UNIVERSITY AT BUFFALO

Fast Virtual Stenting

High Performance Computing Final Project

Nikhil Paliwal

12/17/2013

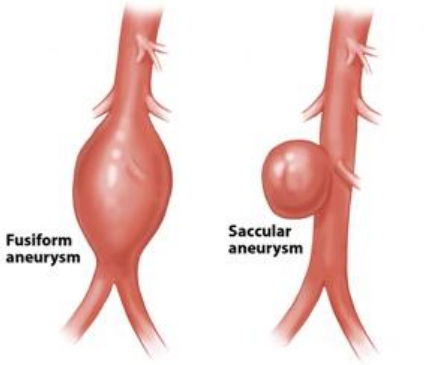
Contents

Introduction

Aneurysm is a brain disorder in which the blood vessel in the brain remodels themselves due to the flow modifications and result in bulging of the blood vessels. It is considered as one of the main reasons for stroke. Research has shown that flow of blood (hemodynamics) and the biological response of the cells (endothelial cells) plays an important role in formation of aneurysm. Studies have shown that aneurysms are usually developed at regions where the flow becomes abnormal, especially at bifurcations of the blood vessels.

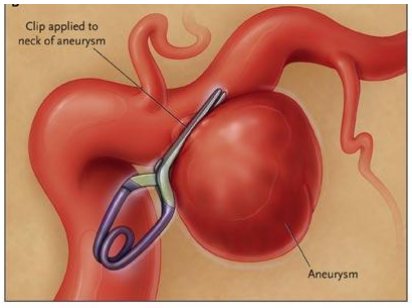
There are mainly two kind of aneurysm found in the blood vessels, namely fusiform and saccular. These are shown in the figure. The flow of blood through this aneurysm might cause the blood vessel to rupture and cause stroke.

The flow of blood and remodeling of the vessel are related to each other. The blood flow is modified, resulting in the different response of the cells and remodeling of the vessel, which further changes the hemodynamics, and this trend continues until the aneurysm actually ruptures.

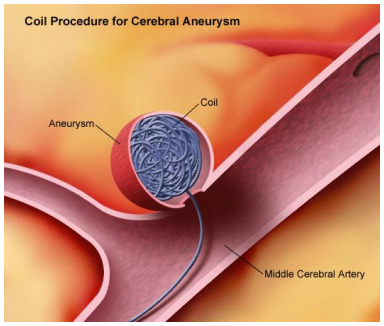


There have been various treatment methods introduced to treat aneurysm. Initially surgical clipping was performed which is done by opening the skull and removing the aneurysm from the flow loop by clipping it. This method is not the most efficient as the skull has to be opened and the brain exposed during the procedure.

Next was the introduction of metallic coils to fill the aneurysm sac with. The coils were mainly made of platinum, and their purpose was to block the blood flow in aneurysm and as a result, the aneurysm would thrombose and clot eventually. There were complications due to this method as well. Coils might get out of the aneurysm and get into the parent vessel, which can cause complications.

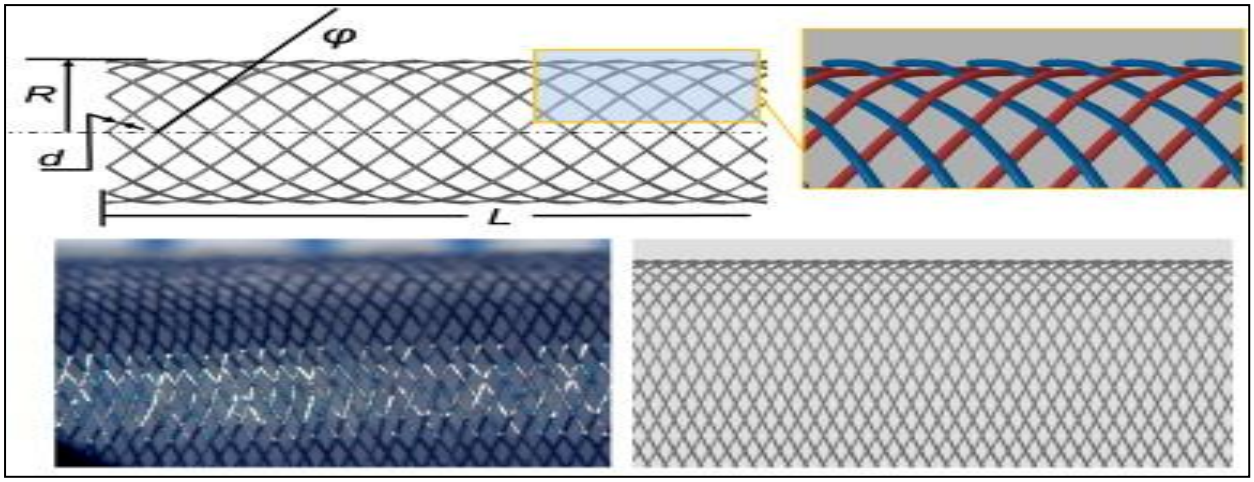


Surgical Clipping



Coiling

After these two methods, stents were introduced to treat aneurysm. Stents are cylindrical porous metallic/alloys wireframe which can be deployed in the main vessel to make sure that flow is going through the parent vessel (parent vessel is the one on which the aneurysm has grown). Different kinds of stents were introduced and the currently surgeons use Flow Diverter (FD). It is made up of platinum and Ni-Co-Cr alloy. 48 wires are braided together in a cylindrical mesh shape. The deployment of these FD makes sure that the flow does not go into the aneurysm and normal hemodynamics is maintained in the blood vessel. This project deals with the virtual deployment of the FD in a blood vessel.



Flow Diverter

Why simulate the FD deployment in patient specific aneurysm?

An important question to consider is why we need to simulate this deployment of FD in a patient on a computer. There are two aspects associated with the proper deployment of a FD. Firstly, surgeons need to make sure that FD is placed correctly inside the blood vessel, and secondly and more importantly, what will be the change in hemodynamics due to placement of FD. There have been cases where FD placement has lead to rupture. There are various theories that try to explain this, but it is necessary to study the hemodynamics after the placement of FD inside the blood vessel. This is where the simulations can help. FD can be deployed virtually in the blood vessel and then CFD can help tell the flow of blood inside the blood vessel, and surgeons can decide whether placement of FD is favorable or not. An alternative method can be making use of some coils as well. This method is called stent-assisted coiling. Hence the simulations can help answer a lot of questions before actually treating the patient.

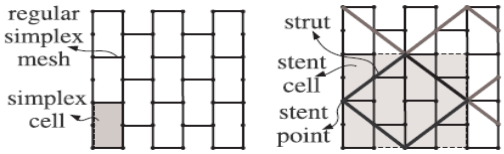
Background

There have been methods developed that deploy FD in blood vessel and CFD is performed. One of them is the HiFiVS method[2] which takes use of Finite Element based method and run on the software ABAQUS. It is a very accurate method but one major disadvantage is the time taken for each simulation. It takes around 120 hours for each simulation to run and hence cannot be used in a large number of clinical cases. In order to counter this problem a new method is developed called the Fast Virtual Stenting (FVS) method. This method is relatively quick but is not as accurate as the HiFiVS method.

Fast Virtual Stenting[3]

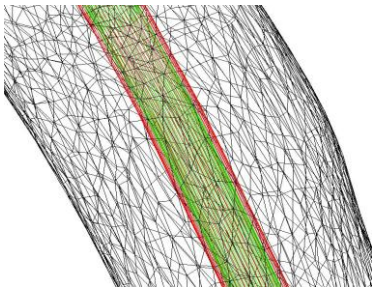
This method simplifies the stent structure into a simplex mesh. A simplex mesh can be considered as a background mesh that is used for the main simulation, and then FD is mapped onto the simplex mesh structure.

Simplex mesh



A simplex mesh for FD can be considered as a 64 concentric circles, with 48 points on each circle connected alternately as shown in the figure on the right. Once cell consists of 4 simplex points, and the connectivity between these points decide the mapping of FD on the simplex mesh. In total a simplex mesh has 64X48 points. This simplex mesh is initially placed in the vessel along the centerline of the vessel.

Parent vessel



Parent vessel is input in the form of triangle mesh, with an average vessel consisting of anywhere between 4000 to 15000 triangles. The spatial coordinates of each point in these triangles is known and stored in a stereolithography file (.stl). This information can be imported by writing a simple code. The figure shows a part of the vessel and the simplex mesh in red and green color. The same stl file can be used to get the centerline of the blood vessel using the open source software virtual modeling tool kit (vmtk). This centerline is used to align the simplex mesh along the vessel centerline.

Method

The simplex mesh is placed inside the blood vessel as shown in the above figure and then

$$\rho \frac{\partial^2 \mathbf{p}_i(t)}{\partial t^2} + \gamma \frac{\partial \mathbf{p}_i(t)}{\partial t} - \alpha \mathbf{f}_{int}(\mathbf{p}_i(t)) = \beta \mathbf{f}_{ext}(\mathbf{p}_i(t))$$

expanded using a mathematical force model based on the equation given below:

This is the dynamic equation for the position p_i of the simplex mesh point within the blood vessel. The first term is the acceleration term and second term in the equation is the damping term. These define the motion of the point. The f_{int} anf f_{ext} terms are the forces among the simplex mesh points and between simplex mesh and blood vessel respectively. Internal forces are calculated based on 2 parameters: F-angle, F-length. F-length is the forces that arise from the relative positioning of the simplex mesh points. Similarly F-angle changes due to the change in the angle between the simplex points. External forces come from the distance of the mesh from the vessel.

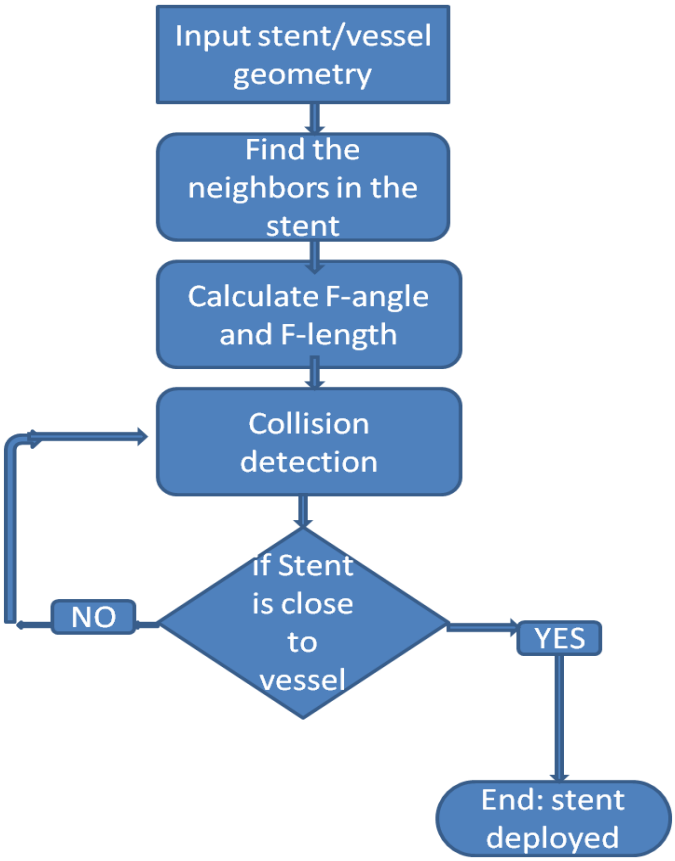
The above equation is discretized and the position of each stent point is calculated with respect to the previous position. Once the mesh starts expanding, it has to stop before it touches the wall. This tolerance to touch the wall is very small and is usually around 0.03 mm, equal to the radius of the actual FD wires. This way once the simulation is completed, the wires can be extruded to the given radius and the final FD will touch the vessel wall.

To make sure the mesh does not go outside the vessel wall, a **collision detection** algorithm is also implemented. This makes sure that the stent at no time goes out of the vessel. For this the

distance of each simplex point to the triangles in the vessel wall is calculated. **This is the most time consuming part of the method, and takes more than 90% of the actual computational time. This project aims at looking at the collision detection part of the code and testing the efficiency of the collision detection of the code in FORTRAN using Open-MP and in MATLAB parallel.**

Algorithm

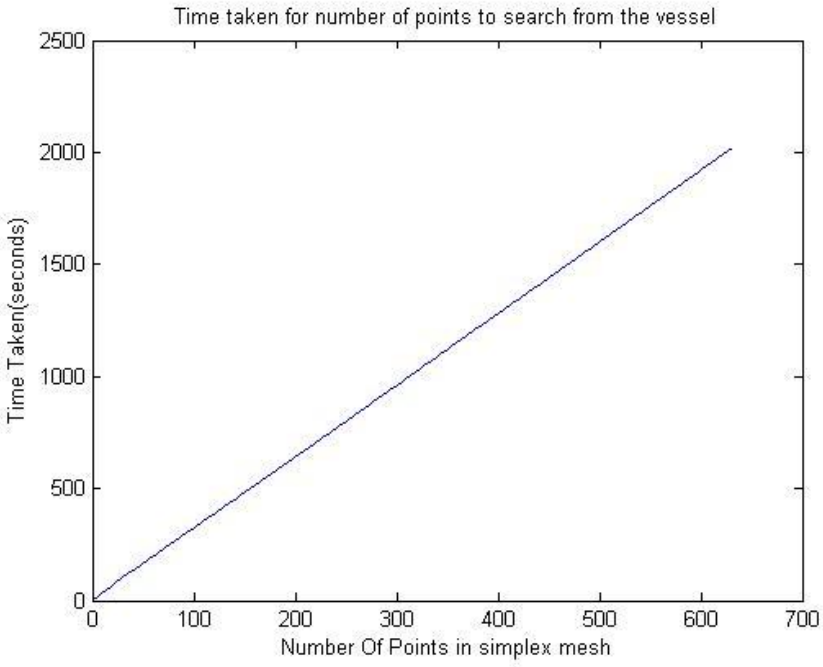
The algorithm can be best explained by the use of this flow chart. This will help understand the main code. As explained previously, collision detection step is the most time consuming part of this algorithm.



Algorithm for collision detection takes each of the 64X48 stent points and calculated the distance of each point with every triangle in the vessel, and gives the distance from the nearest triangle. This part of the code can be parallelized which would increase the efficiency of the whole code tremendously. For this specific project, this part will be coded in FORTRAN and compared with MATLAB sequential as well as MATLAB parallel code.

MATLAB Sequential

The code is attached in the appendix 1. The profiling time of this code is shown in the following figure. As expected, the time taken by the code increases with increasing the number of triangles in the search. This is due to the sequential task of considering one triangle at a time and then moving on. The number of triangles in the particular vessel of consideration in this case is 8037.

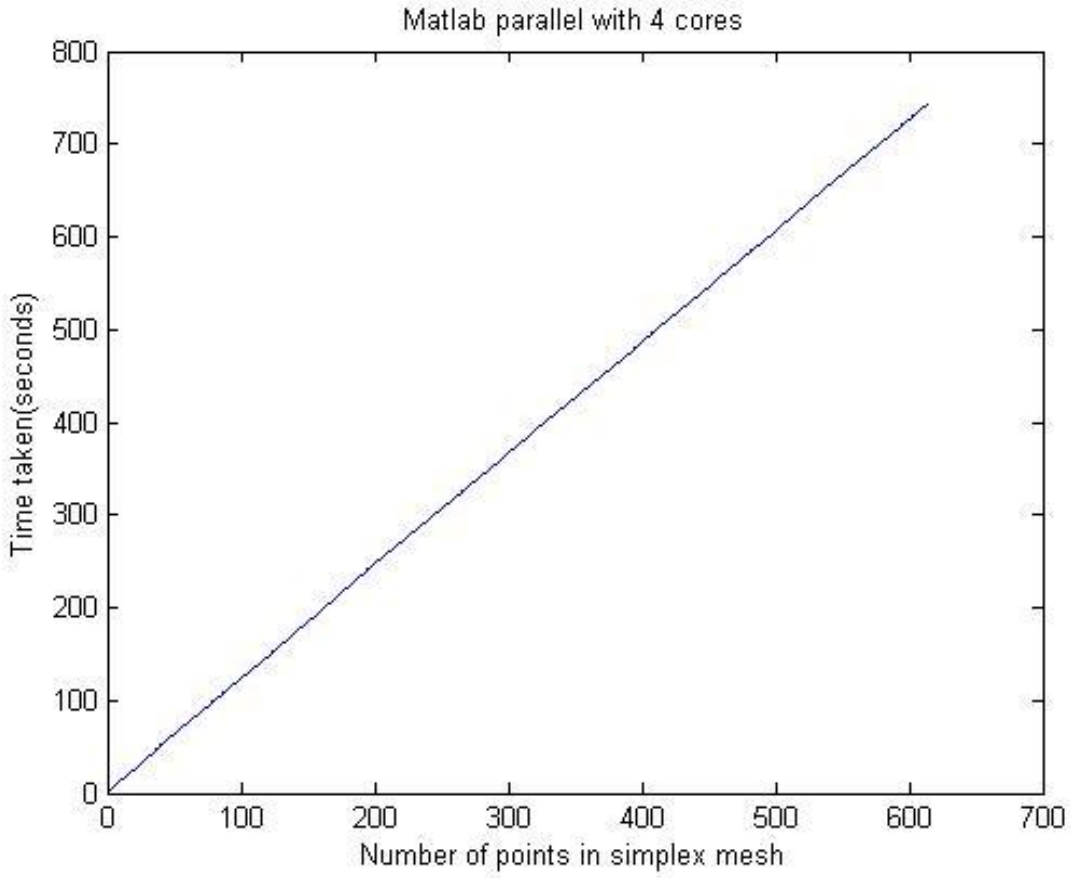


This result gives enough evidence to apply knowledge of parallel computing to this particular code. For this project only the collision detection part of the code is parallelized since the remaining part takes less than a second to execute, and there is no point in parallelizing that part of the code. Parallelizing the entire code might also reduce the efficiency, since it might take

time in communication which will slow down the code.

MATLAB Parallel

Parallel MATLAB code is attached in appendix 2. The changes from the MATLAB sequential is the use of 12-core for the calculation of the collision detection. Only one for loop is modified to parfor (parallel for loop) for parallel computing. The code was run on a quad-core local computer in the Toshiba Stroke and Vascular Research Center (TSVRC) in downtown campus. The functions like sqrt in MATLAB can automatically run on parallel processors in MATLAB so there is no need to modify that part of the code in MATLAB. MATLAB parallel works more like Open-MP for now, hence it can run on a 12-core processor and the processes are shared within the cores of a single node on



CCR. Matlabpool requests the number of cores user want to use in their code.

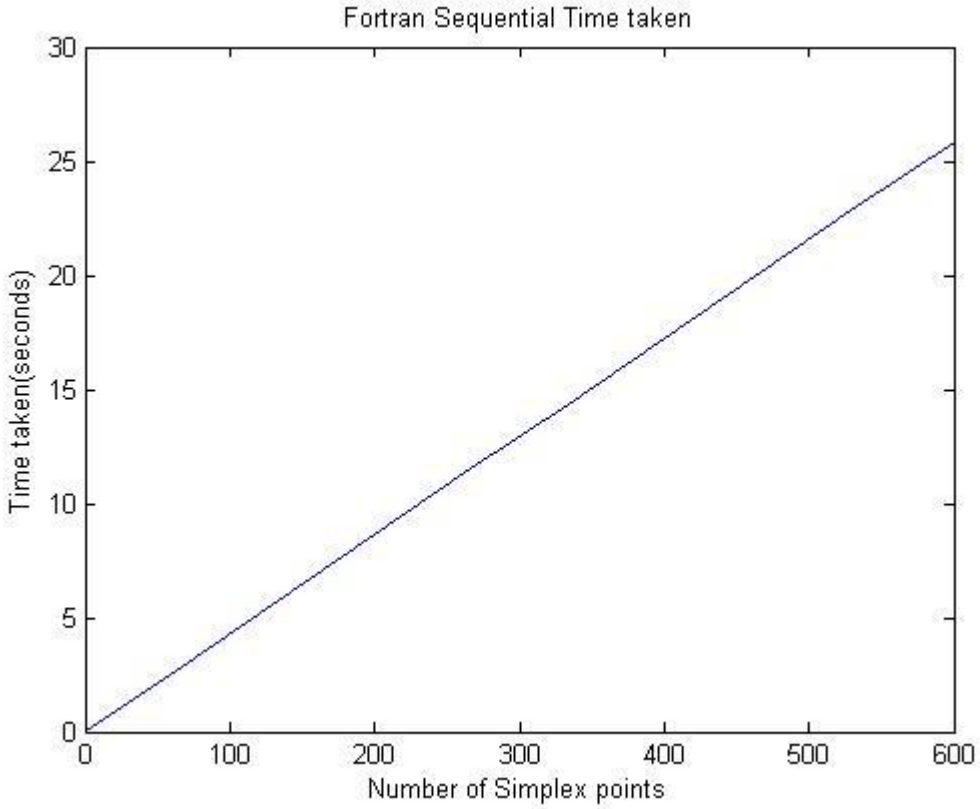
It is quite clear with the above figure that even with 4 cores in MATLAB, the efficiency of the code goes up a lot. It has gone for roughly 390 seconds for 600 points as compared to 2000 seconds for the same number of points.

For further computation, UB CCR server was used for both MATLAB and FORTRAN.

FORTRAN with OPEN MP

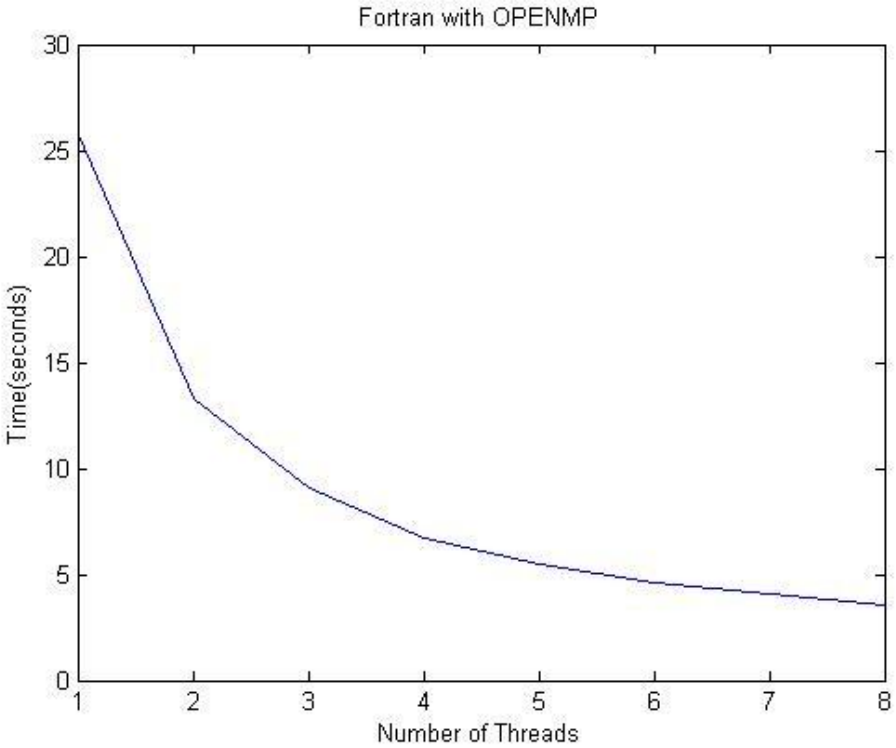
For this project purpose, FORTRAN code is developed and tested for the collision detection part only. The point triangle distance algorithm is taken from the source [1]. There is a single code in

appendix 3 for FORTRAN with OPENMP where sequential code can be obtained from setting Nthreads equal to 1. Data from stl file is imported from MATLAB, and saved in file 'data.dat' and simplex mesh points are saved in 'simplex.dat'. Node k08n18s01 received OPENMP parallel job.



Results

Most important thing was to make sure that the final deployment of the stent makes sense or not. Result obtained is shown in appendix 4 and is consistent across codes.



Clearly FORTRAN is quicker than the MATLAB code, for the same number of simplex points (600 points used to search in 8037 triangles). It is also interesting to note that MATLAB code

converges at 10 cores and does not improve performance on further increasing the number of cores.

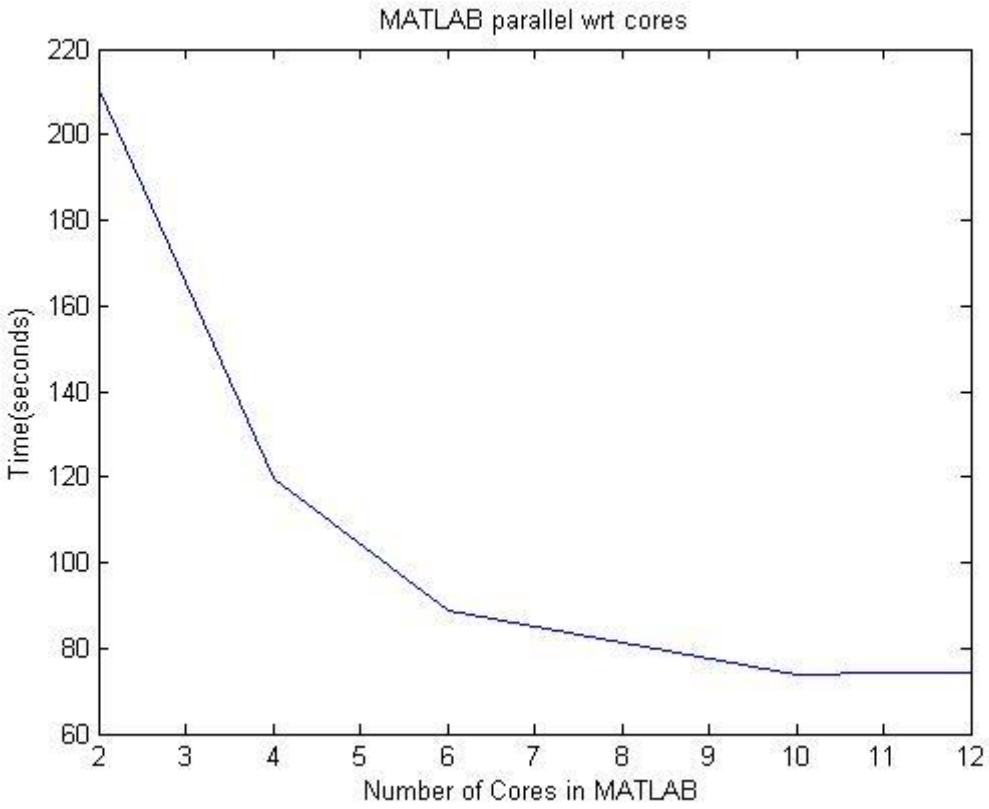
Parallel Speedup

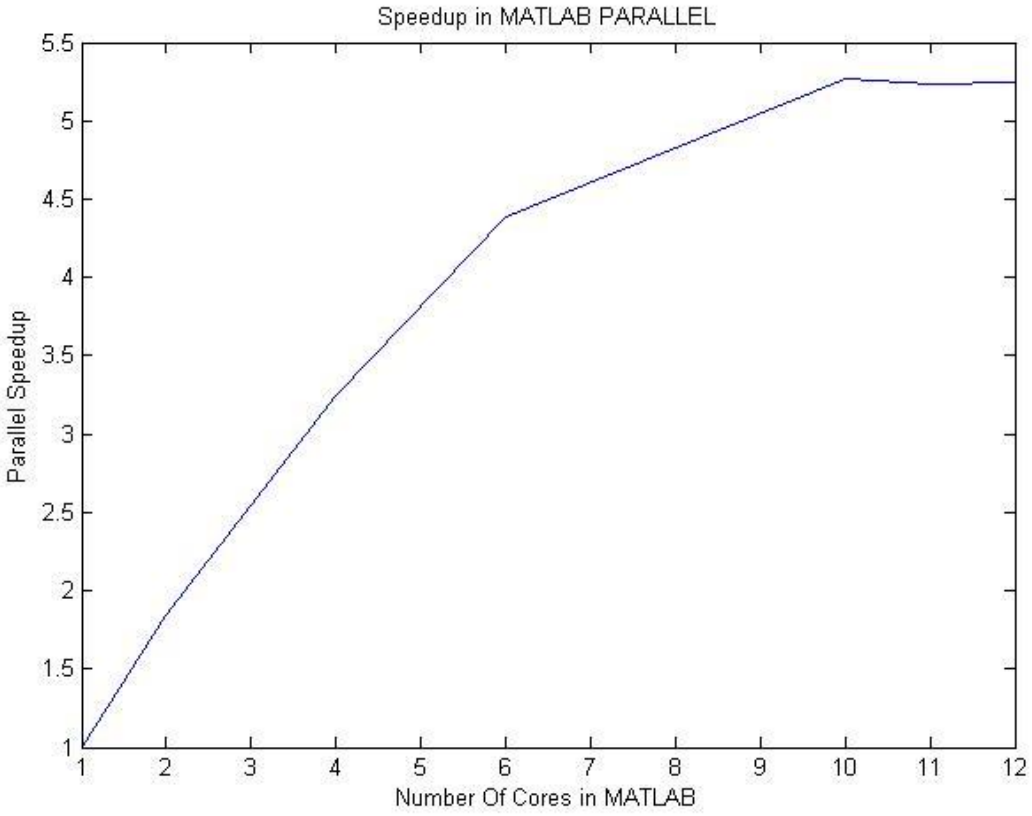
In order to calculate parallel speedup, the time taken for 1 thread was used as the sequential time taken and put in the following equation to get the speedup:

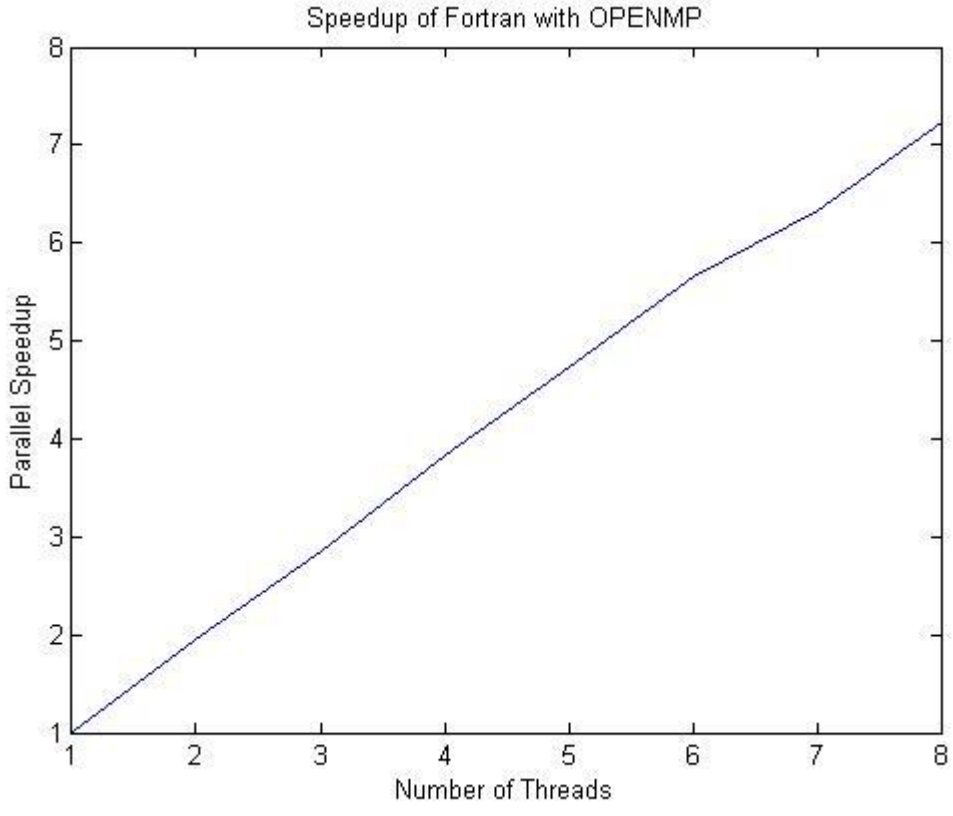
$$Parallel\ Speedup = \frac{Time\ taken\ for\ 1\ thread}{Taken\ for\ n\ threads}$$

The results for parallel speedup are as follows:

MATLAB parallel gives a speedup of around 5.25 and then its speedup does not go up, where as in OPENMP, speedup is more than 7 and it will increase if we increase the number of threads used in calculation. This also gives us a very good indication that parallelizing the code in FORTRAN will surely speedup the code performance wise.




Speedup in MATLAB PARALLEL
Parallel Speedup
Number Of Cores in MATLAB
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
1
2
3
4
5
6
7
8
9
10
11
12

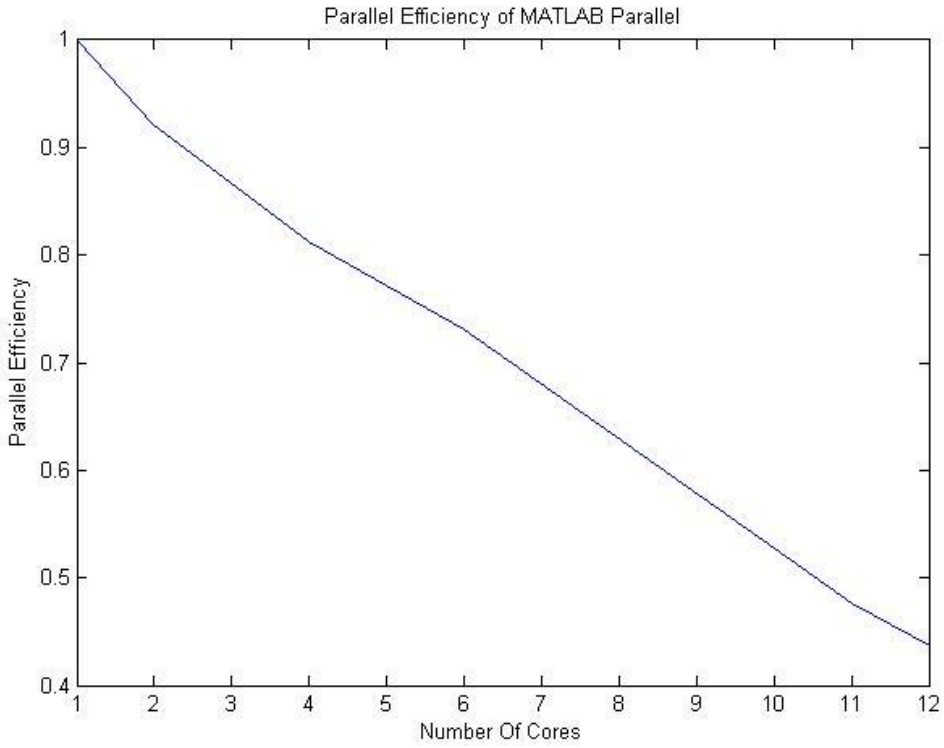

Speedup of Fortran with OPENMP
Parallel Speedup
Number of Threads
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8

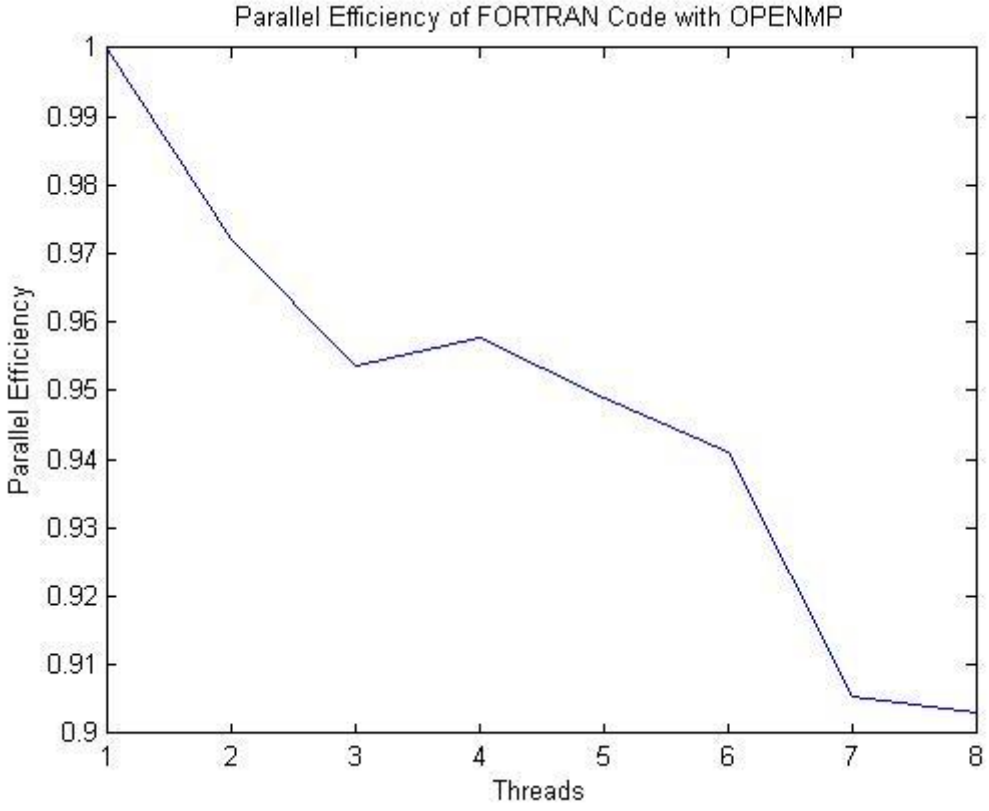
Parallel Efficiency

Efficiency is defined as the speedup per thread, so the equation for parallel efficiency becomes:

$$Parallel\ Efficiency = \frac{1}{P} * \frac{Time\ taken\ for\ 1\ thread}{Taken\ for\ n\ threads}$$

Where P is the number of threads/cores





Discussion

This project provided some very important insight on the importance of high performance computing in the particular field of bioengineering. It is also quite clear that FORTRAN is way superior language than MATLAB when it comes to larger calculations and better parallel speedups can be achieved by parallelizing the code with OPENMP. Having said that, the algorithms for calculating the point triangle distance used in both the codes are different and they might also play a crucial role in eventually efficiency of each codes. A final result is shown in the table below:

Code	Time Taken(seconds)
MATLAB Sequnetial	388.65
MATLAB Parallel with 4 cores	119.65
FORTRAN Sequential	25.86
FORTRAN with OPENMP 4 Threads	6.75

These times are all based on running on UB's CCR server.

Hence this project gives a clear idea that the code needs to be converted into FORTRAN and parallelized with OPENMP for high performance.

Future Directions

This project has been very instrumental in increasing the parallel speedup of the engineering tool used for virtual deployment of the stent inside the aneurysm. The project sets upa stone for future directions in the project. First thing, the whole code should be written in FORTRAN and parallelized with OPENMP. Comparing OPENMP with MPI might also be useful for efficiency of the code. Once this is done, this code will provide a very important tool for clinicians to use and apply in their surgical decision-making on treating patients with aneurysm.

References

1. *http://orion.math.iastate.edu/burkardt/papers/fortran_arrays.html#Dynamic_Array_Allocation*, December 10, 2013
2. *High Fidelity Virtual Stenting (HiFiVS) for Intracranial Aneurysm Flow Diversion: In Vitro and In Silico* Ding Ma, et. al. , Annals of Biomedical Engineering, October 2013, Volume 41, Issue 10, pp 2143-2156
3. *Fast virtual deployment of self-expandable stents: method and in vitro evaluation for intracranial aneurysmal stenting*.Larrabide I, et.al, Med Image Anal. 2012 Apr;16(3):721-30
4. HPC Lecture Slides

Appendix

Appendix 1

Main.m

```
clc;
 clear;
load lijianyicutl1
load stent_03
elapsed_time=[];
number_of_points=[];
raw=48;
slice=length(meshpoint1(:,1,:));
length_c=0.37*2;
arfa=0.1;
beta=0.1;
ax=0.4;
epsl=0.6;
scale=0.5;
meshpoint=meshpoint1;

          tic
for n=1:1
    n
    % Find the 3 neighboring nodes (Ñ°ÕÒ3ÁÚµã)
    [point1,point2,point3]=findneighbor(slice,raw,meshpoint);
    % Calculate angle force and strut length force (¼ÆËã½Ç¶È°Í³¤¶ÈÔ¼ÊøÁ¦)
    [flength,fangle]=com_flength_fangle_ling(meshpoint,length_c,raw,slice);

    for i=1:slice
        for j=1:raw
            stentpoint(1:3)=meshpoint(i,j,1:3);

           distance=zeros(face_num,1);
            for k=1:face_num

        tri=[node_xyz(:,(k-1)*3+1)';node_xyz(:,(k-1)*3+2)';node_xyz(:,(k-
1)*3+3)'];

   distance(k)=pointTriangleDistance(tri,stentpoint);
            end
%             end
            d=min(distance);
            elapsed_time(end+1)=toc
            number_of_points(end+1)=(i-1)*48 + j;

              sign=1;
              stent_center_distance=sqrt((meshpoint(i,j,1)-center(i,1))...
                 *(meshpoint(i,j,1)-center(i,1))+(meshpoint(i,j,2)-...
                 center(i,2))*(meshpoint(i,j,2)-center(i,2))+...
                 (meshpoint(i,j,3)-center(i,3))*(meshpoint(i,j,3)...
                 -center(i,3)));

             % Calculate the expansion coefficient
             if  stent_center_distance>0
                dx=(meshpoint(i,j,1)-center(i,1))/stent_center_distance;
                dy=(meshpoint(i,j,2)-center(i,2))/stent_center_distance;
                dz=(meshpoint(i,j,3)-center(i,3))/stent_center_distance;
```

```
            else

                dx=0;
                dy=0;
                dz=0;
            end
            % Update the position of stent point
            newmeshpoint(i,j,1)=meshpoint(i,j,1)+arfa*(point1(i,j,1)/3+...
                point2(i,j,1)/3+point3(i,j,1)/3-meshpoint(i,j,1)+...
                ax*flength(i,j,1)+epsl*fangle(i,j,1))+beta*scale*...
                meshpoint(i,j,1)*sign*dx;

            newmeshpoint(i,j,2)=meshpoint(i,j,2)+arfa*(point1(i,j,2)/3+...
               point2(i,j,2)/3+point3(i,j,2)/3-meshpoint(i,j,2)+...
               ax*flength(i,j,2)+epsl*fangle(i,j,2))+beta*scale*...
               meshpoint(i,j,2)*sign*dy;

            newmeshpoint(i,j,3)=meshpoint(i,j,3)+arfa*(point1(i,j,3)/3+...
               point2(i,j,3)/3+point3(i,j,3)/3-meshpoint(i,j,3)+...
               ax*flength(i,j,3)+epsl*fangle(i,j,3))+beta*scale*...
               meshpoint(i,j,3)*sign*dz;




        end
    end
meshpoint=newmeshpoint;

end
```

pointTriangeDistance.m

```
function [dist,PP0] = pointTriangleDistance(TRI,P)
% calculate distance between a point and a triangle in 3D
% SYNTAX
%   dist = pointTriangleDistance(TRI,P)
%   [dist,PP0] = pointTriangleDistance(TRI,P)
%
% DESCRIPTION
%   Calculate the distance of a given point P from a triangle TRI.
%   Point P is a row vector of the form 1x3. The triangle is a matrix
%   formed by three rows of points TRI = [P1;P2;P3] each of size 1x3.
%   dist = pointTriangleDistance(TRI,P) returns the distance of the point P
%   to the triangle TRI.
%   [dist,PP0] = pointTriangleDistance(TRI,P) additionally returns the
%   closest point PP0 to P on the triangle TRI.
%
% Author: Gwendolyn Fischer
% Release: 1.0
% Release date: 09/02/02
% Release: 1.1 Fixed Bug because of normalization
% Release: 1.2 Fixed Bug because of typo in region 5 20101013
% Release: 1.3 Fixed Bug because of typo in region 2 20101014

% Possible extention could be a version tailored not to return the distance
% and additionally the closest point, but instead return only the closest
% point. Could lead to a small speed gain.
```

```
% Example:
% %% The Problem
% P0 = [0.5 -0.3 0.5];
%
% P1 = [0 -1 0];
% P2 = [1  0 0];
% P3 = [0  0 0];
%
% vertices = [P1; P2; P3];
% faces = [1 2 3];
%
% %% The Engine
% [dist,PP0] = pointTriangleDistance([P1;P2;P3],P0);
%
% %% Visualization
% [x,y,z] = sphere(20);
% x = dist*x+P0(1);
% y = dist*y+P0(2);
% z = dist*z+P0(3);
%
% figure
% hold all
% patch('Vertices',vertices,'Faces',faces,'FaceColor','r','FaceAlpha',0.8);
% plot3(P0(1),P0(2),P0(3),'b*');
% plot3(PP0(1),PP0(2),PP0(3),'*g')
% surf(x,y,z,'FaceColor','b','FaceAlpha',0.3)
% view(3)

% The algorithm is based on
% "David Eberly, 'Distance Between Point and Triangle in 3D',
% Geometric Tools, LLC, (1999)"
% http:\\www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf
%
%        ^t
%  \     |
%   \reg2|
%    \   |
%     \  |
%      \ |
%       \|
%        *P2
%        |\
%        | \
%  reg3  |  \ reg1
%        |   \
%        |reg0\
%        |     \
%        |      \ P1
% -------*-------*------->s
%        |P0      \
%  reg4  | reg5    \ reg6


%% Do some error checking
% if nargin<2
%   error('pointTriangleDistance: too few arguments see help.');
% end
% P = P(:)';
% if size(P,2)~=3
```

```
%   error('pointTriangleDistance: P needs to be of length 3.');
% end
%
% if size(TRI)~=[3 3]
%   error('pointTriangleDistance: TRI needs to be of size 3x3.');
% end

% ToDo: check for colinearity and/or too small triangles.


% rewrite triangle in normal form
B = TRI(1,:);
E0 = TRI(2,:)-B;
%E0 = E0/sqrt(sum(E0.^2)); %normalize vector
E1 = TRI(3,:)-B;
%E1 = E1/sqrt(sum(E1.^2)); %normalize vector


D = B - P;
a = dot(E0,E0);
b = dot(E0,E1);
c = dot(E1,E1);
d = dot(E0,D);
e = dot(E1,D);
f = dot(D,D);

det = a*c - b*b; % do we have to use abs here?
s   = b*e - c*d;
t   = b*d - a*e;

% Terible tree of conditionals to determine in which region of the diagram
% shown above the projection of the point into the triangle-plane lies.
if (s+t) <= det
  if s < 0
    if t < 0
      %region4
      if (d < 0)
        t = 0;
        if (-d >= a)
          s = 1;
          sqrDistance = a + 2*d + f;
        else
          s = -d/a;
          sqrDistance = d*s + f;
        end
      else
        s = 0;
        if (e >= 0)
          t = 0;
          sqrDistance = f;
        else
          if (-e >= c)
            t = 1;
            sqrDistance = c + 2*e + f;
          else
            t = -e/c;
            sqrDistance = e*t + f;
          end
        end
      end %of region 4
```

```
    else
      % region 3
      s = 0;
      if e >= 0
        t = 0;
        sqrDistance = f;
      else
        if -e >= c
          t = 1;
          sqrDistance = c + 2*e +f;
        else
          t = -e/c;
          sqrDistance = e*t + f;
        end
      end
    end %of region 3
  else
    if t < 0
      % region 5
      t = 0;
      if d >= 0
        s = 0;
        sqrDistance = f;
      else
        if -d >= a
          s = 1;
          sqrDistance = a + 2*d + f;% GF 20101013 fixed typo d*s ->2*d
        else
          s = -d/a;
          sqrDistance = d*s + f;
        end
      end
    else
      % region 0
      invDet = 1/det;
      s = s*invDet;
      t = t*invDet;
      sqrDistance = s*(a*s + b*t + 2*d) ...
                  + t*(b*s + c*t + 2*e) + f;
    end
  end
else
  if s < 0
    % region 2
    tmp0 = b + d;
    tmp1 = c + e;
    if tmp1 > tmp0 % minimum on edge s+t=1
      numer = tmp1 - tmp0;
      denom = a - 2*b + c;
      if numer >= denom
        s = 1;
        t = 0;
        sqrDistance = a + 2*d + f; % GF 20101014 fixed typo 2*b -> 2*d
      else
        s = numer/denom;
        t = 1-s;
        sqrDistance = s*(a*s + b*t + 2*d) ...
                    + t*(b*s + c*t + 2*e) + f;
      end
    else           % minimum on edge s=0
      s = 0;
```

```
    if tmp1 <= 0
      t = 1;
      sqrDistance = c + 2*e + f;
    else
      if e >= 0
        t = 0;
        sqrDistance = f;
      else
        t = -e/c;
        sqrDistance = e*t + f;
      end
    end
  end %of region 2
else
  if t < 0
    %region6
    tmp0 = b + e;
    tmp1 = a + d;
    if (tmp1 > tmp0)
      numer = tmp1 - tmp0;
      denom = a-2*b+c;
      if (numer >= denom)
        t = 1;
        s = 0;
        sqrDistance = c + 2*e + f;
      else
        t = numer/denom;
        s = 1 - t;
        sqrDistance = s*(a*s + b*t + 2*d) ...
                    + t*(b*s + c*t + 2*e) + f;
      end
    else
      t = 0;
      if (tmp1 <= 0)
          s = 1;
          sqrDistance = a + 2*d + f;
      else
        if (d >= 0)
            s = 0;
            sqrDistance = f;
        else
            s = -d/a;
            sqrDistance = d*s + f;
        end
      end
    end
    %end region 6
  else
    % region 1
    numer = c + e - b - d;
    if numer <= 0
      s = 0;
      t = 1;
      sqrDistance = c + 2*e + f;
    else
      denom = a - 2*b + c;
      if numer >= denom
        s = 1;
        t = 0;
        sqrDistance = a + 2*d + f;
      else
```

```
            s = numer/denom;
            t = 1-s;
            sqrDistance = s*(a*s + b*t + 2*d) ...
                        + t*(b*s + c*t + 2*e) + f;
          end
        end %of region 1
      end
    end
end

% account for numerical round-off error
if (sqrDistance < 0)
  sqrDistance = 0;
end

dist = sqrt(sqrDistance);

% if nargout>1
%   PP0 = B + s*E0 + t*E1;
% end
```

Findneighbor.m

```
function [point1,point2,point3]=findneighbor(slice,raw,meshpoint)


% Search 3 neighboring points of the stent point (Ñ°ÕÒÖ§¼ÜµãµÄ3ÁÚµã)
for i=1:slice
    for j=1:raw
                if i==1 % the 1st row (µÚ1ÐÐ)
                     if j==1
                         point1(i,j,1:3)=meshpoint(i,raw,:);
                     else
                          point1(i,j,1:3)=meshpoint(i,j-1,:);
                     end

                    if j==raw
                          point2(i,j,1:3)=meshpoint(i,1,:);
                    else
                          point2(i,j,1:3)=meshpoint(i,j+1,:);
                    end

                    point3(i,j,1:3)=meshpoint(i+1,j,:);
                end

                if i==slice % the last row (µÚÄ©ÐÐ)
                    if j==1
                       point1(i,j,1:3)=meshpoint(i,raw,:);
                    else
                       point1(i,j,1:3)=meshpoint(i,j-1,:);
                    end

                    if j==raw
                       point2(i,j,1:3)=meshpoint(i,1,:);
                    else
                       point2(i,j,1:3)=meshpoint(i,j+1,:);
                    end
```

```
                    point3(i,j,1:3)=meshpoint(i-1,j,:);
                end

                % the rest rows (from 2 to the last 2nd row) (ÆäÓà2-
µ¹ÊýµÚ¶þÐÐ)
                if i>=2 && i<=slice-1
                   if mod(i,2)==0
                      if mod(j,2)==0 % even rows and even columns (Ë«ÊýÐÐ
Ë«ÊýÁÐ)
                         if j==raw
                             point3(i,j,1:3)=meshpoint(i,1,:);
                         else
                            point3(i,j,1:3)=meshpoint(i,j+1,:);
                         end
                        point1(i,j,1:3)=meshpoint(i-1,j,:);
                        point2(i,j,1:3)=meshpoint(i+1,j,:);
                      else  % even rows and odd columns (Ë«ÊýÐÐ µ¥ÊýÁÐ)
                        if j==1
                            point3(i,j,1:3)=meshpoint(i,raw,:);
                        else
                          point3(i,j,1:3)=meshpoint(i,j-1,:);
                        end
                        point1(i,j,1:3)=meshpoint(i-1,j,:);
                        point2(i,j,1:3)=meshpoint(i+1,j,:);
                      end

                else  % odd rows (µ¥ÊýÐÐ)
                   if  mod(j,2)==0  % odd rows and even columns (µ¥ÊýÐÐ
Ë«ÊýÁÐ)
                         if j==1
                            point3(i,j,1:3)=meshpoint(i,raw,:);
                         else
                            point3(i,j,1:3)=meshpoint(i,j-1,:);
                         end
                         point1(i,j,1:3)=meshpoint(i-1,j,:);
                         point2(i,j,1:3)=meshpoint(i+1,j,:);

                    else  % odd rows and odd columns (µ¥ÊýÐÐ µ¥ÊýÁÐ)
                       if j==raw
                             point3(i,j,1:3)=meshpoint(i,1,:);
                       else
                            point3(i,j,1:3)=meshpoint(i,j+1,:);
                       end
                       point1(i,j,1:3)=meshpoint(i-1,j,:);
                       point2(i,j,1:3)=meshpoint(i+1,j,:);
                    end
                end
            end

    end
end
   % Finish searching 3 neighboring points (Ñ°ÕÒ3ÁÚµã½áÊø)
```

Com_flength_fangle_ling.m

```
function [flength,fangle]=
com_flength_fangle_ling(meshpoint,length_c,raw,slice)


%Initialization
flength(1:slice,1:raw,1:3)=0;
fangle(1:slice,1:raw,1:3)=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Search neighboring points on the 1st row and calculate f
i=1;
n=2; %number of neighboring points
  for j=1:4:raw-3
      neighbor2=meshpoint(i+2,j+2,:);
      if j==1
         neighbor1=meshpoint(i+2,raw-1,:);
      else
         neighbor1=meshpoint(i+2,j-2,:);
      end
  % Distance between the point and neighboring point 1
      dneighbor1=sqrt((meshpoint(i,j,1)-neighbor1(1))*    (meshpoint(i,j,1)-
...
                  neighbor1(1))+(meshpoint(i,j,2)-neighbor1(2))*...
                  (meshpoint(i,j,2)-neighbor1(2))+(meshpoint(i,j,3)-...
                  neighbor1(3))*(meshpoint(i,j,3)-neighbor1(3)));
      % Distance between the point and neighboring point 2
      dneighbor2=sqrt((meshpoint(i,j,1)-neighbor2(1))*    (meshpoint(i,j,1)-
...
                  neighbor2(1))+(meshpoint(i,j,2)-neighbor2(2))*...
                  (meshpoint(i,j,2)-neighbor2(2))+(meshpoint(i,j,3)-...
                  neighbor2(3))*(meshpoint(i,j,3)-neighbor2(3)));

      % Calculate flength
      flength(i,j,:)=((length_c-dneighbor1)*(meshpoint(i,j,:)-
neighbor1)/dneighbor1+...
          (length_c-dneighbor2)*(meshpoint(i,j,:)-neighbor2)/dneighbor2)/n;

      pangle12=(neighbor1+neighbor2)/2;
      dpangle12=sqrt((meshpoint(i,j,1)-pangle12(1))* (meshpoint(i,j,1)-...
                  pangle12(1))+(meshpoint(i,j,2)-pangle12(2))*...
                  (meshpoint(i,j,2)-pangle12(2))+(meshpoint(i,j,3)-...
                  pangle12(3))*(meshpoint(i,j,3)-pangle12(3)));
     angle12=2*acos(dpangle12/dneighbor1);

     % Calculate fangle
     fangle(i,j,:)=-abs(dneighbor1*cos(angle12))*(meshpoint(i,j,:)-
pangle12)/dpangle12;
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Search neighboring points on the last row and calculate f
i=slice;
n=2; % Number of neighboring points
  for j=1:4:raw-3
      neighbor2=meshpoint(i-2,j+2,:);
      if j==1
         neighbor1=meshpoint(i-2,raw-1,:);
      else
```

```
        neighbor1=meshpoint(i-2,j-2,:);
      end
 % Distance between the point and neighboring point 1
    dneighbor1=sqrt((meshpoint(i,j,1)-neighbor1(1))* (meshpoint(i,j,1)-...
                neighbor1(1))+(meshpoint(i,j,2)-neighbor1(2))*...
                (meshpoint(i,j,2)-neighbor1(2))+(meshpoint(i,j,3)-...
                neighbor1(3))*(meshpoint(i,j,3)-neighbor1(3)));
    % Distance between the point and neighboring point 2
    dneighbor2=sqrt((meshpoint(i,j,1)-neighbor2(1))* (meshpoint(i,j,1)-...
                neighbor2(1))+(meshpoint(i,j,2)-neighbor2(2))*...
                (meshpoint(i,j,2)-neighbor2(2))+(meshpoint(i,j,3)-...
                neighbor2(3))*(meshpoint(i,j,3)-neighbor2(3)));

     % Calculate fangle
     flength(i,j,:)=((length_c-dneighbor1)*(meshpoint(i,j,:)-
neighbor1)/dneighbor1+...
        (length_c-dneighbor2)*(meshpoint(i,j,:)-neighbor2)/dneighbor2)/n;


     pangle12=(neighbor1+neighbor2)/2;
     dpangle12=sqrt((meshpoint(i,j,1)-pangle12(1))* (meshpoint(i,j,1)-...
                pangle12(1))+(meshpoint(i,j,2)-pangle12(2))*...
                (meshpoint(i,j,2)-pangle12(2))+(meshpoint(i,j,3)-...
                pangle12(3))*(meshpoint(i,j,3)-pangle12(3)));
     angle12=2*acos(dpangle12/dneighbor1);

     % Calculate fangle (¼ÆËãfangle)
     fangle(i,j,:)=-abs(dneighbor1*cos(angle12))*(meshpoint(i,j,:)-
pangle12)/dpangle12;
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Search neighboring points on the 3rd, 7th, 11th, ..., (slice-2)th row and
calculate f
%
n=4;
for i=3:4:slice-2
   for j=3:4:raw-1
     if j==raw-1
              neighbor2=meshpoint(i-2,1,:);
              neighbor4=meshpoint(i+2,1,:);
     else
             neighbor2=meshpoint(i-2,j+2,:);
             neighbor4=meshpoint(i+2,j+2,:);
     end
     neighbor1=meshpoint(i-2,j-2,:);
     neighbor3=meshpoint(i+2,j-2,:);

     % Distance between the point and neighboring point 1
     dneighbor1=sqrt((meshpoint(i,j,1)-neighbor1(1))* (meshpoint(i,j,1)-...
              neighbor1(1))+(meshpoint(i,j,2)-neighbor1(2))*...
              (meshpoint(i,j,2)-neighbor1(2))+(meshpoint(i,j,3)-...
              neighbor1(3))*(meshpoint(i,j,3)-neighbor1(3)));
     % Distance between the point and neighboring point 2
     dneighbor2=sqrt((meshpoint(i,j,1)-neighbor2(1))* (meshpoint(i,j,1)-...
              neighbor2(1))+(meshpoint(i,j,2)-neighbor2(2))*...
              (meshpoint(i,j,2)-neighbor2(2))+(meshpoint(i,j,3)-...
              neighbor2(3))*(meshpoint(i,j,3)-neighbor2(3)));
     % Distance between the point and neighboring point 3
     dneighbor3=sqrt((meshpoint(i,j,1)-neighbor3(1))* (meshpoint(i,j,1)-...
```

```
            neighbor3(1))+(meshpoint(i,j,2)-neighbor3(2))*...
            (meshpoint(i,j,2)-neighbor3(2))+(meshpoint(i,j,3)-...
            neighbor3(3))*(meshpoint(i,j,3)-neighbor3(3)));
    % Distance between the point and neighboring point 4
    dneighbor4=sqrt((meshpoint(i,j,1)-neighbor4(1))* (meshpoint(i,j,1)-...
            neighbor4(1))+(meshpoint(i,j,2)-neighbor4(2))*...
            (meshpoint(i,j,2)-neighbor4(2))+(meshpoint(i,j,3)-...
            neighbor4(3))*(meshpoint(i,j,3)-neighbor4(3)));

    % Calculate flength
    flength(i,j,:)=((length_c-dneighbor1)*(meshpoint(i,j,:)-
neighbor1)/dneighbor1+...
    (length_c-dneighbor2)*(meshpoint(i,j,:)-neighbor2)/dneighbor2+...
    (length_c-dneighbor3)*(meshpoint(i,j,:)-neighbor3)/dneighbor3+...
    (length_c-dneighbor4)*(meshpoint(i,j,:)-neighbor4)/dneighbor4)/n;



    pangle12=(neighbor1+neighbor2)/2;
    pangle13=(neighbor1+neighbor3)/2;
    pangle24=(neighbor2+neighbor4)/2;
    pangle34=(neighbor3+neighbor4)/2;

    dpangle12=sqrt((meshpoint(i,j,1)-pangle12(1))* (meshpoint(i,j,1)-...
            pangle12(1))+(meshpoint(i,j,2)-pangle12(2))*...
            (meshpoint(i,j,2)-pangle12(2))+(meshpoint(i,j,3)-...
            pangle12(3))*(meshpoint(i,j,3)-pangle12(3)));
    dpangle13=sqrt((meshpoint(i,j,1)-pangle13(1))* (meshpoint(i,j,1)-...
            pangle13(1))+(meshpoint(i,j,2)-pangle13(2))*...
            (meshpoint(i,j,2)-pangle13(2))+(meshpoint(i,j,3)-...
            pangle13(3))*(meshpoint(i,j,3)-pangle13(3)));
    dpangle24=sqrt((meshpoint(i,j,1)-pangle24(1))* (meshpoint(i,j,1)-...
            pangle24(1))+(meshpoint(i,j,2)-pangle24(2))*...
            (meshpoint(i,j,2)-pangle24(2))+(meshpoint(i,j,3)-...
            pangle24(3))*(meshpoint(i,j,3)-pangle24(3)));
   dpangle34=sqrt((meshpoint(i,j,1)-pangle34(1))* (meshpoint(i,j,1)-...
            pangle34(1))+(meshpoint(i,j,2)-pangle34(2))*...
            (meshpoint(i,j,2)-pangle34(2))+(meshpoint(i,j,3)-...
            pangle34(3))*(meshpoint(i,j,3)-pangle34(3)));

    angle12=2*acos(dpangle12/dneighbor1);
    angle13=2*acos(dpangle12/dneighbor1);
    angle24=2*acos(dpangle12/dneighbor2);
    angle34=2*acos(dpangle12/dneighbor3);

    % Calculate fangle
    fangle(i,j,:)=-(abs(dneighbor1*cos(angle12))*(meshpoint(i,j,:)-
pangle12)/...
    dpangle12+abs(dneighbor1*cos(angle13))*(meshpoint(i,j,:)-pangle13)/...
    dpangle13+abs(dneighbor2*cos(angle24))*(meshpoint(i,j,:)-pangle24)/...
    dpangle24-abs(dneighbor3*cos(angle34))*(meshpoint(i,j,:)-pangle34)/...
       dpangle34)/n;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Search neighboring points on the 5th, 9th, 13th, ..., (slice-4)th row and
calculate f
%
```

```
n=4;
for i=5:4:slice-4
   for j=1:4:raw-3
     if j==1
              neighbor1=meshpoint(i-2,raw-1,:);
              neighbor3=meshpoint(i+2,raw-1,:);
     else
             neighbor1=meshpoint(i-2,j-2,:);
             neighbor3=meshpoint(i+2,j-2,:);
     end
     neighbor2=meshpoint(i-2,j+2,:);
     neighbor4=meshpoint(i+2,j+2,:);

     % Distance between the point and neighboring point 1
     dneighbor1=sqrt((meshpoint(i,j,1)-neighbor1(1))* (meshpoint(i,j,1)-...
              neighbor1(1))+(meshpoint(i,j,2)-neighbor1(2))*...
              (meshpoint(i,j,2)-neighbor1(2))+(meshpoint(i,j,3)-...
              neighbor1(3))*(meshpoint(i,j,3)-neighbor1(3)));
     % Distance between the point and neighboring point 2
     dneighbor2=sqrt((meshpoint(i,j,1)-neighbor2(1))* (meshpoint(i,j,1)-...
              neighbor2(1))+(meshpoint(i,j,2)-neighbor2(2))*...
              (meshpoint(i,j,2)-neighbor2(2))+(meshpoint(i,j,3)-...
              neighbor2(3))*(meshpoint(i,j,3)-neighbor2(3)));
     % Distance between the point and neighboring point 3
    dneighbor3=sqrt((meshpoint(i,j,1)-neighbor3(1))* (meshpoint(i,j,1)-...
              neighbor3(1))+(meshpoint(i,j,2)-neighbor3(2))*...
              (meshpoint(i,j,2)-neighbor3(2))+(meshpoint(i,j,3)-...
              neighbor3(3))*(meshpoint(i,j,3)-neighbor3(3)));
     % Distance between the point and neighboring point 4
     dneighbor4=sqrt((meshpoint(i,j,1)-neighbor4(1))* (meshpoint(i,j,1)-...
              neighbor4(1))+(meshpoint(i,j,2)-neighbor4(2))*...
              (meshpoint(i,j,2)-neighbor4(2))+(meshpoint(i,j,3)-...
              neighbor4(3))*(meshpoint(i,j,3)-neighbor4(3)));

     % Calculate flength
     flength(i,j,:)=((length_c-dneighbor1)*(meshpoint(i,j,:)-
neighbor1)/dneighbor1+...
     (length_c-dneighbor2)*(meshpoint(i,j,:)-neighbor2)/dneighbor2+...
     (length_c-dneighbor3)*(meshpoint(i,j,:)-neighbor3)/dneighbor3+...
     (length_c-dneighbor4)*(meshpoint(i,j,:)-neighbor4)/dneighbor4)/n;


     pangle12=(neighbor1+neighbor2)/2;
     pangle13=(neighbor1+neighbor3)/2;
     pangle24=(neighbor2+neighbor4)/2;
     pangle34=(neighbor3+neighbor4)/2;

     dpangle12=sqrt((meshpoint(i,j,1)-pangle12(1))* (meshpoint(i,j,1)-...
              pangle12(1))+(meshpoint(i,j,2)-pangle12(2))*...
              (meshpoint(i,j,2)-pangle12(2))+(meshpoint(i,j,3)-...
              pangle12(3))*(meshpoint(i,j,3)-pangle12(3)));
     dpangle13=sqrt((meshpoint(i,j,1)-pangle13(1))* (meshpoint(i,j,1)-...
              pangle13(1))+(meshpoint(i,j,2)-pangle13(2))*...
              (meshpoint(i,j,2)-pangle13(2))+(meshpoint(i,j,3)-...
              pangle13(3))*(meshpoint(i,j,3)-pangle13(3)));
     dpangle24=sqrt((meshpoint(i,j,1)-pangle24(1))* (meshpoint(i,j,1)-...
              pangle24(1))+(meshpoint(i,j,2)-pangle24(2))*...
              (meshpoint(i,j,2)-pangle24(2))+(meshpoint(i,j,3)-...
              pangle24(3))*(meshpoint(i,j,3)-pangle24(3)));
    dpangle34=sqrt((meshpoint(i,j,1)-pangle34(1))* (meshpoint(i,j,1)-...
```

```
                pangle34(1))+(meshpoint(i,j,2)-pangle34(2))*...
                (meshpoint(i,j,2)-pangle34(2))+(meshpoint(i,j,3)-...
                pangle34(3))*(meshpoint(i,j,3)-pangle34(3)));

    angle12=2*acos(dpangle12/dneighbor1);
    angle13=2*acos(dpangle12/dneighbor1);
    angle24=2*acos(dpangle12/dneighbor2);
    angle34=2*acos(dpangle12/dneighbor3);

    % Calculate fangle
    fangle(i,j,:)=-(abs(dneighbor1*cos(angle12))*(meshpoint(i,j,:)-
pangle12)/...
     dpangle12+abs(dneighbor1*cos(angle13))*(meshpoint(i,j,:)-pangle13)/...
     dpangle13+abs(dneighbor2*cos(angle24))*(meshpoint(i,j,:)-pangle24)/...
     dpangle24-abs(dneighbor3*cos(angle34))*(meshpoint(i,j,:)-pangle34)/...
        dpangle34)/n;
   end
end
```

Appendix 2

Main.m for parallel MATLAB

```
clc;
 clear;
matlabpool 4
load lijianyicutl1
load stent_03
elapsed_time=[];
number_of_points=[];
raw=48;
slice=length(meshpoint1(:,1,:));
length_c=0.37*2;
arfa=0.1;
beta=0.1;
ax=0.4;
epsl=0.6;
scale=0.5;
meshpoint=meshpoint1;

          tic
for n=1:1
    n
    % Find the 3 neighboring nodes (Ñ°ÕÒ3ÁÚµã)
    [point1,point2,point3]=findneighbor(slice,raw,meshpoint);
    % Calculate angle force and strut length force (¼ÆËã½Ç¶È°Í³¤¶ÈÔ¼ÊøÁ¦)
    [flength,fangle]=com_flength_fangle_ling(meshpoint,length_c,raw,slice);

    for i=1:slice
        for j=1:raw
            stentpoint(1:3)=meshpoint(i,j,1:3);

           distance=zeros(face_num,1);
            parfor k=1:face_num

        tri=[node_xyz(:,(k-1)*3+1)';node_xyz(:,(k-1)*3+2)';node_xyz(:,(k-
1)*3+3)'];
```

```
    distance(k)=pointTriangleDistance(tri,stentpoint);
            end
%             end
            d=min(distance);
            elapsed_time(end+1)=toc
            number_of_points(end+1)=(i-1)*48 + j;

              sign=1;
              stent_center_distance=sqrt((meshpoint(i,j,1)-center(i,1))...
                 *(meshpoint(i,j,1)-center(i,1))+(meshpoint(i,j,2)-...
                 center(i,2))*(meshpoint(i,j,2)-center(i,2))+...
                 (meshpoint(i,j,3)-center(i,3))*(meshpoint(i,j,3)...
                 -center(i,3)));

             % Calculate the expansion coefficient
             if  stent_center_distance>0
                dx=(meshpoint(i,j,1)-center(i,1))/stent_center_distance;
                dy=(meshpoint(i,j,2)-center(i,2))/stent_center_distance;
                dz=(meshpoint(i,j,3)-center(i,3))/stent_center_distance;
             else

                dx=0;
                dy=0;
                dz=0;
             end
            % Update the position of stent point
            newmeshpoint(i,j,1)=meshpoint(i,j,1)+arfa*(point1(i,j,1)/3+...
                point2(i,j,1)/3+point3(i,j,1)/3-meshpoint(i,j,1)+...
                ax*flength(i,j,1)+epsl*fangle(i,j,1))+beta*scale*...
                meshpoint(i,j,1)*sign*dx;

            newmeshpoint(i,j,2)=meshpoint(i,j,2)+arfa*(point1(i,j,2)/3+...
               point2(i,j,2)/3+point3(i,j,2)/3-meshpoint(i,j,2)+...
               ax*flength(i,j,2)+epsl*fangle(i,j,2))+beta*scale*...
               meshpoint(i,j,2)*sign*dy;

            newmeshpoint(i,j,3)=meshpoint(i,j,3)+arfa*(point1(i,j,3)/3+...
               point2(i,j,3)/3+point3(i,j,3)/3-meshpoint(i,j,3)+...
               ax*flength(i,j,3)+epsl*fangle(i,j,3))+beta*scale*...
               meshpoint(i,j,3)*sign*dz;




        end
    end
meshpoint=newmeshpoint;

end
matlabpool close;
```

Appendix 3

```
program PIomp
Use omp_lib
implicit none

real   :: mysum,pi,x1,y1,z1,x2,y2,z2,x3,y3,z3,x,y,z,dist
real, dimension(3,24111) :: coordinates
real, dimension(3,600) :: simplex
real, dimension(24112/3) :: distance
real, dimension(600) :: D
double precision   :: t_start, t_end, tot_time
integer                                          ::    Mflops,    myid,Np,Nt,
Nperprocess,triangle,a,stent,temp,iteration
integer            ::i,j,k,N, ntr
integer, dimension(8) :: Nthreads
open(1, file='data.dat')
open(2, file='simplex.dat')
read(1,*)coordinates
read(2,*)simplex


Nthreads=(/ 1,2,3,4,5,6,7,8/)
do ntr=1,size(Nthreads)
Nt=Nthreads(ntr)
!print *, Nt
call OMP_SET_NUM_THREADS(Nt)



temp=0
 myid = OMP_GET_THREAD_NUM()
 Nt = omp_get_num_threads()
 Np=omp_get_num_procs()
 Nperprocess= N/Nt
  t_start = OMP_GET_WTIME()
!$OMP PARALLEL DO
do iteration=1,100
do stent=1,600
do triangle=1,(24112/3)

a=(triangle-1)*3
x1=coordinates(1,a + 1)
y1=coordinates(2,a + 1)
z1=coordinates(3,a + 1)
x2=coordinates(1,a + 2)
y2=coordinates(2,a + 2)
z2=coordinates(3,a + 2)
x3=coordinates(1,a + 3)
y3=coordinates(2,a + 3)
z3=coordinates(3,a + 3)
x=simplex(1,stent)
y=simplex(2,stent)
z=simplex(3,stent)

   call triangle_point_dist_3d(x1,y1,z1,x2,y2,z2,x3,y3,z3,x,y,z,dist)
   temp=temp+1
distance(triangle)=dist
```

```
end do
D(stent)=minval(distance)
!print *,D(stent)
end do
end do
!$OMP END PARALLEL DO
!print *,temp


 t_end = OMP_GET_WTIME()
 tot_time = t_end - t_start
print *,tot_time, Nthreads(ntr)
 Mflops = int(real((N*10)*1000)/(1000000*tot_time))

!print *, N, Nt, tot_time
!write(1,*)N, Nt, tot_time



end do

end program
```

triangle_point_dist_3d taken from reference 1.

Slurm file for CCR:

```
#!/bin/sh
##SBATCH --partition=general-compute
#SBATCH --time=01:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
##SBATCH --mem=24000
# Memory per node specification is in MB. It is optional.
# The default limit is 3GB per core.
#SBATCH --job-name="hpc_project"
#SBATCH --output=hpc_project.out
#SBATCH --mail-user=npaliwal@buffalo.edu
#SBATCH --mail-type=END
##SBATCH --requeue
#Specifies that the job will be requeued after a node failure.
#The default is that the job will not be requeued.


echo "SLURM_JOBID="$SLURM_JOBID
echo "SLURM_JOB_NODELIST"=$SLURM_JOB_NODELIST
echo "SLURM_NNODES"=$SLURM_NNODES
echo "SLURMTMPDIR="$SLURMTMPDIR

cd $SLURM_SUBMIT_DIR
echo "working directory = "$SLURM_SUBMIT_DIR

module load intel/13.0
module load intel-mpi/4.1.0
module list
ulimit -s unlimited
#
echo "Launch helloworld with srun"

srun ./main

echo "All Done!"
```

Appendix 4