System Analysis and Design

Homework Assignment 3（期中）

学号：19335286 姓名：郑有为

Examine the source code of a mesh application in the folder Airfoil in the Repository homework. According to your understanding of mesh applications, try to complete the following tasks:

1. **What is the mesh file format in file new_grid.dat? Write out your answer in C structures.**
    - 该文件是一个 2D翼面（Airfoil） 的一个非结构网格存储（Unstructured Mesh）。

    ```
    //new_grid.dat文件中的网格文件格式
    struct new_grid{
        int nnode,ncell,nedge,nbedge;
        double* x[2];
        int* cell[4];
        int* edge[4];
        int* bedge[4];
    };
    ```

    - `new_grid.dat`中的四种基本结构
        1. **Node**
            - 一行表示一个结点，每个Node有两个值`((double)x,(double)y)`，表示该结点在二维坐标轴上的的位置。
        2. **Cell**
            - 在Airfoil案例中，Cell的形状是一个Quad（四面体），故由四个结点组成；
            - 一行表示一个Cell，每个Cell有四个值`[(int)n1,(int)n2,(int)n3,(int)n4]`，表示四个结点的编号。
        3. **Edge**
            - 一行表示一条Edge，每个Edge有有四个值`[(int)n1,(int)n2,(int)c1,(int)c2]`，前两个为结点的编号，后两个为Cell（四面体）的编号；
        4. **Bedge** 全称 **Border Edge**
            - 一行表示一条Border Edge，指的就是整个Mesh边缘上的那些边。每个Border Edge有有四个值`[(int)n1,(int)n2,(int)c1,(int)b2]`，前两个为结点的编号，第三个为Cell（四面体）的编号，最后一个为Border Cell 的编号。
    - `new_grid.dat`的文件布局：
        1. 第一行为四个数，分别为`nnode,ncell,nedge,bedge`，分别表示Node总数，Cell总数，Edge总数，Border Edge总数。
        2. Node数据区域：从第`1`行到第`nnode+1`行为Node数据，每一行有两个元素，我们将其理解为一个`double`类型的二维数组`*x[2]`。
        3. Cell数据区域：紧接下来的`ncell`行为Cell数据，每一行有四个元素，我们将其理解为一个`int`类型的二维数组`*cell[4]`，每行的四个元素都是Node的编号，编号从0开始。
        4. Edge数据区域：紧接下来的`nedge`行为Edge数据，每一行有四个元素，我们将其理解为一个`int`类型的二维数组`*edge[4]`，每行前两个是Node编号，后两行是Cellr编号，都从0开始编号。

5. Bedge数据区域：紧接下来的`nbedge`行为Bedge数据，每一行有四个元素，我们将其理解为一个`int`类型的二维数组`*bedge[4]`，每行前两个是Node编号，第三个是Cell编号，第四个是Border Cell编号。

2. **How to load in a mesh from a mesh file like new_grid.dat? Write out your answer as a C function.**

```
grid* load_gird(const char* pathname){ /*pathname: the .dat file to load*/
    grid* new_grid=(grid*)malloc(sizeof(grid));
    FILE *fp;
    if ( (fp = fopen(pathname,"r")) == NULL) { /*opne grid file*/
        printf("can't open file new_grid.dat\n");
        exit(-1);
    }
    if (fscanf(fp,"%d %d %d %d \n",&new_grid->nnode, &new_grid->ncell,
&new_grid->nedge, &new_grid->nbedge) != 4) {
        printf("error reading from new_grid.dat\n");
        exit(-1);
    }
    new_grid->cell     = (int *) malloc(4*new_grid->ncell*sizeof(int));
    new_grid->edge     = (int *) malloc(2*new_grid->nedge*sizeof(int));
    new_grid->ecell    = (int *) malloc(2*new_grid->nedge*sizeof(int));
    new_grid->bedge    = (int *) malloc(2*new_grid->nbedge*sizeof(int));
    new_grid->becell   = (int *) malloc(  new_grid->nbedge*sizeof(int));
    new_grid->bound    = (int *) malloc(  new_grid->nbedge*sizeof(int));
    new_grid->x        = (real *) malloc(2*new_grid->nnode*sizeof(real));
    new_grid->cell2edge = (int *) malloc(4*new_grid->ncell*sizeof(int));
    for(int i = 0; i< 4*new_grid->ncell; ++i){
        new_grid->cell2edge[i] = -1;
    }
    for (int i=0; i<new_grid->nnode; i++) {
        if (fscanf(fp,"%lf %lf \n",&new_grid->x[2*i], &new_grid->x[2*i+1])
!= 2) {
            printf("error reading from new_grid.dat\n");
            exit(-1);
        }
    }
    for (int i=0; i<new_grid->ncell; i++) {
        if (fscanf(fp,"%d %d %d %d \n",&new_grid->cell[4*i], &new_grid-
>cell[4*i+1], &new_grid->cell[4*i+2], &new_grid->cell[4*i+3]) != 4) {
            printf("error reading from new_grid.dat\n"); exit(-1);
        }
    }

    for (int n=0; n<new_grid->nedge; n++) {
        if (fscanf(fp,"%d %d %d %d \n",&new_grid->edge[2*n], &new_grid-
>edge[2*n+1], &new_grid->ecell[2*n],&new_grid->ecell[2*n+1]) != 4) {
            printf("error reading from new_grid.dat\n"); exit(-1);
        }
        for(int i = 0; i < 4;++i){
            if(new_grid->cell2edge[new_grid->ecell[2*n]+i] == -1){
                new_grid->cell2edge[new_grid->ecell[2*n]+i] = n;
                break;
            }
        }
        for(int i = 0; i < 4;++i){
            if(new_grid->cell2edge[new_grid->ecell[2*n+1]+i] == -1){
                new_grid->cell2edge[new_grid->ecell[2*n+1]+i] = n;
```

```
                break;
            }
        }
    }/*bedge[0],bedge[1]:edge bedge[2]:becell bedge[3]:bound*/

    for (int n=0; n<new_grid->nbedge; n++) {
        if (fscanf(fp,"%d %d %d %d \n",&new_grid->bedge[2*n],&new_grid-
>bedge[2*n+1], &new_grid->becell[n], &new_grid->bound[n]) != 4) {
            printf("error reading from new_grid.dat\n"); exit(-1);
        }
    }
    fclose(fp);
    return new_grid;
}
```

3. **What is the in-memory storage structure of a mesh? Write out your answer as a fragment of C program.**

整理了一下源程序中定义的一些相关变量：

```
typedef struct new_grid{ /*grid struct*/
    int nnode,ncell,nedge,nbedge;
    double* x;
    int* cell;
    int* edge;
    int* bedge;
    int* becell;
    int* bound;
    int* ecell;
    int* cell2edge;
}grid;
```

4. **What are the kernel functions as the operations over mesh elements in this mesh application? Write out your answers as C functions.**

```
double set_timer(){
    return omp_get_wtime();
}

double get_duration(double start){
    return omp_get_wtime()-start;
}

typedef struct {
    grid* Grid;
    real* q;
    real* adt;
}CAT_ARG;

void* calculate_area_timstep(void* arg,int i){
    CAT_ARG* argu=(CAT_ARG*)arg;
    grid* Grid=argu->Grid;
    real* q=argu->q;
    real* adt=argu->adt;
    double dx,dy, ri,u,v,c;
    int* cell = Grid->cell;
    real* x = Grid->x;
```

```
    ri =  1.0f/q[4*i+0];
    u  =   ri*q[4*i+1];
    v  =   ri*q[4*i+2];
    c  = sqrt(gam*gm1*(ri*q[4*i+3]-0.5f*(u*u+v*v)));

    dx = x[cell[4*i+1]*2+0] - x[cell[4*i+0]*2+0];
    dy = x[cell[4*i+1]*2+1] - x[cell[4*i+0]*2+1];
    adt[i]  = fabs(u*dy-v*dx) + c*sqrt(dx*dx+dy*dy);

    dx = x[cell[4*i+2]*2+0] - x[cell[4*i+1]*2+0];
    dy = x[cell[4*i+2]*2+1] - x[cell[4*i+1]*2+1];
    adt[i] += fabs(u*dy-v*dx) + c*sqrt(dx*dx+dy*dy);

    dx = x[cell[4*i+3]*2+0] - x[cell[4*i+2]*2+0];
    dy = x[cell[4*i+3]*2+1] - x[cell[4*i+2]*2+1];
    adt[i] += fabs(u*dy-v*dx) + c*sqrt(dx*dx+dy*dy);

    dx = x[cell[4*i+0]*2+0] - x[cell[4*i+3]*2+0];
    dy = x[cell[4*i+0]*2+1] - x[cell[4*i+3]*2+1];
    adt[i] += fabs(u*dy-v*dx) + c*sqrt(dx*dx+dy*dy);
    adt[i] = adt[i] / cfl;//反正就是把四个点的xy值计算一下存起来
}

void calculate_flux_residual(grid* Grid,int i,real* q,real* adt,real* res){
    double dx,dy,mu, ri, p1,vol1, p2,vol2, f;
    real* x=Grid->x;
    int* edge=Grid->edge;
    int* ecell=Grid->ecell;
    dx = x[edge[2*i+0]*2+0] - x[edge[2*i+1]*2+0];
    dy = x[edge[2*i+0]*2+1] - x[edge[2*i+1]*2+1];

    ri   = 1.0f/q[ecell[2*i+0]*4+0];
    p1   = gm1*(q[ecell[2*i+0]*4+3]-0.5f*ri*
(q[ecell[2*i+0]*4+1]*q[ecell[2*i+0]*4+1]+q[ecell[2*i+0]*4+2]*q[ecell[2*i+0]*
4+2]));
    vol1 =  ri*(q[ecell[2*i+0]*4+1]*dy - q[ecell[2*i+0]*4+2]*dx);

    ri   = 1.0f/q[ecell[2*i+1]*4+0];
    p2   = gm1*(q[ecell[2*i+1]*4+3]-0.5f*ri*
(q[ecell[2*i+1]*4+1]*q[ecell[2*i+1]*4+1]+q[ecell[2*i+1]*4+2]*q[ecell[2*i+1]*
4+2]));
    vol2 =  ri*(q[ecell[2*i+1]*4+1]*dy - q[ecell[2*i+1]*4+2]*dx);

    mu = 0.5f*(adt[ecell[2*i+0]]+adt[ecell[2*i+1]])*eps;

    f = 0.5f*(vol1* q[ecell[2*i+0]*4+0]         + vol2* q[ecell[2*i+1]*4+0]
       ) + mu*(q[ecell[2*i+0]*4+0]-q[ecell[2*i+1]*4+0]);
    res[ecell[2*i+0]*4+0] += f;
    res[ecell[2*i+1]*4+0] -= f;

    f = 0.5f*(vol1* q[ecell[2*i+0]*4+1] + p1*dy + vol2* q[ecell[2*i+1]*4+1]
+ p2*dy) + mu*(q[ecell[2*i+0]*4+1]-q[ecell[2*i+1]*4+1]);
    res[ecell[2*i+0]*4+1] += f;
    res[ecell[2*i+1]*4+1] -= f;
    f = 0.5f*(vol1* q[ecell[2*i+0]*4+2] - p1*dx + vol2* q[ecell[2*i+1]*4+2]
- p2*dx) + mu*(q[ecell[2*i+0]*4+2]-q[ecell[2*i+1]*4+2]);
    res[ecell[2*i+0]*4+2] += f;
    res[ecell[2*i+1]*4+2] -= f;
```

```
    f = 0.5f*(vol1*(q[ecell[2*i+0]*4+3]+p1)     + vol2*
(q[ecell[2*i+1]*4+3]+p2)    ) + mu*(q[ecell[2*i+0]*4+3]-
q[ecell[2*i+1]*4+3]);
    res[ecell[2*i+0]*4+3] += f;
    res[ecell[2*i+1]*4+3] -= f;
}

void Apply_boundary_conditions(grid* Grid,real* adt,real* q,real* res,real*
qinf,int index){
    int i=index;
    double dx,dy,mu, ri, p1,vol1, p2,vol2, f;
    int nbedge=Grid->nbedge;
    int* bedge=Grid->bedge;
    int* becell=Grid->becell;
    int* bound=Grid->bound;
    real* x=Grid->x;
    dx = x[bedge[2*i+0]*2+0] - x[bedge[2*i+1]*2+0];
    dy = x[bedge[2*i+0]*2+1] - x[bedge[2*i+1]*2+1];

    ri = 1.0f/q[becell[i]*4+0];
    p1 = gm1*(q[becell[i]*4+3]-0.5f*ri*
(q[becell[i]*4+1]*q[becell[i]*4+1]+q[becell[i]*4+2]*q[becell[i]*4+2]));

    if (bound[i]==1) { //Far-field
        res[becell[i]*4+1] += + p1*dy;
        res[becell[i]*4+2] += - p1*dx;
    }
    else {
        vol1 =  ri*(q[becell[i]*4+1]*dy - q[becell[i]*4+2]*dx);

        ri   = 1.0f/qinf[0];
        p2   = gm1*(qinf[3]-0.5f*ri*(qinf[1]*qinf[1]+qinf[2]*qinf[2]));
        vol2 =  ri*(qinf[1]*dy - qinf[2]*dx);

        mu = adt[becell[i]]*eps;

        f = 0.5f*(vol1* q[becell[i]*4+0]         + vol2* qinf[0]        ) +
mu*(q[becell[i]*4+0]-qinf[0]);
        res[becell[i]*4+0] += f;
        f = 0.5f*(vol1* q[becell[i]*4+1] + p1*dy + vol2* qinf[1] + p2*dy) +
mu*(q[becell[i]*4+1]-qinf[1]);
        res[becell[i]*4+1] += f;
        f = 0.5f*(vol1* q[becell[i]*4+2] - p1*dx + vol2* qinf[2] - p2*dx) +
mu*(q[becell[i]*4+2]-qinf[2]);
        res[becell[i]*4+2] += f;
        f = 0.5f*(vol1*(q[becell[i]*4+3]+p1)     + vol2*(qinf[3]+p2)    ) +
mu*(q[becell[i]*4+3]-qinf[3]);
        res[becell[i]*4+3] += f;
    }
}

void calculate_rms(grid* Grid,real* rms,real* adt,real* q,real* qold,real*
res,int index){
    double del, adti;
    int i=index;
    adti = 1.0f/adt[i];
    for (int n=0; n<4; n++) {
        del    = adti*res[4*i+n];
```

```
            q[4*i+n]    = qold[4*i+n] - del;
            res[4*i+n] = 0.0f;
            *rms   += del*del;
        }
    }
```

5. **What is the whole user algorithm of this mesh application? Write out the algorithm as a C function.**

   观察到迭代执行所需要的前置数据包含着色函数的结果，所以原码中的主函数就是用户的整一个算法的执行函数。

```
void user(){
    real  *q, *qold, *adt, *res;

    int    niter;
    real  rms;

    //timer
    double wall_t1, wall_t2;

    // read in grid
    printf("reading in grid \n");

    grid* Grid=load_gird("./new_grid.dat");//load in grid
    q      = (real *) malloc(4*(Grid->ncell)*sizeof(real));
    qold   = (real *) malloc(4*(Grid->ncell)*sizeof(real));
    res    = (real *) malloc(4*(Grid->ncell)*sizeof(real));
    adt    = (real *) malloc(  (Grid->ncell)*sizeof(real));

    //set variables for graph coloring
    std::vector<std::vector<int>>  color2edge=Edge_coloring(Grid);
    printf("%d %d %d %d %d %d",
color2edge[0].size(),color2edge[1].size(),color2edge[2].size(),color2edge[3]
.size(),color2edge[4].size(),color2edge[5].size());

    // set constants and initialise flow field and residual
    printf("initialising flow field \n");

    gam = 1.4f;
    gm1 = gam - 1.0f;
    cfl = 0.9f;
    eps = 0.05f;

    real mach  = 0.4f;
    real alpha = 3.0f*atan(1.0f)/45.0f;
    real p     = 1.0f;
    real r     = 1.0f;
    real u     = sqrt(gam*p/r)*mach;
    real e     = p/(r*gm1) + 0.5f*u*u;

    qinf[0] = r;
    qinf[1] = r*u;
    qinf[2] = 0.0f;
    qinf[3] = r*e;

    for (int n=0; n<Grid->ncell; n++) {
```

```
        for (int m=0; m<4; m++) {
            q[4*n+m] = qinf[m];
            res[4*n+m] = 0.0f;
        }
    }

    //initialise timers for total execution wall time
    wall_t1 = set_timer();

    // main time-marching loop

    niter = 1000;
    double save = 0, area = 0, update = 0, flux_res = 0, perem = 0, wall_t_a
= 0, wall_t_b = 0;

    CAT_ARG cat_arg;
    cat_arg.Grid=Grid;
    cat_arg.q=q;
    cat_arg.adt=adt;

    #pragma omp parallel
    {
    for(int iter=1; iter<=niter; iter++) {
        wall_t_b = set_timer();
        // save old flow solution
        #pragma omp for
        for (int i = 0; i < Grid->ncell; i++) {
            for (int n=0; n<4; n++) qold[4*i+n] = q[4*i+n];
        }

        save += get_duration(wall_t_b);

        // predictor/corrector update loop
        for(int k=0; k<2; k++) {
            #pragma omp single
            {
                wall_t_b = set_timer();
            }

            looping(0,Grid->ncell-1,calculate_area_timstep,(void*)&cat_arg);

            #pragma omp single
            {
                area += get_duration(wall_t_b);
                // calculate flux residual
                wall_t_b = set_timer();
            }

            for(size_t color = 0; color < color2edge.size(); ++color){
                #pragma omp for
                for(size_t ind = 0; ind < color2edge[color].size(); ++ind){
                    calculate_flux_residual(Grid,color2edge[color]
[ind],q,adt,res);
                }
            }

            #pragma omp single
            {
```

```
                flux_res += get_duration(wall_t_b);
                // Apply boundary conditions
                wall_t_b = set_timer();
                for (int i = 0; i < Grid->nbedge; i++) {
                    Apply_boundary_conditions(Grid,adt,q, res,qinf,i);
                }
                perem += get_duration(wall_t_b);

                // update flow field
                wall_t_b = set_timer();
                rms = 0.0;
            }

            #pragma omp for reduction(+:rms)
            for (int i = 0; i < Grid->ncell; i++) {
                calculate_rms(Grid,&rms,adt,q,qold,res,i);
            }
            update += get_duration(wall_t_b);
        }

        #pragma omp single
        {
            rms = sqrt(rms/(double) Grid->ncell);
            if (iter%100 == 0){
                printf(" %d  %10.5e \n",iter,rms);
                printf("\tsave: %f\n",save);
                printf("\tarea: %f\n",area);
                printf("\tflux_res: %f\n",flux_res);
                printf("\tperem: %f\n",perem);
                printf("\tupdate: %f\n",update);
                char buf[50];
                sprintf(buf,"out%d.vtk",iter);
                WriteMeshToVTKAscii(buf, Grid->x, Grid->nnode, Grid->cell,
Grid->ncell, q);
            }
        }
    }
}
wall_t2 = get_duration(wall_t1);
printf("Max total runtime = \n%f\n",wall_t2);
free(q);
free(qold);
free(res);
free(adt);
grid_free(Grid);
}
```

6. **In this mesh application, the computed mesh finally is stored in a VTK format text file or a binary file. Describe in some detail what is the storage structure of the VTK text format of mesh?**
    - 详见 hw3-6.pdf
7. **In this mesh application, there is a fragment of code doing coloring over the mesh. Try to understand what this coloring does, describe in a paragraph to explain your understanding about this coloring algorithm.**
    - 作用：给mesh的每一条边上色，使得所有cell中，所有边不重色。

- 算法的理解：

  ```
      for(int i = 0; i < 4;++i){
          if(cell2edge[ecell[2*n]+i] == -1){
              cell2edge[ecell[2*n]+i] = n;
              break;
          }
      }
      for(int i = 0; i < 4;++i){
          if(cell2edge[ecell[2*n+1]+i] == -1){
              cell2edge[ecell[2*n+1]+i] = n;
              break;
          }
      }
  ```

  - cell2edge的数组的功能在于给由对应的四边形单元找到对应的边。
  - 上面是以读取的边为顺序对cell2edge进行初始化。

  ```
      for(int edge_ind = 0; edge_ind < nedge; ++edge_ind){
          int color = 0;
          while(1){
              bool valid_color = true;
              for(int i = 0; i < 4; ++i){
                  if(edge2color[cell2edge[ecell[edge_ind]+i]] == color ||
                          edge2color[cell2edge[ecell[edge_ind+1]+i]] == color)
  {
                      valid_color = false;
                  }
              }
              if(valid_color){
                  edge2color[edge_ind] = color;
                  if(color2edge.size() == color){
                      color2edge.push_back(std::vector<int>(1,edge_ind));
                  } else if(color < color2edge.size()){
                      color2edge[color].push_back(edge_ind);
                  } else {
                      printf("ismet para van\n");
                  }
                  if(color > max){
                      max = color;
                      printf("%d\n",max);
                  }
                  break;
              }
              ++color;
          }
  }
  ```

- 这里则是对每条边的上色过程。color2edge为存放各种颜色的边的vector。
- 先对对应边的两个四边形单元的八条边（有重复）判断是否有和即将要涂的色有相同的（由于将要涂色的边标记为尚未涂色，所以不用考虑与自身相同），颜色从小到大枚举，如果对应单元没有该颜色，则对该边上该优颜色，并把该边放置在该颜色的队列里。

8. **Rewrite the source code in the folder Airfoil using your own functions you have done above.**

- 见同文件中的airfoil_seq.cpp
- 检查整体运行情况，解压Airfoil_new.zip，make并运行airfoil_seq即可

以下是整个程序重写后的整理：

DOCUMENT TYPE

1. `.dat`
2. `.vtk`

GRID STRUCTURE

```
typedef struct new_grid{ /*grid struct*/
    int nnode,ncell,nedge,nbedge;
    double* x;
    int* cell;
    int* edge;
    int* bedge;
    int* becell;
    int* bound;
    int* ecell; // ecell每两个一组，记录第edge_ind条边所在的两个四面体的索引
    int* cell2edge; // ell2edge每四个一组，记录每个四面体的四条边的索引
}grid;
```

```
// 一种不规范但易于理解的视角，根据load函数调整了成员变量的顺序
int nnode,ncell,nedge,nbedge;
//点数、四面体数、（非边缘边）边数，边缘边边数
typedef struct new_grid{
    double x[2  * nnode];
    // ∧ 数组元素每相邻2个一组，表示结点的(X,Y)坐标

    int cell[4 * ncell];
    // ∧ 数组元素每相邻4个一组，记录一个四面体单元的4个顶点的标号
    int cell2edge[4 * ncell];
    // ∧ 数组元素每相邻4个一组，记录每个四面体的四条边的标号

    int edge[2 * nedge];
    // ∧ 数组元素每相邻2个一组，记录一条边的2个顶点的标号
    int ecell[2 * nedge];
    // ∧ 数组元素每相邻2个一组，记录一条边所在的两个四面体的标号

    int bedge[2 * nbedge];
    // ∧ 数组元素每相邻2个一组，记录一条边缘边的2个顶点的标号
    int becell[nbedge];
    // ∧ 记录一条边缘边所在的四面体的标号
    int bound[nbedge];
    // ∧ 记录一条边缘边的边界的标号
}grid;
```

MESH FUNCTION

ATRFOIL SEQ

1. 将`.dat`格式的Mesh数据载入`grid`结构体：

    ```
    grid* load_gird(const char* pathname);
    ```

2. 设置计时器，获取当前时间：

    ```
    double set_timer();
    ```

3. 获取从开始时间(start)到此时的时长：

    ```
    double get_duration(double start);
    ```

4. 解决边着色问题，使每个四面体的四条边颜色各不相同，返回二维向量（外：颜色队列，内：某种颜色的边队列）：

    ```
    std::vector<std::vector<int>> edge_coloring(grid *Grid);
    ```

5. 利用四面体各顶点计算两个四面体的ADT值，更新数组`adt`：

    ```
    void *calculate_area_timstep(void *arg, int i);
    ```

6. 计算通量残留，结果存放在`res`数组中：

    ```
    void calculate_flux_residual(grid *Grid, int i, real *q, real *adt, real *res);
    ```

7. 应用边界条件，更新`res`数组：

    ```
    void apply_boundary_conditions(grid *Grid, real *adt, real *q, real *res, real *qinf, int index)
    ```

8. 计算RMS值：

    ```
    void calculate_rms(grid *Grid, real *rms, real *adt, real *q, real *qold, real *res, int index);
    ```

9. 释放`grid`内存：

    ```
    void grid_free(grid *Grid);
    ```

10. 多线程循环执行`end-start`次`func`函数：

    ```
    void looping(int start, int end, void *(*func)(void *, int), void *arg);
    ```

11. 用户执行函数：

```
void user()
```

用户函数做了什么呢?

> 1. **load_gird**: 将`new_grid.dat`文件的内容读入`grid`结构体。
> 2. 申请了四个`double`数组`q[4*ncell], qold[4*ncell], res[4*ncell], adt[ncell]`，功能未知。
> 3. **edge_coloring**: 求出`gird`的一种边染色方案，保存在`color2edge`中。
> 4. 初始化`double`数组`q[4*ncell], res[4*ncell]`，前者每四个一组，赋值成`r, r*u, 0, r*e`，后者全部赋值成0。
> 5. 保存数组`q`到`qold`中，将耗时算进 **save** 中。
> 6. 调用**looping**，多线程执行`ncell`次 **calculate_area_timstep** 函数，执行完毕后，将耗时算进 **area** 中。
> 7. 遍历 **color2edge** 执行`nedge`次 **calculate_flux_residual** 函数，执行完毕后，将耗时算进 **flux_res** 中。
> 8. **单线程**执行`nbedge`次 **apply_boundary_conditions** 函数，执行完毕后，将耗时算进 **perem** 中。
> 9. 多线程执行`ncell`次 **calculate_rms** 函数，**rms**是特殊变量，执行完毕后，将耗时算进 **update** 中，各线程的**rms**相加。
> 10. 输出上述耗时，每执行一百次，并将程序结果写入到一个VTK文件中。
> 11. 其中，第四步到第十步循环执行1000次，再外层循环内，第六步到第九步每次执行两次，作为预估和校正。
> 12. 最后输出总耗时，并释放申请的空间。

WRITE VTK

两种写入格式：AscII和二进制写入。

12. AscII格式写入VTK文件:

```
void WriteMeshToVTKAscii(const char *filename, double *nodeCoords_data, int nnode, int *cellsToNodes_data, int ncell, double *values_data);
```

13. 二进制写入VTK文件:

```
void WriteMeshToVTKBinary(const char *filename, double *nodeCoords_data, int nnode, int *cellsToNodes_data, int ncell, double *values_data);
```

OPENMP INSTURCUTION

1. `#pragma omp parallel`

   用在一个代码段之前，表示这段代码将被多个线程并行执行

2. `#pragma omp for`

   表示for循环的代码将被多个线程并行执行

3. `#pragma omp single`

   表示后面的代码段将被单线程执行

4. `#pragma omp for reduction(+ : rms)`

   reduction子句为变量指定一个操作符，每个线程都会创建reduction变量的私有拷贝，在OpenMP区域结束处，将使用各个线程的私有拷贝的值通过制定的操作符进行迭代运算，并赋值给原来的变量。