EE4371 Data Structures and Algorithms

Fluid in thermal equilibrium problem

Sanjana S Prabhu -EE17B072

December 10, 2019

1 Introduction

In this assignment, we have to analyse the behaviour of fluid particles, in a 2m*2m two-dimensional box. The force between two interactive particles is given by:

$$\bar{F} = \frac{x\hat{x} + y\hat{y}}{(x^2 + y^2)^{3/2}} \tag{1}$$

2 Algorithm

After the initialisation of the positions and velocities of the particles, the following algorithm is followed:

- Since, initially we are considering all particles' impact on every particle, we have to calculate the distance between every pair of particles.
- Compute the distance between that particular particle and each of the other 10^8 particles(with the particle of interest as the origin), and determine the force in the x and y directions respectively. Put a negative sign since the force is repulsive.
- Assuming all the particles to be of unit mass, find the acceleration in the x and y directions. Compute the new velocities, and the displacement in both directions using the following formulae:

$$v = u + at \tag{2}$$

$$s = s_i + ut + 1/2at^2 \tag{3}$$

  where t=10^{-4} and s_i stands for the initial position.
- Suppose any of the new positions are greater than 1 or lesser than -1(out of boundaries), then reverse the velocities in the direction in which the out of boundary condition has occurred.
- Now repeat this 10^4 times, since time interval is 10^{-4} seconds, and the time is 1 second.

2.1 Estimation of time required for the above algorithm

- We initially need to compute the x and y(distance between every two particles), therefore two floating point calculations are involved, which can be done parallely as they are independent, so totally 1 FPC. For each force calculation, there are 6 floating point calculations(2 squaring, 1 addition, 1 raised to the power of 3/2, and finally two divisions corresponding to the forces in x and y directions). Therefore 7 FPCs for each pair.

- To compute the total force on a particle, we need to perform these 7 FPCs, 10^8-1 times, and add all these values together(which is again 10^8-2), however addition can be done in parallel with the computation of force. Therefore , it approximates to $7*2*10^8$ FPCs.

- To move each particle we need to compute the new velocity and the new position.

- New velocity computation involves 1 multiplication and 1 addition = 2 FPCs.

- New position computation involves 2 multiplications and 1 addition(since s_i and ut can be added parallely while computing the other term)= 3 FPCs.

- Since, this is done step by step in C, we require 5 FPCs.

- To check boundary conditions, we need to make 1 comparison, which is one FPC and the rest are integer operations and assignments which are done parallelly.

- So the time taken roughly for one particle in one time step is 14*10^8 FPCs= 14 seconds.

- So for this particular CPU, the time taken for this problem would be **14 seconds**, for each particle, for each time-step, if we consider the effect of all the particles.

3 Solving the problem using trees

We can use a 4-way tree by converting the x and y coordinates into a signed binary fraction, and segregating the particles based on the 4 possible variations of the x and y coordinate bits upto 10 bits and the sign bit.

3.1 Pseudocode

```
struct node * Newnode( struct node * Node, struct particle *p
int condition, int num_dig)
begin
switch (condition)
begin
case  1 :
if(num_dig==10)
    begin
p=(struct node*)malloc(sizeof(struct node));
Node->last_nodes[a]=p;
Node->number_of_children=a+1;
p=(struct node*)malloc(sizeof(struct node));
if (Node->child[0]==NULL)
Node->child[0]=p;
else
p=Node->child[0];
p->num_dig=num_dig;
a++;
return NULL;}
p=(struct node*)malloc(sizeof(struct node));
if (Node->child[0]==NULL)
Node->child[0]=p;
else
p=Node->child[0];
p->num_dig=num_dig;

return p;
break;
(similarly for all four conditions)
        end
    end
end

struct node* SearchParent (struct node *root, struct Particle *p)
begin
for(i=0;i<11;i++)
begin
if(p->px[i]==0&&p->py[i]==0)
j=0;
else if(p->px[i]==1&&p->py[i]==0)
j=1;
```

```
else if(p->px[i]==0&&p->py[i]==1)
j=2;
else if(p->px[i]==1&&p->py[i]==1)
j=3;
if((root->child[j])->num_dig==10)
return root;
else
root=root->child[j];
end
end

void FindForce (struct node *Parent, struct Particle *p, float *fx, float *fy)
{
int i;
float x,y;
int p1;
p1=Parent->number_of_children;// number of leaves of the parent(closest particles)
for(i=0;i<p1;i++)
{
x=-(((Parent->last_nodes[i])->pxd)-(p->pxd));// distance b/w them in x direction
y=-(((Parent->last_nodes[i])->pyd)-(p->pyd));// distance b/w them in y direction
(*fx)+=x/pow((x*x+y*y),1.5);// force b/w them in x direction
(*fy)+=y/pow((x*x+y*y),1.5);// force b/w them in y direction
}

}

void decimal_to_binary(float p, struct Particle *p1, char coord)
begin
if(coord=='x')
{
int i;
if(p>0)// sign bit(MSB)
(p1->px)[0]=0;
else
(p1->px)[0]=1;

for (i=1;i<=10;i++)// conversion upto 10 bits
{
p=2*p;
if(p>1)
{
(p1->px)[i]=1;
p=p-1;
```

```
}
else
(p1->px)[i]=0;

}
}
else
{
int i;
if(p>0)
(p1->py)[0]=0;
else
(p1->py)[0]=1;

for (i=1;i<=10;i++)
{
p=2*p;
if(p>1)
{
(p1->py)[i]=1;
p=p-1;
}
else
(p1->py)[i]=0;
}
}
end

struct Particle *p;
int i;
p=(struct Particle*)malloc(MAX*sizeof(struct Particle));
/* Particle initialisation */
for(i=0;i<MAX;i++)
begin
(p+i)->pxd=-1+2*rand()/RAND_MAX;// random float between -1 and 1
float pxd=(p+i)->pxd; // variable pxd stores the decimal value of the x coordinate
(p+i)->pyd=-1+2*rand()/RAND_MAX;
float pyd=(p+i)->pyd; // variable pyd stores the decimal value of the y coordinate
if(i/2)// velocities
(p+i)->vx=0;
else
(p+i)->vx=-1+2*rand()/RAND_MAX;
if(i/2)
(p+i)->vy=0;
```

```
else
(p+i)->vy=-1+2*rand()/RAND_MAX;
decimal_to_binary(pxd,(p+i),'x'); // array px stores the binary value
of the x coordinate
decimal_to_binary(pyd,(p+i),'y'); // array py stores the binary value
of the y coordinate
end
int t,j;
int t_max=10;
float vx,vy,sx,sy,fx,fy,dt=0.0001;

/*Placing particles in the tree*/
struct node *root,*temproot;
root=(struct node*)malloc(sizeof(struct node));// creates main root of the tree

for(t=0;t<t_max;t++)// for each time interval
{
for(i=0;i<MAX;i++)// for each particle
{
a=0;
b=0;
c=0;
d=0;
temproot=root;

for(j=0;j<11;j++)
{

if((p+i)->px[j]==0&&(p+i)->py[j]==0)
{
temproot=Newnode(temproot,(p+i)->pxd,(p+i)->pyd,1,j);// assigns temproot to the root
}
else if((p+i)->px[j]==1&&(p+i)->py[j]==0)
{
temproot=Newnode(temproot,(p+i)->pxd,(p+i)->pyd,2,j);
}
else if((p+i)->px[j]==0&&(p+i)->py[j]==1)
{
temproot=Newnode(temproot,(p+i)->pxd,(p+i)->pyd,3,j);
}
else if((p+i)->px[j]==1&&(p+i)->py[j]==1)
{
temproot=Newnode(temproot,(p+i)->pxd,(p+i)->pyd,4,j);
}
```

```
}
}

for(i=0;i<MAX;i++)// finding the force by checking which particles are the particle
{
temproot=root;
struct node *parent= SearchParent(temproot,(p+i));
FindForce(parent,(p+i),&fx,&fy);
vx=(p+i)->vx;
vy=(p+i)->vy;
sx=vx*dt+0.5*fx*dt*dt;// displacement in x direction
sy=vy*dt+0.5*fy*dt*dt;// displacement in y direction
(p+i)->vx=vx+fx*dt;// velocity in x direction
(p+i)->vy=vy+fy*dt;// velocity in y direction
(p+i)->pxd+=sx;// new position in x direction
(p+i)->pyd+=sy;// new position in y direction

if((p+i)->pxd>1||(p+i)->pxd<-1)// checking for out of boundary condition
{
(p+i)->vx*=-1;// reversing normal velocity
if((p+i)->pxd)
(p+i)->pxd=0.99999999;// making sure the particle is inside the boundary
else
(p+i)->pxd=-0.99999999;
}
if((p+i)->pyd>1||(p+i)->pyd<-1)
{
(p+i)->vy*=-1;
if((p+i)->pyd)
(p+i)->pyd=0.99999999;
else
(p+i)->pyd=-0.99999999;
}
}
}
```

3.2 Time-complexity of the 4-way tree algorithm

- To create the tree every time interval, we need to create the array of binary representation.
- For each of x and y, we have 11 integer operations(comparisons) and 10 multiplications(by 2). Therefore, approximately 20 FPCs for binary representation of a number.

- For all particles, we have to perform these 20 operations so this step is O(20n).
- Then we have to fill the tree with these particles, for each particle we have to do 4 comparisons(worst case), for 11 bits, and then assign a particle to a child. So totally 44 operations. This step is O(44n).
- To search for the neighbours of a particle, we have to search for its parent, which is again 44 operations per particle. Then for each of these neighbours, we have to find force, so: 2 subtractions for distance, and 6 operations for forces. So totally 8*number of neighbours=800(for this case).
- To move each particle we need to compute the new velocity and the new position.
- New velocity computation involves 1 multiplication and 1 addition = 2 FPCs.
- New position computation involves 2 multiplications and 1 addition(since s_i and ut can be added parallely while computing the other term)= 3 FPCs.
- Since, this is done step by step in C, we require 5 FPCs.
- To check boundary conditions, we need to make 1 comparison, which is one FPC and the rest are integer operations and assignments which are done parallelly.
- So to calculate the force and move a particle we need 44+800+2+3+1=845. Therefore the iteration is O(44n+845n), which is O(890n).
- Since we need to do this for each time interval, the order is O(890nT), where T is the total number of time intervals.
- For this particular problem we approximately need $890*10^8*10^4$ $10^-9*10=89*10^5$
- If we reduce the number of particles to 10^6 and lesser, we can assume that we have 1 neighbour for each particle, so the order comes out to be O(100nT).

3.3 Comparison of both the algorithms

Clearly the second algorithm is better than the first

4 Sorting the velocity array

Sort algorithm used:Binsort L2 cache:1 Mb Code is included in ee17b072.c file.