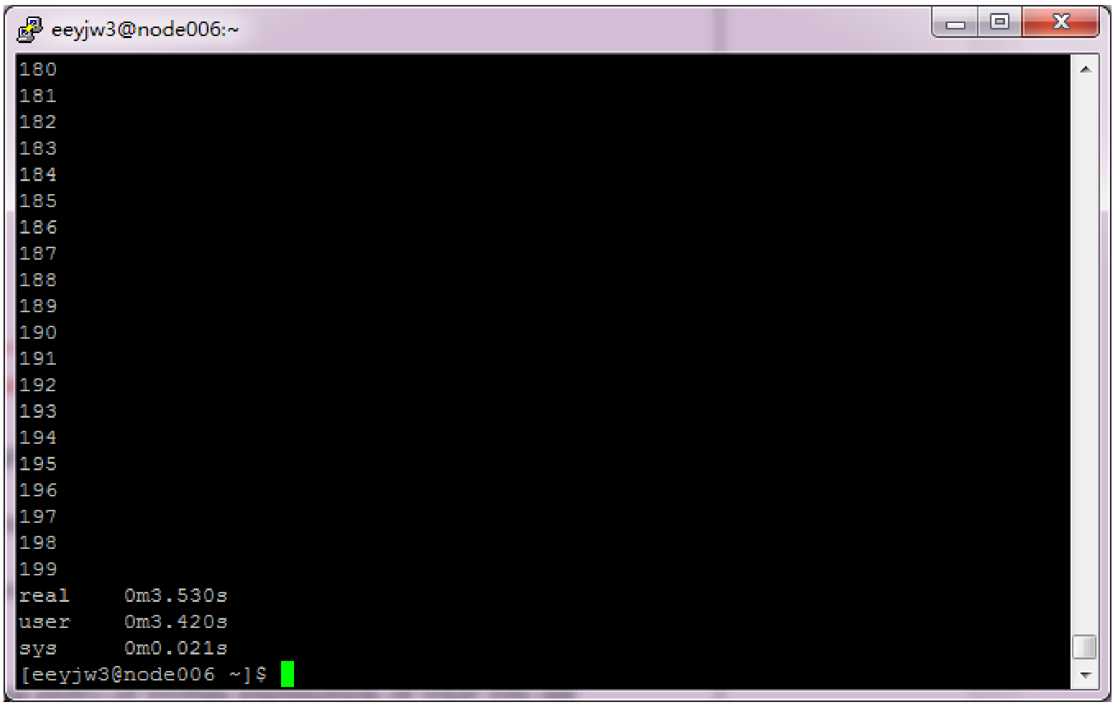
Task1 original code

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

int main(int argc, char* argv[]) {

        int nx(10000);

        int ny(200);

        int nt(200);
        double** vi=new double*[nx];
        double** vr=new double*[nx];
        double pi=(4.*atan(1.));
        for(int i=0;i<nx;i++) {

                vi[i]=new double[ny];

                vr[i]=new double[ny];

        }

        for(int i=0;i<nx;i++) {

                for(int j=0;j<ny;j++) {//j++ should be replaced with ++j in the for loop

                  vi[i][j]=double(i*i)*double(j)*sin(pi/double(nx)*double(i));//pi/nx is
                                                                    //recalculated
                        vr[i][j]=0.;
                }
        }
        ofstream fout("data.txt");

        for(int t=0;t<nt;t++) { //There are many judgments in the loop that reduces the
                                //speed. The loop can be rewritten to avoid these
                                //judgments.
        cout<<"\n"<<t;cout.flush();

        for(int i=0;i<nx;i++) {

          for(int j=0;j<ny;j++) {
          if(i>0&&i<nx-1&&j>0&&j<ny-1) {

          vr[i][j]=(vi[i+1][j]+vi[i-1][j]+vi[i][j-1]+vi[i][j+1])/4.;

          } else if(i==0&&i<nx-1&&j>0&&j<ny-1) {

          vr[i][j]=(vi[i+1][j]+10.+vi[i][j-1]+vi[i][j+1])/4.;

          } else if(i>0&&i==nx-1&&j>0&&j<ny-1) {

          vr[i][j]=(5.+vi[i-1][j]+vi[i][j-1]+vi[i][j+1])/4.;

          } else if(i>0&&i<nx-1&&j==0&&j<ny-1) {

          vr[i][j]=(vi[i+1][j]+vi[i-1][j]+15.45+vi[i][j+1])/4.;

          } else if(i>0&&i<nx-1&&j>0&&j==ny-1) {

        vr[i][j]=(vi[i+1][j]+vi[i-1][j]+vi[i][j-1]-6.7)/4.;

        }
        }
}
for(int i=0;i<nx;i++) {                    // This loop and the next loop can be combined
        for(int j=0;j<ny;j++) {     //and there is no controdiction
        if(fabs(fabs(vr[i][j])-fabs(vi[i][j]))<1e-2) fout<<"\n"<<t<<" "<<i<<" "<<j<<"
        "<<fabs(vi[i][j])<<" "<<fabs(vr[i][j]);
          }
          }

      for(int i=0;i<nx;i++) {

    for(int j=0;j<ny;j++) vi[i][j]=vi[i][j]/2.+vr[i][j]/2.;

        }
        }
        }
```

Task1 modified code

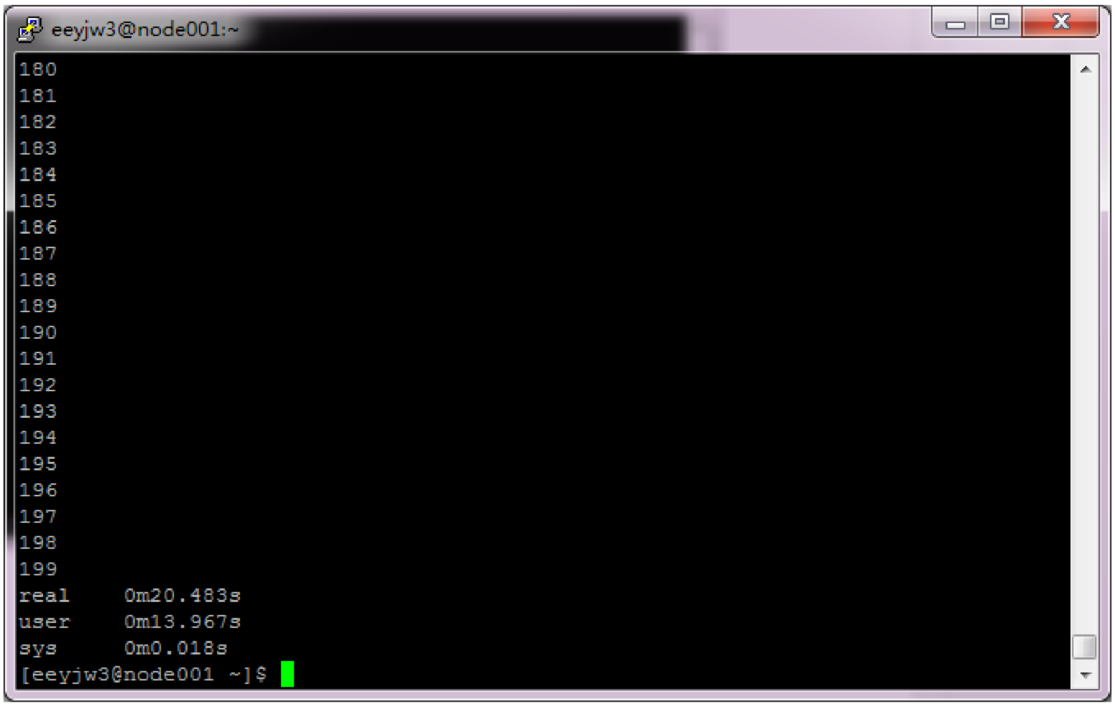
```
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdio.h>
#include <string.h>
using namespace std;

int main(int argc, char* argv[]) {
        int const nx(10000);

        int const ny(200);

        int const nt(200);
        int i,j,t,m,k;
        double** vi=new double*[nx];

        double** vr=new double*[nx];

        double const pi=(4.*atan(1.))/double(nx);//4*atan(1)/nx is set as a const to avoid
                                                  //recalculation

        for(int i=0;i<nx;i++) { //The judgments in the loop is removed by replacing the
                                //the loop with more loops without judgments
                vi[i]=new double[ny];

                vr[i]=new double[ny];

        }

        for(int i=0;i<nx;i++) {

                for(int j=0;j<ny; ++j) { //j++ is replaced with ++j

                        vi[i][j]=double(i*i)*double(j)*sin(pi*double(i));
                        vr[i][j]=0.;
                }
        }


        ofstream fout("data_out.txt");

        for( t=0;t<nt;++t) {
                cout<<"\n"<<t;cout.flush();

                i=0;
                for(j=1;j<ny-1;++j)
                {
                  vr[i][j]=(vi[i+1][j]+10.+vi[i][j-1]+vi[i][j+1])*0.25;

                }
                i=nx-1;
                for(j=1;j<ny-1;++j)
                {
                  vr[i][j]=(5.+vi[i-1][j]+vi[i][j-1]+vi[i][j+1])*0.25;
                }
                j=0;
                for(i=1;i<nx-1;++i)
                {
                vr[i][j]=(vi[i+1][j]+vi[i-1][j]+15.45+vi[i][j+1])*0.25;
                }
                j=ny-1;
                for(i=1;i<nx-1;++i)
                {
                   vr[i][j]=(vi[i+1][j]+vi[i-1][j]+vi[i][j-1]-6.7)*0.25;
                }
                for( i=1;i<nx-1;++i) {
                        for( j=1;j<ny-1;++j) {
                        vr[i][j]=(vi[i+1][j]+vi[i-1][j]+vi[i][j-1]+vi[i][j+1])*0.25;

                        }
                }

                for( i=0;i<nx;++i) { // Two loops are combined without contradiction

                        for( j=0;j<ny;++j) {

          if(fabs(fabs(vr[i][j])-fabs(vi[i][j]))<1e-2) fout<<"\n"<<t<<" "<<i<<" "<<j<<"
          "<<fabs(vi[i][j])<<" "<<fabs(vr[i][j]);

          vi[i][j]=(vi[i][j]+vr[i][j])*0.5;
                        }
                }
}
}
```

The best speed that I have achieved is 3.53 s. The program is compiled with –O4 to increase the speed. Execution time for modified code is shown in figure 1. The previous code takes 20.48 s. The execution of the original code is shown in figure 2.



```
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
real    0m3.530s
user    0m3.420s
sys     0m0.021s
[eeyjw3@node006 ~]$
```

Figure 1: Execution time with modified code



```
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
real    0m20.483s
user    0m13.967s
sys     0m0.018s
[eeyjw3@node001 ~]$
```

Figure 2: Execution time of original code

Task2 code

```
#include <iostream>
#include <sys/time.h>
```

```
using namespace std;
int main()
{      int i,j;
       int n=150;
       int m=1000000;
       double *a;
       double *b;
       double *c;
       a=new double [n];
       b=new double [n];
       c=new double [n];

       int offset=50;
       struct timeval start_time,end_time;

       for (i=0;i<n;i+=1)
       {
              b[i]=i;
              c[i]=i;
       }
       gettimeofday(&start_time,NULL);
       for (j=0;j<=m;j+=1)
       {
       for (i=1;i<n-offset;i+=1)
       {
         a[i]=b[i]+c[i+offset];
       }
       }
       gettimeofday(&end_time,NULL);
       cout<<"\n\ngettimeofday wall time="<<end_time.tv_sec -
start_time.tv_sec+(end_time.tv_usec-start_time.tv_usec)/1e6<<endl;

}
```

There are two loops in this task2. The inner loop does the required calculation and an outer loop is set to repeat the inner loop. The inner loop is repeated for 1000 times. The program is compiled with –O4. Different n is chosen and the execution time is recorded. The figure 3 is plotted to represent the relationship between execution time and the value n.

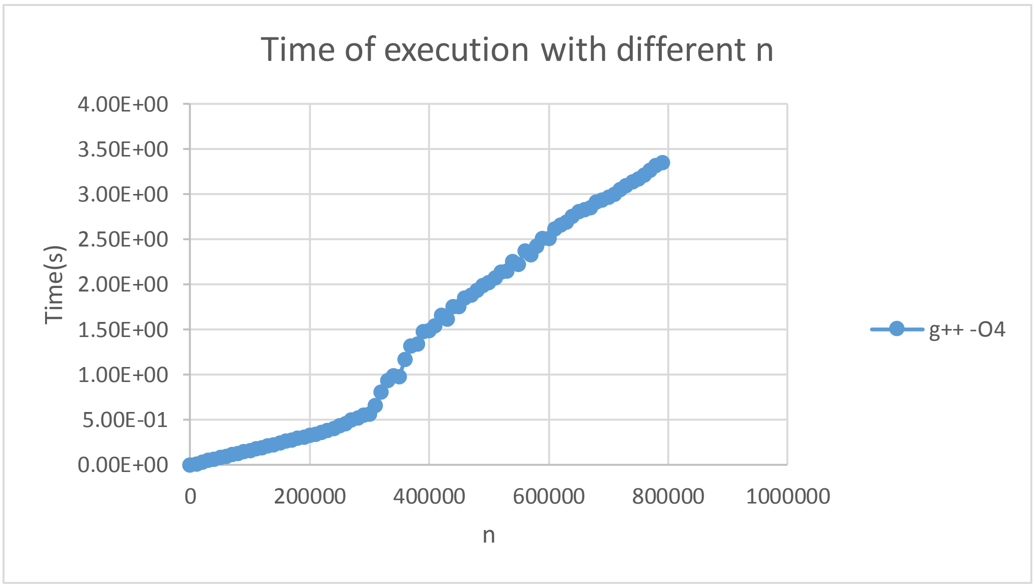


Figure 3: The relationship between execution time and value of n

It can be found that the execution time is not linear with the number of executions. There are inflection points in the drawn figure. Hence, when the n doubles, the execution time is not exactly doubled. Because the L1 cache and L2 cache have capacity. When the number of execution is beyond the certain amount, an inflection point appears. It can be found that there is an obvious inflection point between 300000 and 400000. Then the slope of the curve becomes lager.

Different values of offset are chosen when n equals to 500000, 700000 and 1000000. The execution time is recorded. Figure 4 represents the relationship between execution time and offset value.

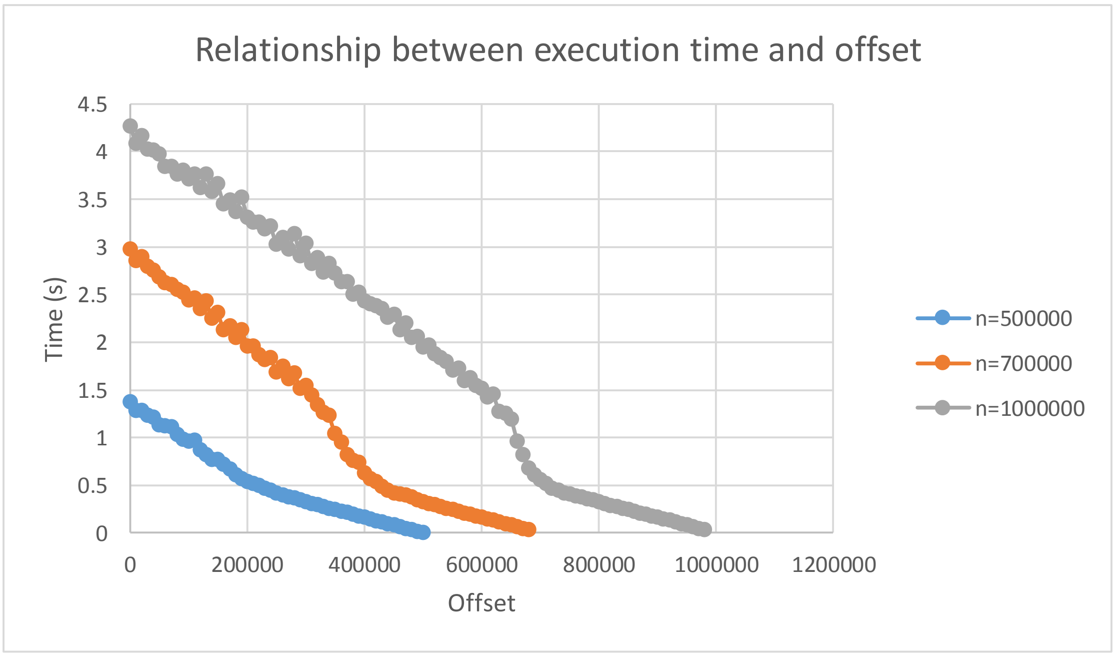


Figure 4: The execution time different value of offset is chosen

It can be found that the relationship between the execution time and the offset value is a smooth curve. However, the relationship is not linear relationship. Due to the capacity of caches. When the (n-offset) is close to 350000, there is an infraction point on the figure 4. The result generally meets figure 3.

Task4

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <math.h>
#include <stdlib.h>
#include <time.h>              /* time */
#include <vector>
#include <omp.h>
using namespace std;
fstream f1;
struct point
{
    int id;
    float x;
    float y;
    float z;
};

struct triangle
{     int id;
      int n1;
      int n2;
      int n3;

};

float sign (point p1, point p2, point p3);

bool PointInTriangle (point pt, point v1, point v2, point v3);

class interface
{
private:
      int np,nd,n_o_t;
```

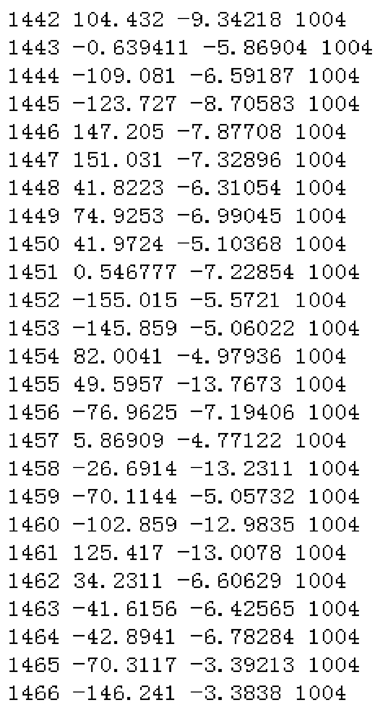
```
        point *points;
        triangle *tri;
        vector<int> array;
public:
        interface(){}
        ~interface(){delete [] points;delete [] tri;}
        const interface &operator=(const interface &f);
        interface(const interface &f);
        void fileinput();
        void tri_contain(double &x, double &y);
        point get_centre(int &target);
        double pointDistance(point p1, point p2);
        double area(point p1, point p2, point p3);
        void get_integration(double (*foo)(double,double));
};

    interface::interface(const interface &f)
    {
        np=f.np;
        nd=f.nd;
        n_o_t=f.n_o_t;
        points=f.points;
        tri=f.tri;
    }
    const interface &interface ::operator=(const interface &f)
    {
      if(this==&f) return(*this);
        else
        {
        np=f.np;
        nd=f.nd;
        n_o_t=f.n_o_t;
        points=f.points;
        tri=f.tri;
        }
        return *this;
    }
point interface::get_centre(int &target)
        {
            int n1,n2,n3;
            n1=tri[target].n1;
            n2=tri[target].n2;
            n3=tri[target].n3;
            double x1,x2,x3,y1,y2,y3;
            x1=points[n1].x;
            y1=points[n1].y;
            x2=points[n2].x;
            y2=points[n2].y;
            x3=points[n3].x;
            y3=points[n3].y;
            double t1=x1*x1+y1*y1;
            double t2=x2*x2+y2*y2;
            double t3=x3*x3+y3*y3;
            double temp=x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2;
            point centre;
            centre.x=(t2*y3+t1*y2+t3*y1-t2*y1-t3*y2-t1*y3)/temp/2;
            centre.y=(t3*x2+t2*x1+t1*x3-t1*x2-t2*x3-t3*x1)/temp/2;
            return centre;
        }
      double interface::pointDistance(point p1, point p2)
{
        double distance = 0;
        distance = sqrt((p1.y-p2.y)*(p1.y-p2.y)+(p1.x-p2.x)*(p1.x-p2.x));
        return distance;
}
double interface::area(point p1, point p2, point p3)
{
        double area = 0;
        double a = 0, b = 0, c = 0, s = 0;
        a = pointDistance(p1, p2);
        b = pointDistance(p2, p3);
        c = pointDistance(p1, p3);
        s = 0.5*(a+b+c);
        area = sqrt(s*(s-a)*(s-b)*(s-c));
        return area;
}

float sign (point p1, point p2, point p3)
{
        return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
}

bool PointInTriangle (point pt, point v1, point v2, point v3)
{
        bool b1, b2, b3;

        b1 = sign(pt, v1, v2) < 0.0f;
        b2 = sign(pt, v2, v3) < 0.0f;
        b3 = sign(pt, v3, v1) < 0.0f;
      // cout<<b1<<endl;
        return ((b1 == b2) && (b2 == b3));
}
void interface::fileinput()
{
        int cell;
        int i,j;
        f1.open("triangulation#1.TRI");
        f1>>np>>nd>>cell;
        points=new point[np];
        //cout<<np<<endl;
        for (i=0;i<np;i+=1)
        {
                f1>>points[i].id>>points[i].x>>points[i].y>>points[i].z;
                //cout <<points[i].id<<endl;
        }
        int n_d,n_s;
        float extra;
        f1>>n_o_t>>n_d>>n_s;
        cout<<n_o_t<<endl;
        tri=new triangle[n_o_t];
                for (i=0;i<n_o_t;i+=1)
        {
                f1>>tri[i].id>>tri[i].n1>>tri[i].n2>>tri[i].n3;
                for(j=0;j<17;j+=1){f1>>extra;}
        }
        }
        point interface::get_centre(int &target)
                {
                    int n1,n2,n3;
                    n1=tri[target].n1;
                    n2=tri[target].n2;
                    n3=tri[target].n3;
                    double x1,x2,x3,y1,y2,y3;
                    x1=points[n1].x;
                    y1=points[n1].y;
                    x2=points[n2].x;
                    y2=points[n2].y;
                    x3=points[n3].x;
                    y3=points[n3].y;
                    double t1=x1*x1+y1*y1;
                    double t2=x2*x2+y2*y2;
                    double t3=x3*x3+y3*y3;
```

```
            double temp=x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2;
            point centre;
            centre.x=(t2*y3+t1*y2+t3*y1-t2*y1-t3*y2-t1*y3)/temp/2;
            centre.y=(t3*x2+t2*x1+t1*x3-t1*x2-t2*x3-t3*x1)/temp/2;
            return centre;
        }
      double interface::pointDistance(point p1, point p2)
{
        double distance = 0;
        distance = sqrt((p1.y-p2.y)*(p1.y-p2.y)+(p1.x-p2.x)*(p1.x-p2.x));
        return distance;
}
double interface::area(point p1, point p2, point p3)
{
        double area = 0;
        double a = 0, b = 0, c = 0, s = 0;
        a = pointDistance(p1, p2);
        b = pointDistance(p2, p3);
        c = pointDistance(p1, p3);
        s = 0.5*(a+b+c);
        area = sqrt(s*(s-a)*(s-b)*(s-c));
        return area;
}

float sign (point p1, point p2, point p3)
{
        return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
}

bool PointInTriangle (point pt, point v1, point v2, point v3)
{
        bool b1, b2, b3;

        b1 = sign(pt, v1, v2) < 0.0f;
        b2 = sign(pt, v2, v3) < 0.0f;
        b3 = sign(pt, v3, v1) < 0.0f;
      // cout<<b1<<endl;
        return ((b1 == b2) && (b2 == b3));
}
void interface::fileinput()
{
        int cell;
        int i,j;
        f1.open("triangulation#1.TRI");
        f1>>np>>nd>>cell;
        points=new point[np];
        //cout<<np<<endl;
        for (i=0;i<np;i+=1)
        {
                f1>>points[i].id>>points[i].x>>points[i].y>>points[i].z;
                //cout <<points[i].id<<endl;
        }
        int n_d,n_s;
        float extra;
        f1>>n_o_t>>n_d>>n_s;
        cout<<n_o_t<<endl;
        tri=new triangle[n_o_t];
                for (i=0;i<n_o_t;i+=1)
        {
                f1>>tri[i].id>>tri[i].n1>>tri[i].n2>>tri[i].n3;
                for(j=0;j<17;j+=1){f1>>extra;}
        }
        }
point interface::get_centre(int &target)
        {
            int n1,n2,n3;
            n1=tri[target].n1;
            n2=tri[target].n2;
            n3=tri[target].n3;
            double x1,x2,x3,y1,y2,y3;
            x1=points[n1].x;
            y1=points[n1].y;
            x2=points[n2].x;
            y2=points[n2].y;
            x3=points[n3].x;
            y3=points[n3].y;
            double t1=x1*x1+y1*y1;
            double t2=x2*x2+y2*y2;
            double t3=x3*x3+y3*y3;
            double temp=x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2;
            point centre;
            centre.x=(t2*y3+t1*y2+t3*y1-t2*y1-t3*y2-t1*y3)/temp/2;
            centre.y=(t3*x2+t2*x1+t1*x3-t1*x2-t2*x3-t3*x1)/temp/2;
            return centre;
        }
      double interface::pointDistance(point p1, point p2)
{
        double distance = 0;
```

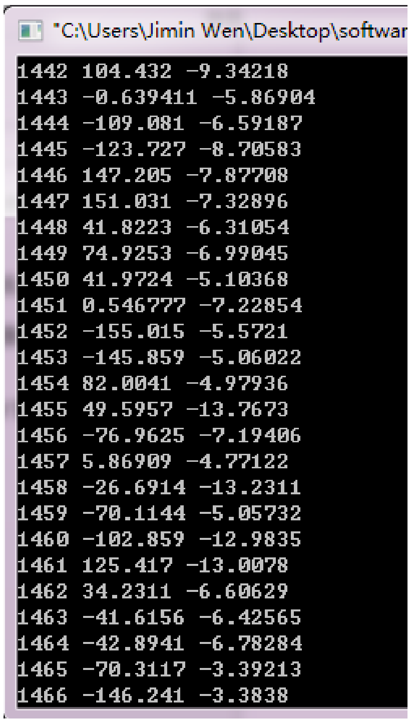
```
	distance = sqrt((p1.y-p2.y)*(p1.y-p2.y)+(p1.x-p2.x)*(p1.x-p2.x));
	return distance;
}
double interface::area(point p1, point p2, point p3)
{
	double area = 0;
	double a = 0, b = 0, c = 0, s = 0;
	a = pointDistance(p1, p2);
	b = pointDistance(p2, p3);
	c = pointDistance(p1, p3);
	s = 0.5*(a+b+c);
	area = sqrt(s*(s-a)*(s-b)*(s-c));
	return area;
}

float sign (point p1, point p2, point p3)
{
	return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
}

bool PointInTriangle (point pt, point v1, point v2, point v3)
{
	bool b1, b2, b3;

	b1 = sign(pt, v1, v2) < 0.0f;
	b2 = sign(pt, v2, v3) < 0.0f;
	b3 = sign(pt, v3, v1) < 0.0f;
      // cout<<b1<<endl;
	return ((b1 == b2) && (b2 == b3));
}
void interface::fileinput()
{
	int cell;
	int i,j;
	f1.open("triangulation#1.TRI");
	f1>>np>>nd>>cell;
	points=new point[np];
	//cout<<np<<endl;
	for (i=0;i<np;i+=1)
	{
		f1>>points[i].id>>points[i].x>>points[i].y>>points[i].z;
		//cout <<points[i].id<<endl;
	}
	int n_d,n_s;
```

```
float extra;
f1>>n_o_t>>n_d>>n_s;
cout<<n_o_t<<endl;
tri=new triangle[n_o_t];
    for (i=0;i<n_o_t;i+=1)
{
    f1>>tri[i].id>>tri[i].n1>>tri[i].n2>>tri[i].n3;
    for(j=0;j<17;j+=1){f1>>extra;}
}
}
```

(a) The program reads the file and store all the data as points and triangles. The read file is "triangulation#1.tri". After all the data is stored in the program, it is output on the screen. Different points and triangles are compared with the original file. There are no mistakes on the stored data. A comparison between the data in the file and the stored data is represented in figure 5.



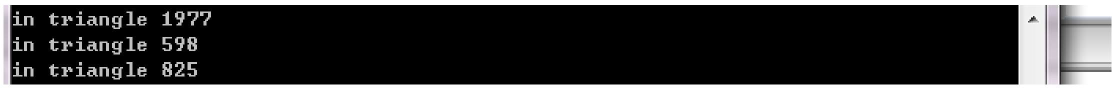
```
1442 104.432 -9.34218 1004
1443 -0.639411 -5.86904 1004
1444 -109.081 -6.59187 1004
1445 -123.727 -8.70583 1004
1446 147.205 -7.87708 1004
1447 151.031 -7.32896 1004
1448 41.8223 -6.31054 1004
1449 74.9253 -6.99045 1004
1450 41.9724 -5.10368 1004
1451 0.546777 -7.22854 1004
1452 -155.015 -5.5721 1004
1453 -145.859 -5.06022 1004
1454 82.0041 -4.97936 1004
1455 49.5957 -13.7673 1004
1456 -76.9625 -7.19406 1004
1457 5.86909 -4.77122 1004
1458 -26.6914 -13.2311 1004
1459 -70.1144 -5.05732 1004
1460 -102.859 -12.9835 1004
1461 125.417 -13.0078 1004
1462 34.2311 -6.60629 1004
1463 -41.6156 -6.42565 1004
1464 -42.8941 -6.78284 1004
1465 -70.3117 -3.39213 1004
1466 -146.241 -3.3838 1004
```



```
1442 104.432 -9.34218
1443 -0.639411 -5.86904
1444 -109.081 -6.59187
1445 -123.727 -8.70583
1446 147.205 -7.87708
1447 151.031 -7.32896
1448 41.8223 -6.31054
1449 74.9253 -6.99045
1450 41.9724 -5.10368
1451 0.546777 -7.22854
1452 -155.015 -5.5721
1453 -145.859 -5.06022
1454 82.0041 -4.97936
1455 49.5957 -13.7673
1456 -76.9625 -7.19406
1457 5.86909 -4.77122
1458 -26.6914 -13.2311
1459 -70.1144 -5.05732
1460 -102.859 -12.9835
1461 125.417 -13.0078
1462 34.2311 -6.60629
1463 -41.6156 -6.42565
1464 -42.8941 -6.78284
1465 -70.3117 -3.39213
1466 -146.241 -3.3838
```

Figure 5: A comparison between the stored data and data in the file.

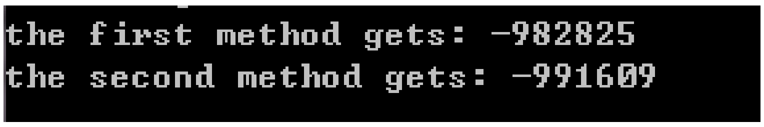
(b) This section is evidenced by proving that a random point can be found contained in correct triangles. The point (33.5101, -3.00017) is chosen as a check point. It is represented that the point is in triangle 1977, 598 and 825. When checking the file, it is found that the result is correct.



```
in triangle 1977
in triangle 598
in triangle 825
```

Figure 6: the output when checking the point

(c) The correctness of this section is proved by proving that results of the two methods are close to each other. The output of the integration when reading the file is represented in figure 5. The same result is found when reading different files. The code is also revised to only output integration of a single triangle, the result meets the calculation with matlab. Thus, the algorithm is proved to be correct.



```
the first method gets: -982825
the second method gets: -991609
```

Figure 7: the output of the integration

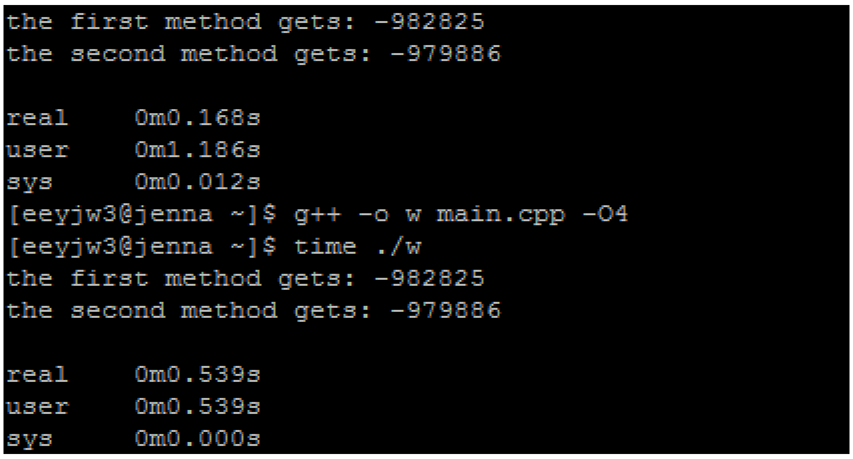
Task5 (the parts with parallel)

```
void interface::tri_contain(double &x, double &y)
    {
    vector<int> array1;
    int i;
    struct point extrapoint;
    extrapoint.x=x;
    extrapoint.y=y;
    bool check=false;
    int v1,v2,v3,target;
    #pragma omp   parallel //shared (tri)
 {
     #pragma omp for private(i,v1,v2,v3,check)
    for (i=0;i<n_o_t;i++)
    {
    v1=tri[i].n1;
    v2=tri[i].n2;
    v3=tri[i].n3;
    check=PointInTriangle (extrapoint, points[v1], points[v2], points[v3]);
    if (check==1)
    {
     #pragma omp critical
     array1.push_back(tri[i].id);
   //   cout<<"point in triangle"<<i<<endl;
    }
    }
    }
   sort(array1.begin(),array1.end());
     for(i==0;i<array1.size();i++)
     {
       cout<<"point in triangle: "<<array1[i]<<endl;
     }
```

```
}
void interface::get_integration(double (*foo)(double,double))
    {
      int i;
      double FO,V1,V2,F1,F2,F3;
      V1=0;
      V2=0;
      point centre;
      int n1,n2,n3;
      double thearea;
      #pragma omp parallel reduction(+:V1)
      {
      #pragma omp    for private(i,n1,n2,n3,centre,FO,thearea)
      for (i=0;i<n_o_t;i++)
      {
      n1=tri[i].n1;
      n2=tri[i].n2;
      n3=tri[i].n3;
      thearea=area(points[n1],points[n2],points[n3]);
      centre=get_centre(i);
      FO=foo(centre.x,centre.y);
      V1=V1+FO*thearea;
      }
      }
      cout<<"the first method gets: "<<V1<<endl;
      double x1,y1,x2,y2,x3,y3;

      #pragma omp parallel reduction(+:V2)
      {
      #pragma omp for private(i,n1,n2,n3,F1,F2,F3,x1,y1,y2,y3,x2,x3,thearea)
      for (i=0;i<n_o_t;i++)
      {
      n1=tri[i].n1;
      n2=tri[i].n2;
      n3=tri[i].n3;
      x1=points[n1].x;
      y1=points[n1].y;
      x2=points[n2].x;
      y2=points[n2].y;
      x3=points[n3].x;
      y3=points[n3].y;
      F1=foo(x1,y1);
      F2=foo(x2,y2);
```

```
        F3=foo(x3,y3);
        thearea=area(points[n1],points[n2],points[n3]);
        V2=V2+(F1+F2+F3)*thearea/3;
        }
        }
        cout<<"the second method gets: "<<V2<<endl;
    }
```

After parallel is added on the program, the program is compiled twice, with and without –fopenmp. It can be found that the openmp consumes only one third the time compared with linear comparing. The obtained result is also correct. To take a further test, the data of the triangles that contain a certain point is output. It is found the the program with and without openmp generate the same result. Hence, the correctness and the efficiency of the openmp can be proved.



```
the first method gets: -982825
the second method gets: -979886

real    0m0.168s
user    0m1.186s
sys     0m0.012s
[eeyjw3@jenna ~]$ g++ -o w main.cpp -O4
[eeyjw3@jenna ~]$ time ./w
the first method gets: -982825
the second method gets: -979886

real    0m0.539s
user    0m0.539s
sys     0m0.000s
```

Figure 8: Comparison between the compiling with and without openmp.

Part 2 header file

```
typedef struct point
{
    int id;
float x;
float y;
} VERTEX2D, *VERTEX2D_PTR;

typedef struct edge
{
point v1;
point v2;
} EDGE, *EDGE_PTR;

typedef struct triangle
{
    int id;
point i1; // vertex index
point i2;
point i3;
    edge e1;
    edge e2;
    edge e3;

} TRIANGLE, *TRIANGLE_PTR;
```

The gernerated images are plotted and be compared with the provided demo results. Figure 9, 11 and 13 are generated from the tri files. It could be found that the generated figures are very similar to the provided results.

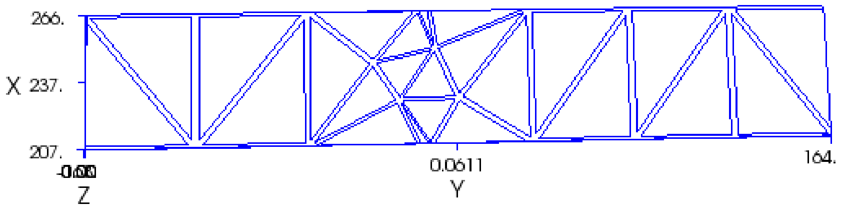


Figure 9: Generated result for vertical#1.tri

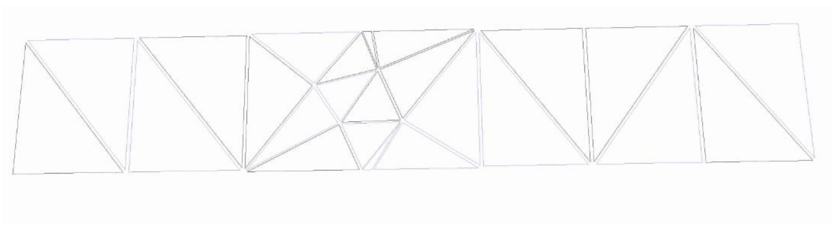


Figure 10: Sample result for vertical#1.tri

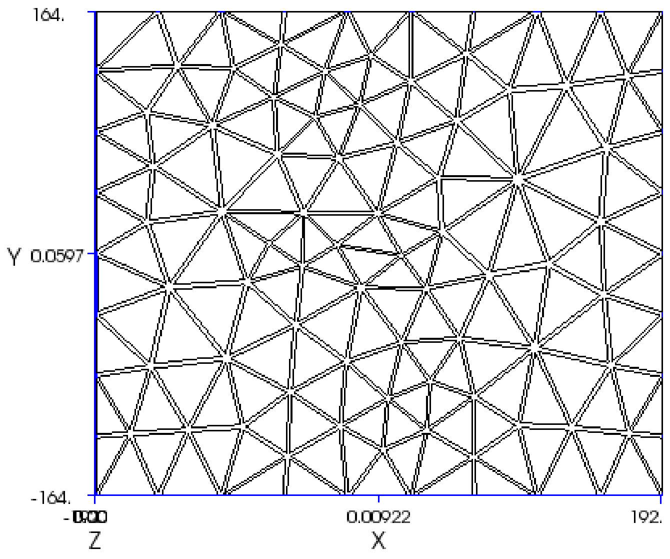


Figure 11: the generated triangulation from vertical#2.tri

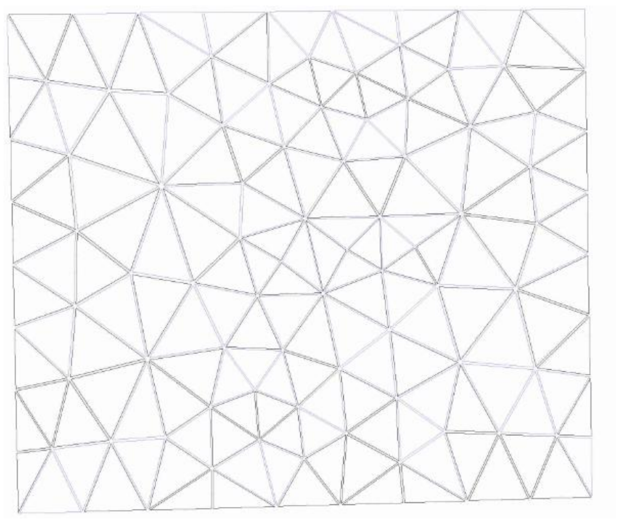


Figure 12: The sample result from vertical#2.tri

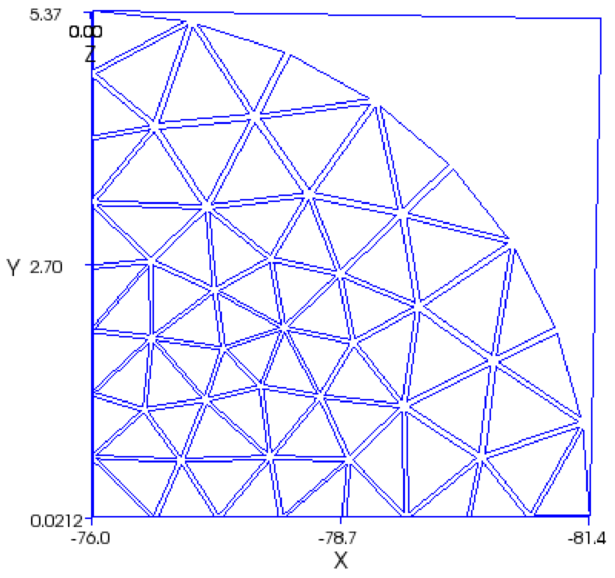


Figure 13: The generated result from vertical#3.tri

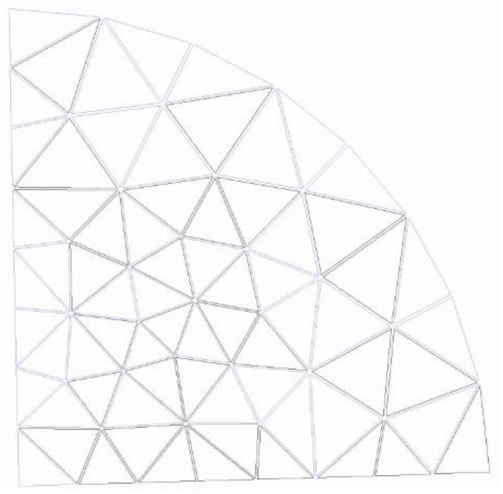


Figure 14: The sample result from vertical#3.tri