int i = 0; i < mpi_comm_size; i++) {
            sendcnts[i] = elem_size * compute_local_size(g_size,
mpi_comm_size, i);
        }
        for (int i = 0; i < mpi_comm_size; i++) {
            displs[i] = disp;
            disp = disp + sendcnts[i];
        }

        // find which process has the data
        int source_rank = 0;
        for (int i = 0; i < mpi_comm_size; i++) {
            if (g_begin < displs[i] + sendcnts[i]) {
              source_rank = i;
              break;
          }
      }
      return source_rank;
    }

void get_int_array_from_another_process(int g_size, int l_size, int
elem_size,
      int my_rank, int destination_rank, int mpi_comm_size, int g_begin, int*
data, int* g_array) {
      int source_rank = calculate_source_rank(mpi_comm_size, g_size, g_begin,
elem_size);

      // calculate local l_begin
      int l_begin = 0;
      for (int i = 0; i < source_rank; i++) {
          l_begin += compute_local_size(g_size, mpi_comm_size, i);
      }

      // check if the data is in the same process
      if (source_rank == destination_rank) {
          // data is in the same process
          // load data into array
          if (source_rank == my_rank) {
              for (int i = 0; i < l_size; i++) {
                  for (int j = 0; j < elem_size; j++) {
                      data[i * elem_size + j] = g_array[(g_begin + i) *
elem_size + j];
                  }
              }
          }
          else {
              //do nothing
          }
      }
      else {
          // data is in another process
          // send data to the process
          if (my_rank == source_rank) {
              MPI_Send(g_array + g_begin * elem_size, l_size * elem_size,
MPI_INT, destination_rank, tag, MPI_COMM_WORLD);
          }
          else if (my_rank == destination_rank) {
                MPI_Recv(data, l_size * elem_size, MPI_INT, source_rank, tag, 
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            }
            else {
                //do nothing
            }
            tag++;
        }
    }
```

5. 实验结果和结论



```
initialising flow field
Number of nodes, cells, edges, bedges on process 0 = 496, 450, 865, 70
Writing OutputSimulation to ASCII file: new_grid.vtk
writing in grid
writing in grid
 10  1.59220e-03
 20  1.77930e-03
 30  8.75852e-04
 40  5.80319e-04
 50  3.81129e-04
 60  2.70134e-04
 70  1.89619e-04
 80  1.35511e-04
 90  9.71545e-05
 100  7.30126e-05
1 tests run
There were no test failures
Your grade is 0
请按任意键继续. . .

C:\Users\豹豹\OneDrive - 中山大学\大三上\程序设计\Mesh12.5.DistributedMesh-EMD\Debug\Mesh12-EMD-ASCII.exe (进程 11040)已
退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

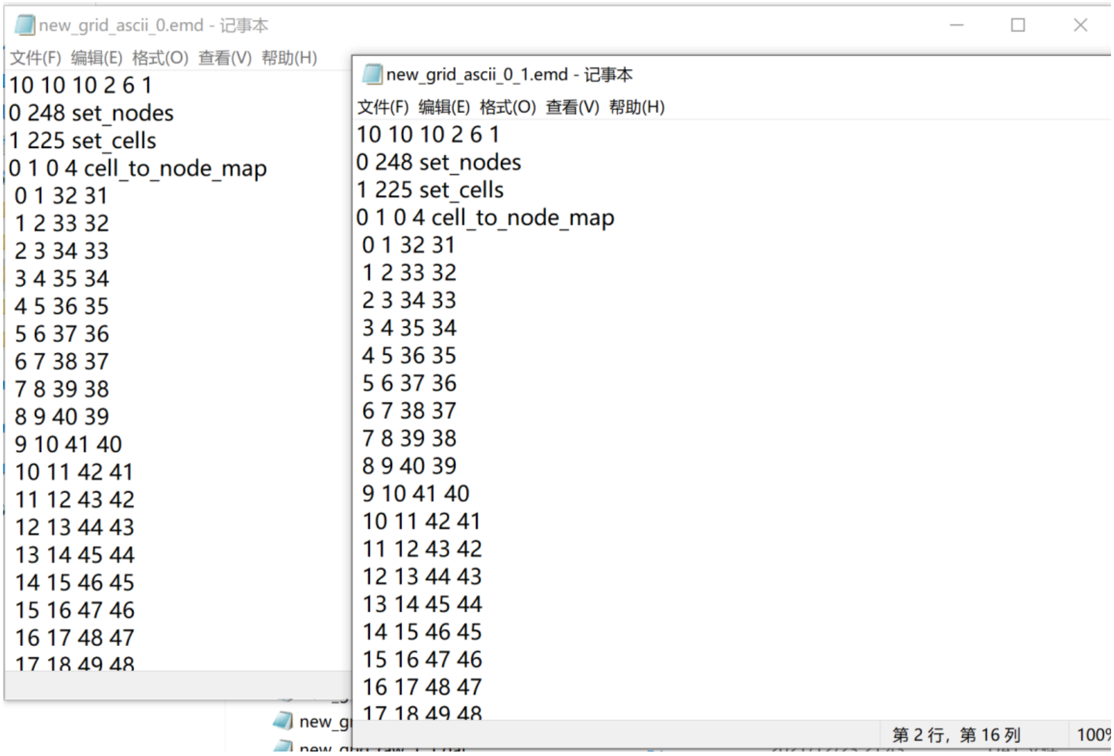
如图所示，在 Visual Studio 2019 中运行串行程序，运行结果正确。以下是用 mpiexec 并行化运行的结果：



```
C:\Users\豹豹\OneDrive - 中山大学\大三上\程序设计\Mesh12.5.DistributedMesh-EMD>mpiexec -n 2 Mesh12-EMD-ASCII
initialising flow field
Number of nodes, cells, edges, bedges on process 1 = 248, 225, 432, 35
initialising flow field
Number of nodes, cells, edges, bedges on process 0 = 248, 225, 433, 35
Writing OutputSimulation to ASCII file: new_grid.vtk
writing in grid
writing in grid
 10  1.59220e-03
 20  1.77930e-03
 30  8.75852e-04
 40  5.80319e-04
 50  3.81129e-04
 60  2.70134e-04
 70  1.89619e-04
 80  1.35511e-04
 90  9.71545e-05
 100  7.30126e-05
1 tests run
writing in grid
writing in grid
1 tests run
There were no test failures
There were no test failures
Your grade is 0
Your grade is 0
请按任意键继续. . .
请按任意键继续. . .
```

如图所示，并行程序运行结果与串行程序相同，分布式 mesh 的读写成功执行。打开相

应的 emd 文件，可以看到前后写入的文件内容一致：


new_grid_ascii_0.emd - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
10 10 10 2 6 1
0 248 set_nodes
1 225 set_cells
0 1 0 4 cell_to_node_map
 0 1 32 31
 1 2 33 32
 2 3 34 33
 3 4 35 34
 4 5 36 35
 5 6 37 36
 6 7 38 37
 7 8 39 38
 8 9 40 39
 9 10 41 40
 10 11 42 41
 11 12 43 42
 12 13 44 43
 13 14 45 44
 14 15 46 45
 15 16 47 46
 16 17 48 47
 17 18 49 48
```

new_grid_ascii_0_1.emd - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
10 10 10 2 6 1
0 248 set_nodes
1 225 set_cells
0 1 0 4 cell_to_node_map
 0 1 32 31
 1 2 33 32
 2 3 34 33
 3 4 35 34
 4 5 36 35
 5 6 37 36
 6 7 38 37
 7 8 39 38
 8 9 40 39
 9 10 41 40
 10 11 42 41
 11 12 43 42
 12 13 44 43
 13 14 45 44
 14 15 46 45
 15 16 47 46
 16 17 48 47
 17 18 49 48
```

至此，实验就成功了。