Attached is the code for my project.

galaxy_v5.0_1.c - main code, has flag for nearest neighbor and n-body.
gpuv_D.c - gpu code, kernel contains main computational loop with
opencl features to pass data through memory buffers, etc
vcolor.c - data processing code to read vdat output files and make plots/movie frames
vrots.c - data processing code to read vdat files and compute rotational velocity plots.

galaxy_v5.0_1.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#define NMAX 4096
#define NNMAX 600

void main (void)
{
double year,sm,day,oneau,twoau,kpc,dt,myr,tmax,Mdisc,Mcore;
double mv,mvc,alpha,rdiskmin,alpha2,g,A;
 double m0,rc,mcdm0,PI,rv,t2,t1;
double dr,dro,math,rkpc,delvt,mass,vm;
 double x2,y2,restart,galaxy,elapsed;
double delv,rad,rn,rmin,drmin,En,En0;
double vm2,U,t,fr,ex,ey,f,rsq,vxo,vyo;

double Fintx,Finty,fint;
double exi,eyi,Fx,Fy,Axn,Ayn,dx,dy,cdm;

//double *m = malloc(10 * NMAX (double));
// double //* m;
 // m = (double*) malloc (NMAX+1);
 // if (m = NULL) printf("fail"), exit (1);
 double  m[NMAX],r[NMAX],th;
double mt[NMAX],mcdm[NMAX];
double Ax[NMAX],Ay[NMAX];
double vx[NMAX],vy[NMAX],x[NMAX],y[NMAX],vs[120],rvs[120];
int  n0,idump,ik,iv,nvs,starran,flag,idumpmax,inn;
int  i,j,k,n,ind,nn[NMAX],jn[NMAX*NNMAX];
int i1,i2,i3,i4;

static char *digit[]={"0","1","2","3","4","5","6","7","8","9"};
char fn[4];
char fname[14];
```

```
//static char version[14];

/* File pointers for data output */
 FILE *fp1,*fpE;

 /* inn=1 is Nearest Neighbor */
 /* inn=0 is N-Body */
inn=1;

// WITH CDM (cdm=1)
// cdm=1.0;
 // NO CDM!!
 cdm=1.0;

 /* galaxy = 2 is NGC 2403 */
 /* galaxy = 1 is NGC 3198 */
 /* galaxy = 0 is ANDROMEDA */
 galaxy = 2;

 /*version info */
 // version = "galaxy_v5.1.c";

/* one solar mass in kg*/
sm = 1.98892e30;
/* Length of day in seconds*/
day = 24.0*3600.0;
/* Length of a year in seconds*/
year = 365.25*day;
/* 1 million years */
myr=1.0E6*year;
/* one AU in meters*/
oneau = 149597871.0E3;
twoau = 229388960.0E3;
/* one kilo parsec*/
kpc = 2.06e8*oneau;
/*----------------------------------------*/
dt = year*40.0;
/* Maximum simulation time  (seconds)*/
tmax = myr*1000.0;
idumpmax=100000/4;
idump = 0;
/*-----------------------------*/
/*DISC AND CORE MASSES in kg!!*/
 if(galaxy == 2) {
   Mdisc=1.18e10*sm*.88;
   Mcore=.72e10*sm*0.75;
 }
 if(galaxy == 1) {
```

```
  Mdisc=.98e10*sm;
  Mcore=1.03e10*sm;
 }
 if(galaxy == 0) {
  Mdisc=7.0e10*sm;
  Mcore=2.5e10*sm;
 }
/*------------------------------*/
n=4096;
n0=3810;
/*------------------------------*/
/*Mass of each "star" in DISC AND CORE in kg!!*/
/*------------------*/
//printf("%d\n",version);
 printf("galaxy_v5.2.c\n");
 printf("==================================\n");
 printf("galaxy=%d - (0 = And - 1 = NGC3198)\n",galaxy);
 printf("==================================\n");
 printf("\n");
 printf("==========Init Variables==========\n");
 printf("n=%d\n",n);
 printf("n0=%d\n",n0);
 printf("dt=%d | %d\n",dt,dt/year);
 printf("Mdisc=%d\n",Mdisc);
 printf("Mcore=%d\n",Mcore);
 printf("tmax=%d\n",tmax);
 printf("idump=%d\n",idump);
 printf("idumpmax=%d\n",idumpmax);
/*Mass of each "star"*/
mv = (Mdisc / n0);
mvc = (Mcore / (n-n0));
/*----------------------*/



/*BLACK HOLE mass + 1% of the core*/
//m0 = 1.0e8*sm;
 printf("mv=%d\n",mv);
 printf("mvc=%d\n",mvc);
/* DM core density*/
 if(galaxy == 2) {
  mcdm0=3.2e11*sm*0.8;
 }
 if(galaxy == 1) {
  mcdm0=3.2e11*sm;
    }
 if(galaxy == 0) {
  mcdm0=7.2e11*sm;
```

```
 }
printf("m0=%d\n",m0);
printf("mcdm0=%d\n",mcdm);
/*Constant in NFW CDM distribution (units of kpc)*/
A=15.0;

/*gravity constant: N m^2/kg-2*/
/* so masses are in kg, and distances are in meters*/
g = 6.673e-11;


/* constant PI */
PI = 4.0*atan(1.0);

/* Initialize star masses in DISK and CORE */
//printf("n=%d n0=%d\n",n,n0);
// printf("setting masses..");
for (i=0;i<n0;i++){
/* set all star masses the same in kg in DISC*/
   m[i] = mv;
}

for (i=n0;i<n;i++){
  /* set all star masses the same in kg in CORE*/
    m[i] = mvc;
}


/* ========================================*/
/* INitialize star locations and velocities*/
/* ========================================*/

/* Minimum radius for DISK mass */
//rdiskmin = 2.0*kpc;
//rdiskmin = 1.5*kpc;
 rdiskmin = 3.0*kpc;

/*DISK region goes from rdiskmin out to around 50kpc */
/* This is for an exponential: exp(-alpha*r) mass distribution */
alpha = 0.125;
//alpha = 0.15;
 printf("alpha=%d\n",alpha);
 printf("==================================\n");
 printf("\n");
 printf("==========Unit Variables==========\n");
 printf("sm=%d\n",sm);
 printf("day=%d\n",day);
 printf("year=%d\n",year);
```

```
printf("au=%d\n",oneau);
printf("kpc=%d\n",kpc);
printf("==================================\n");
printf("\n");
printf("==========Flag Variables==========\n");
printf("inn=%d\n",inn);
printf("cdm=%d\n",cdm);
printf("==================================\n");
printf("\n");
   printf("Initializing mass distribution: Disk\n");
   for (i=0;i<n0;i++)
   {
    rv = ((double) (rand() %10000))/10000.1;
    //          printf("i= %d rv= %f\n",i,rv);
    r[i]=rdiskmin + kpc*(-1.0/alpha)*log(1.0-rv);
     //         printf("i= %d r= %f\n",i,r[i]);
      rv = ((double) (rand() %10000))/10000.1;
      th=rv*2.0*PI;
      x[i]=r[i]*cos(th);
      y[i]=r[i]*sin(th);
      //          printf("%f  %f \n",x[i]/kpc,y[i]/kpc);
   }

/* Minimum distance in meters between two mass points in disk region*/
   dro=0.1*kpc;
   flag = 1;
   while(flag==1)
   {
      flag=0;
      printf("Scanning stars if too close\n");

      for(i=0;i<n0;i++)
      {
          for(j=0;j<n0;j++)
          {
              if(j!=i)
                  {
                  dr=sqrt((x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]));

                  if(dr<dro)
                          {
                              rv = ((double) (rand() %10000))/10000.1;
                      r[i]=rdiskmin + kpc*(-1.0/alpha)*log(1.0-rv);
                              rv = ((double) (rand() %10000))/10000.1;
                      th=rv*2.0*PI;
                      x[i]=r[i]*cos(th);
                      y[i]=r[i]*sin(th);
                      flag=1;
```

```
                    }
                }
            }
        }
    }


/* CORE region starts at 0.25kpc and goes out to 2.0kpc */
/* This is for an exponential: exp(-alpha*r) mass distribution */
    printf("Initializing mass distribution: Core\n");
    alpha2=0.05;

    for(i=n0;i<n;i++)
    {
        rv = ((double) (rand() %10000))/10000.1;
        //          printf("i= %d rv= %f\n",i,rv);
        r[i]=0.25*kpc + kpc*(-1.0/alpha2)*log(1.0-rv);
        rv = ((double) (rand() %10000))/10000.1;
        th=rv*2.0*PI;
        x[i]=r[i]*cos(th);
        y[i]=r[i]*sin(th);
    }

    //Minimum distance between two mass points in core region
    dro=0.07*kpc;
    flag = 1;
    while(flag==1)
    {
        flag=0;
        printf("Scanning stars if too close\n");

        for(i=n0;i<n;i++)
        {
            for(j=n0;j<n;j++)
            {
                if(j!=i)
                    {
                    dr=sqrt((x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]));

                    if(dr<dro)
                            {
                                rv = ((double) (rand() %10000))/10000.1;
                        //r=.25*kpc ORIG
                        r[i]=0.05*kpc + kpc*(-1.0/alpha2)*log(1.0-rv);
                                rv = ((double) (rand() %10000))/10000.1;
                        th=rv*2.0*PI;
                        x[i]=r[i]*cos(th);
                        y[i]=r[i]*sin(th);
```

```
                    flag=1;
                        }
                }
            }
        }

    }

/*=============================================*/
/* Dump Mass distributons for plotting */
/*=============================================*/
    fp1=fopen("mass.mtv","w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",0.0);
    fprintf(fp1,"%%xmin= -100\n");
    fprintf(fp1,"%%xmax= +100\n");
    fprintf(fp1,"%%ymin= -100\n");
    fprintf(fp1,"%%ymax= +100\n");
    fprintf(fp1,"%%equalscale= T\n");
    fprintf(fp1,"%%xlabel= \"x (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"y (kpc)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linetype= 0\n");
    for (i=0;i<n;i++)

     {
      fprintf(fp1,"%f %f\n",x[i]/kpc,y[i]/kpc);
     }
    fprintf(fp1,"\n");
    fclose(fp1);


/*=============================================*/
/* Compute Effective central mass = All Mass within a radius rkpc */
/*=============================================*/
for(i=0;i<n;i++)
{
  mt[i]=0.0;
/*THIS DARK MATTER DIST is NFW*/
/*static, spherically symmetric mass*/
/* mcdm is the integrated NFW CDM density x volume = total CDM within radius rkpc */
    rkpc=r[i]/kpc;
    /* this gives an approximately linear mass increase in DISK region */
    /* it becomes flat at very large radius*/
    mcdm[i]=cdm*mcdm0*(-rkpc/(A + rkpc) - log(A) + log(A+rkpc));

/* Compute total galaxy mass within radius r[i] */
```

```
/*n0 to n is only CORE!!
/*0 to n0 is disk only
/*0 to n is BOTH core + disc*/
    for(j=0;j<n;j++)
    {
        if(r[j]<r[i])
        {
            mt[i]=m[j]+mt[i];
        }
    }
}

/*Minimum distance (rmin=1.0 kpc) for force calculation*/
/* prevents large 2 body interactions*/
rmin=1.0*kpc;
drmin=rmin*rmin;

/*=============================================*/
/* Initialize the mass velocities consistent with uniform */
/* circular motion about a central mass point */
/*=============================================*/
for(i=0;i<n;i++)
{
  ///* total mass is black hole + core + disk + CDM*/
  mass=m0 + mt[i] + mcdm[i];

/* DARK MATTER ONLY*/
  //  mass=mcdm[i];
/* BLACK HOLE + DISK AND/OR CORE*/
//  mass=m0+mt[i];
/* DISK only*/
//  mass=mt[i];

/* Equilibrium velocity for uniform circular motion
   about the total mass */
//    vm=sqrt(g*mass/r[i]);
/* include drmin in order to set better initial velocities near center */
    dr=r[i]*r[i] + drmin*0.1;
    vm=sqrt(r[i]*g*mass/dr);

/* Now Add in random radial velocity dispersion*/
/* to DISK only */
    rv = ((double) (rand() %10000))/10000.1;
    rv = rv * 2.0 - 1.0;
    if(r[i]>3.0*kpc && r[i]< 60.0*kpc)
    {
        delv=rv*40000.0*(60.0*kpc - r[i] )/(57.0*kpc);
    }
```

```
    else
    {
        delv=0.0;
    }
/* Tangential dispersion */
/* to DISK only */
    rv = ((double) (rand() %10000))/10000.1;
    rv = rv * 2.0 -1.0;
    if(r[i]>3.0*kpc && r[i]<60.0*kpc)
    {
        delvt=rv*40000.0*(60.0*kpc-r[i])/(57.0*kpc);
    }
    else
    {
        delvt=0.0;
    }

/* Total velocity + random radial + random tangential dispersion */

    vx[i]=vm*y[i]/r[i] + delv*x[i]/r[i] + delvt*y[i]/r[i];
    vy[i]=-vm*x[i]/r[i] + delv*y[i]/r[i] - delvt*x[i]/r[i];
}
/*
fname[0]=0;
strcat(&fname[0],"data/vdat_");
strcat(fname,&fn[0]);
printf("%s\n",fname);

fp1=fopen(fname,"r");
for (i=0;i<n;i++)
   {
     fscanf(fp1,"%le %le %le %le\n",&x[i],&y[i],&vx[i],&vy[i]);
     r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
      //          printf("%e %e %e %e\n",x[i],y[i],vx[i],vy[i],r[i]);
   }
 fclose(fp1);
*/
/* If nearest neighbor (inn=1), then */
/* update mass within radius r[i]. */
/* In order to speed up calculations, we */
/* don't do this every time step, only every idump */
   if (inn == 1) {
        for(i=0;i<n;i++)
        {
            rv=sqrt(x[i]*x[i] + y[i]*y[i])/kpc;
            rn = 0.5*kpc*(1.0 + rv/5.0);

             mt[i]=0.0;
```

```
            for(j=0;j<n;j++)
            {
                  x2=(x[j] - x[i])*(x[j] - x[i]);
                  y2=(y[j] - y[i])*(y[j] - y[i]);
                  dr=sqrt(x2 + y2);
// if mass j is inside r[i] and OUTSIDE nearest neighbor radius rn
// then include it in the total mass within radius r[i]
               if(r[j] < r[i] && dr > rn)
                   {
                  mt[i] = m[j] + mt[i];
                   }
            }
        }
    }


/*================================================*/
/* Compute Nearest Neighbor interactions if inn = 1*/
/*================================================*/

if (inn == 1 ) {
neighbor(n,&nn,&jn,&x,&y);
 }

/* Dump list for degubbing only
for(i=0;i<n;i++)
  {
    printf("i=%d nn=%d\n",i,nn[i]);
    for (j=0;j<nn[i];j++){
      ind=i*NNMAX + j;
      printf("j=%d jn=%d\n",j,jn[ind]);}

     }
*/

/*================================================*/
/* Compute velocity rotation curve*/
/*================================================*/


velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

     fp1=fopen("velrot.mtv","w");
     fprintf(fp1,"%%xmin= 0\n");
     fprintf(fp1,"%%xmax= +30\n");
     fprintf(fp1,"%%ymin= 0\n");
     if(galaxy == 0) {
       fprintf(fp1,"%%ymax= +300\n");
```

```
    }
    if(galaxy == 1) {
      fprintf(fp1,"%%ymax= +200\n");
    }
    if(galaxy == 2) {
      fprintf(fp1,"%%ymax= +200\n");
    }
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linecolor= 3\n");
    fprintf(fp1,"%%linetype= 1\n");

    for (i=0;i<41;i++)
      {
       fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
      }
    fprintf(fp1,"\n");
    fclose(fp1);



/* Compute total energy */
eng(&En0,&vx,&vy,n,m0,&mt,&mcdm,&r,&x,&y,drmin,&m);

//printf("Initial Energy En0= %f\n",En0);

/* Open file to store energy vs time */
    fpE=fopen("En.mtv","w");
    fprintf(fpE,"%%xmin= 0\n");
    fprintf(fpE,"%%xmax= +1000\n");
    fprintf(fpE,"%%ymin= 0\n");
    fprintf(fpE,"%%ymax= +2.0\n");
    fprintf(fpE,"%%xlabel= \"t (myr)\"\n");
    fprintf(fpE,"%%ylabel= \"E/E0\"\n");
    fprintf(fpE,"%%markertype= 13\n");
    fprintf(fpE,"%%markercolor= 0\n");
    fprintf(fpE,"%%linecolor= 0\n");
    fprintf(fpE,"%%linetype= 1\n");

/*=========================================*/
/* Start Galaxy evolution loop */
/*=========================================*/
t=0;
ik=0;
```

```
/* set file name counters to zero */
  i1=0;
  i2=0;
  i3=0;
  i4=0;

/* Initial Energy ratio = 1.0 */
fprintf(fpE,"%f %f\n",t,En0/En0);

/* Compute initial Acclerations of all mass points */

getaccel(n, inn, cdm, mcdm0, m0, A, &r, &x, &y, &Ax, &Ay, &mcdm, &mt, &m, &jn, &nn, drmin);


/*==================================*/
/* Start time evolution */
/*==================================*/

printf(">running program..\n");

while (t<tmax)
{
    t=t+dt;

/* idump determines how often to write results to disk */
    idump = idump + 1;

    for(j=0;j<n;j++)
    {
/* update the position of mass j */
        x[j] = x[j] + vx[j]*dt + 0.5*Ax[j]*dt*dt;
        y[j] = y[j] + vy[j]*dt + 0.5*Ay[j]*dt*dt;
/* form 1st half of velocity update */
        vx[j]=vx[j] + 0.5*Ax[j]*dt;
        vy[j]=vy[j] + 0.5*Ay[j]*dt;
 }


/* ------------------------------------------- */
/* Now compute the Ax and Ay at the new x and y
/* So that the Velocity can be updated (velocity Verlet algorithm)
/* The effective accleration is the average of A(t) and A(t+dt)
/* This method is second order accurate and has good stability for galaxy type
/* simulations (better stability than Runge-Kutta and predictor corrector)
/* ------------------------------------------- */

getaccel(n, inn, cdm, mcdm0, m0, A, &r, &x, &y, &Ax, &Ay, &mcdm, &mt, &m, &jn, &nn, drmin);
```

```
    for(j=0;j<n;j++)
    {
       vx[j]=vx[j] + 0.5*Ax[j]*dt;
       vy[j]=vy[j] + 0.5*Ay[j]*dt;

    }


/* Dump data it idumpmax count is reached */
    if (idump == idumpmax)
    {
     idump=0;
 time_t timeraw;
      struct tm * timeinfo;

      time (&timeraw);
      timeinfo = localtime (&timeraw);
      printf("Time is: %s \n", asctime(timeinfo));

      t2=clock();
      elapsed = ((double) (t2-t1))/CLOCKS_PER_SEC;

        printf("t (myr)= %f %%= %f\n",t/myr,100.0*t/tmax);
        printf("Time= %f (sec) | %f (min) || Localtime= %s\n",elapsed,elapsed/60,asctime(timeinfo));

        printf("t (myr)= %f %%= %f\n",t/myr,100.0*t/tmax);
         printf("Updating Mass...\n");
/* file name counter update */
     ik=ik+1;

     i1++;
     if (i1 % 10 == 0) {
       i1=0;
      i2++;
     if (i2 % 10 == 0) {
       i2=0;
      i3++;
     if (i3 % 10 == 0) {
       i3=0;
      i4++;
     }}}

//    printf ("File # %d = %d %d %d %d\n",ik,i4,i3,i2,i1);
/* make string file name out of time step counters */
/* fn = 0000,00001,00002, etc */
   fn[0]=0;
   strcat(&fn[0],digit[i4]);
   strcat(&fn[0],digit[i3]);
```

```
   strcat(&fn[0],digit[i2]);
   strcat(&fn[0],digit[i1]);
   printf ("Fstring= %s\n",fn);

/* If nearest neighbor (inn=1), then */
/* update mass within radius r[i]. */
/* In order to speed up calculations, we */
/* don't do this every time step, only every idump */
   if (inn == 1) {
        for(i=0;i<n;i++)
       {
           rv=sqrt(x[i]*x[i] + y[i]*y[i])/kpc;
           rn = 0.5*kpc*(1.0 + rv/5.0);

            mt[i]=0.0;
            for(j=0;j<n;j++)
           {
                 x2=(x[j] - x[i])*(x[j] - x[i]);
                 y2=(y[j] - y[i])*(y[j] - y[i]);
                 dr=sqrt(x2 + y2);
// if mass j is inside r[i] and OUTSIDE nearest neighbor radius rn
// then include it/ in the total mass within radius r[i]
               if(r[j] < r[i] && dr > rn)
                   {
                   mt[i] = m[j] + mt[i];
                    }
           }
       }
   }

/*=============================================*/
/* UPdate Nearest Neighbors every idump */
/*=============================================*/
if (inn == 1 ) {
neighbor(n,&nn,&jn,&x,&y);
 }
/*=========================================*/
/* compute and save total energy*/
/* Total energy conservation      */

eng(&En,&vx,&vy,n,m0,&mt,&mcdm,&r,&x,&y,drmin,&m);

printf("Energy En/En0= %f\n",En/En0);
fprintf(fpE,"%f %f\n",t/myr,En/En0);
fflush(fpE);

/*=============================================*/
/*Compute rotational velocity at this time step*/
```

```
velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

    fname[0]=0;
    strcat(&fname[0],"data/vrot_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",t/myr);
    fprintf(fp1,"%%xmin= 0\n");
    fprintf(fp1,"%%xmax= +30\n");
    fprintf(fp1,"%%ymin= 0\n");
    fprintf(fp1,"%%ymax= +300\n");
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%linetype= 1\n");
    for (i=0;i<41;i++)
     {
      fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
     }
    fprintf(fp1,"\n");
    fclose(fp1);


/*=======================================*/
/* Dump data to disk here */
/*=======================================*/

    fname[0]=0;
    strcat(&fname[0],"data/mass_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",t/myr);
    fprintf(fp1,"%%xmin= -100\n");
    fprintf(fp1,"%%xmax= +100\n");
    fprintf(fp1,"%%ymin= -100\n");
    fprintf(fp1,"%%ymax= +100\n");
    fprintf(fp1,"%%equalscale= T\n");
    fprintf(fp1,"%%xlabel= \"x (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"y (kpc)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linetype= 0\n");
```

```
    for (i=0;i<n;i++)
     {
      fprintf(fp1,"%f %f\n",x[i]/kpc,y[i]/kpc);
     }
    fprintf(fp1,"\n");
    fclose(fp1);

/* Dump velocities  to disk */
    fname[0]=0;
    strcat(&fname[0],"data/vdat_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    for (i=0;i<n;i++)
     {
      fprintf(fp1,"%e %e %e %e\n",x[i]/kpc,y[i]/kpc,vx[i],vy[i]);
     }
    fprintf(fp1,"\n");
    fclose(fp1);


/*=======================================*/
    }
}


/* Compute velocity rotation curve AT END OF SIMULATION*/
/* Average all the star velocities with +0.5 and -0.5 kpc*/
/* of radius = rad*/
velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

    fp1=fopen("velrotF.mtv","w");
    fprintf(fp1,"%%xmin= 0\n");
    fprintf(fp1,"%%xmax= +30\n");
    fprintf(fp1,"%%ymin= 0\n");
    fprintf(fp1,"%%ymax= +300\n");
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linetype= 1\n");
    fprintf(fp1,"%%linecolor= 3\n");
    for (i=0;i<41;i++)
     {
      fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
     }
    fprintf(fp1,"\n");
```

```
    fclose(fp1);

/*=========================================*/
/*END OF MAIN PROGRAM*/
/*=========================================*/

}


/* ========================================================= *
/* SUBROUTINES BELOW */
/* ========================================================= */

/* Computes Total Energy returned in En*/
eng(double *En,double *vx,double *vy,int n,double m0, double *mt, double *mcdm,double *r,
double *x,double *y,double drmin,double *m)
{
  int i,j;
  double vm2,vx2,vy2,x2,y2;
  double mass,g,U,dr;

/*gravity constant: N m^2/kg-2*/
g = 6.673e-11;

// printf("m[0]= %f\n",*(m+0));

*En=0.0;
for(j=0;j<n;j++)
{
  vx2=(*(vx+j))*(*(vx+j));
  vy2=(*(vy+j))*(*(vy+j));
  vm2=vx2+vy2;

/*CENTRAL mass = BLACK hole + CDM*/
  mass=m0 + *(mcdm+j);

/* Potential energy due to large central mass */
/* use drmin to limit force/potential when r -> 0 */
/* central force drmin is 1/4 drmin to better model core */
  U=-g*(*(m+j))*mass/sqrt((*(r+j))*(*(r+j)) + drmin*0.1);

  //  printf("vm2= %f, mass= %f, U= %f\n",vm2,mass,U);

/* Add up potential energies due to all other mass points */
    for(i=0;i<n;i++)
    {
        if (i != j)
       {
```

```
        x2=(*(x+j) - *(x+i))*(*(x+j) - *(x+i));
        y2=(*(y+j) - *(y+i))*(*(y+j) - *(y+i));
        dr=x2 + y2 + drmin;
         U=U-g*(*(m+j))*(*(m+i))/sqrt(dr);
        //          printf("i=%d U= %f\n",i,U);
       }
    }
    *En=*En + 0.5*(*(m+j))*vm2 + U;
 }
}


/*================================================*/
/* Compute Nearest Neighbor lists */
/* nn(i) is the number of nearest neighbors about ith mass point
/* jn stores the index for each neighbor about ith mass point
/*================================================*/

neighbor(int n,int *nn,int *jn,double *x, double *y)
{

#define NNMAX 600

 int i,j,ind,nnx;
 double rv,kpc,rn,oneau,x2,y2,dr;

/* one kilo parsec*/
oneau = 149597871.0E3;
kpc = 2.06e8*oneau;

nnx=0;

for(i=0;i<n;i++)
{
/*make NN range increase with radius*/
  rv=sqrt((*(x+i))*(*(x+i)) + (*(y+i))*(*(y+i)))/kpc;
  rn = 0.5*kpc*(1.0 + rv/5.0);

  *(nn+i)=0;

    for(j=0;j<n;j++)
    {
        if(j != i)
       {
         x2=(*(x+j) - *(x+i))*(*(x+j) - *(x+i));
         y2=(*(y+j) - *(y+i))*(*(y+j) - *(y+i));
         dr=sqrt(x2 + y2);
            if(dr < rn)
```

```
          {
               ind=i*NNMAX + *(nn+i);
               *(jn+ind)=j;
            (*(nn+i))++;
               if (*(nn+i) > NNMAX) {
                 printf("ERROR: nn > NNMAX\n");
              printf("Terminating..\n");
              break;
             }

          }
        }
    }
    if (*(nn+i) > nnx) {
      nnx=*(nn+i);
        }
}
 printf("Max nn = %d\n",nnx);
}



/*================================================*/
/* Compute velocity rotation curve*/
/* Average all the star tangential velocities within +0.5 and -0.5 kpc */
/* of radius rad=1-30, returned in vs(0-29) */
/*================================================*/

velrot(int n,double *r,double *vx,double *vy,double *vs,double *rvs,double *x,double *y)
{
int j,i,nvs,jp;
double rad,oneau,kpc,dnvs,v2,v2x,v2y;

/* one kilo parsec*/
oneau = 149597871.0E3;
kpc = 2.06e8*oneau;

/* use a finer grid from 1 - 5 kpc */
for(j=0;j<16;j++)
{
/* Add one to put on 1 - 5 kpc range */
  rad = ((double) j + 1.0)*0.25;
  *(rvs+j)=rad;
/* nvs counts the number of stars */
    nvs=0;
/* initialize velocity to zero */
    *(vs+j)=0;
/* loop over all stars and test if within +/- 0.5kpc of rad */
```

```
        for(i=0;i<n;i++)
        {
    /* is star j within +/- 0.125 kpc of rad?*/
         if ((*(r+i) / kpc > rad - 0.125) && (*(r+i) / kpc < rad + 0.125)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+j)=*(vs+j)+v2;
    /* count the number of stars in this range*/
          nvs=nvs+1;
           }
        }

        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
    /* divide by 1000 to get km/sec*/
            dnvs = (double) nvs;
            *(vs+j)=*(vs+j)/(1000.0*dnvs);
        }
        else {
    /* set to previous value if no particles were inside this radius range */
            *(vs+j)=*(vs+j-1);
         }
    }

    /* use a coarser grid from 5-30kpc */
    for(j=0;j<26;j++)
    {
     jp=j+16;
     rad = ((double) j)*1.0 + 5.0;
     *(rvs+jp)=rad;
    /* nvs counts the number of stars */
        nvs=0;
    /* initialize velocity to zero */
        *(vs+jp)=0;
    /* loop over all stars and test if within +/- 0.5kpc of rad */
        for(i=0;i<n;i++)
        {
    /* is star j within +/- 0.125 kpc of rad?*/
         if ((*(r+i) / kpc > rad - 0.5) && (*(r+i) / kpc < rad + 0.5)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
```

```
    /* Add up the tangential velocities */
       *(vs+jp)=*(vs+jp)+v2;
    /* count the number of stars in this range*/
         nvs=nvs+1;
          }
       }

       if(nvs > 0)
       {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
    /* divide by 1000 to get km/sec*/
           dnvs = (double) nvs;
           *(vs+jp)=*(vs+jp)/(1000.0*dnvs);
       }
       else {
    /* Set to previous value if no stars are in this range of rad */
        *(vs+jp)=*(vs+jp-1);
       }
     }


    }


    /* ======================================== */
    /* Compute Force due to gravity and accleration */
    /* ======================================== */

    getaccel(int n, int inn, double cdm, double mcdm0, double m0,double A, double *r, double *x,
double *y, double *Ax, double *Ay, double *mcdm, double *mt, double *m, int *jn, int*nn, double
drmin)
    {

    #define NNMAX 600

        int i,j,iv,ind;
        double kpc,rkpc,rsq,oneau,mass,g,f;
        double ex,ey,exi,eyi,dr,Fintx,Finty,fint;
        double Fx,Fy;



    /*gravity constant: N m^2/kg-2*/
    g = 6.673e-11;
    /* one kilo parsec*/
    oneau = 149597871.0E3;
    kpc = 2.06e8*oneau;
```

```
        for(j=0;j<n;j++)
        {
            rsq = x[j]*x[j] + y[j]*y[j];
            r[j]=sqrt(rsq);
            rkpc=r[j]/kpc;

    /*THIS DARK MATTER DIST is NFW*/
    /*static, spherically symmetric mass*/
    /* mcdm is the integrated NFW CDM density x volume = total CDM within radius rkpc */
    /* this gives an approximately linear mass increase in DISK region */
    /* it becomes flat at very large radius to give a finite mass*/
            mcdm[j]=cdm*mcdm0*(-rkpc/(A + rkpc) - log(A) + log(A+rkpc));

    /*Total mass inside radius rkpc= Black hole + core + disK + CDM*/
    /* if nearest neighbor (inn=1), then central mass includes mt */
    /* if N-body (inn=0), then don't include it since it is in the full interaction term */
           if (inn == 1) {
            mass=m0 + mt[j] + mcdm[j]; }
           else {
            mass=m0 + mcdm[j]; }

    /* force due to gravity between mass point m[j] and central total mass, use drmin to cut off force if
rsq ->0 */
    /* central force drmin is 1/4 drmin to better model core */
           f = (g*m[j]*mass)/(rsq+drmin*0.1);
    /* cosine and sine */
           ex = x[j] / r[j];
           ey = y[j] / r[j];

    /* Compute NEAREST NEIGHBOR force INTERACTIONS if inn == 1 else do N-Body (all interactions)*/
     if (inn == 1 ) {
            Fintx=0.0;
            Finty=0.0;
            if (nn[j] != 0)
            {
                for(i=0;i<nn[j];i++)
                {
                    ind=j*NNMAX + i;
                    iv=jn[ind];
                    dr=(x[j] - x[iv])*(x[j] - x[iv]) + (y[j]-y[iv])*(y[j]-y[iv]) + drmin;
                    fint=g*m[j]*m[iv]/dr;
                    dr=sqrt(dr);
                    exi = (x[j]-x[iv]) / dr;
                    eyi = (y[j]-y[iv]) / dr;
                    Fintx=-fint*exi + Fintx;
                    Finty=-fint*eyi + Finty;
                }
```

```
      }
 }
 else {
/* Add up ALL interactions N-Body */
       Fintx=0.0;
       Finty=0.0;
          for(i=0;i<n;i++)
          {
                if (i != j) {
              dr=(x[j] - x[i])*(x[j] - x[i]) + (y[j]-y[i])*(y[j]-y[i]) + drmin;
              fint=g*m[j]*m[i]/dr;
              dr=sqrt(dr);
              exi = (x[j]-x[i]) / dr;
              eyi = (y[j]-y[i]) / dr;
              Fintx=-fint*exi + Fintx;
              Finty=-fint*eyi + Finty;
                }
          }

 }

/* Total force due to Gravity of central mass + NN interactions*/
       Fx = -f * ex   + Fintx;
       Fy = -f * ey   + Finty;
/* Total acceleration of mass m[j] */
       Ax[j] = Fx / m[j];
       Ay[j] = Fy / m[j];
   }
 }
```

gpuv_D.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <time.h>

#define NMAX 2048
#define NNMAX 600

#pragma OPENCL EXTENSION cl_amd_fp64:enable

//GLOBALS
cl_uint length= 2048;
  cl_uint local= 64;
  cl_double * x = NULL;
  cl_double * y = NULL;
  cl_double * m = NULL;
  cl_double * xnw = NULL;
  cl_double * ynw = NULL;
  cl_double * Ax = NULL;
  cl_double * Ay = NULL;
  cl_double * vx = NULL;
  cl_double * vy = NULL;
  cl_double * mt = NULL;

//GLOBALS //
/*Minimum distance (rmin=1.0 kpc) for force calculation*/
/* prevents large 2 body interactions*/
double drmin=1.0;
double drmin2=0.1;
double m0;
double mcdm0;

// WITH CDM (cdm=1)
double cdm=1.0;
 // NO CDM!!
//double cdm=0.0;
/*Constant in NFW CDM distribution (units of kpc)*/
double A=15.0;

// Kernel
const char *source =
"#pragma OPENCL EXTENSION cl_amd_fp64:enable\n"
"__kernel void nbody( __global double *x,\n"
"                     __global double *y,\n"
```

```
"                   __global double *xnw,\n"
"                   __global double *ynw,\n"
"                   __global double *vx,\n"
"                   __global double *vy,\n"
"                   __global double *Ax,\n"
"                   __global double *Ay,\n"
"                   __global double *mt,\n"
"                   __global double *m,\n"
"                     __local double *gxb,\n"
"                     __local double *gyb,\n"
"                     __local double *gmb)\n"

"{\n"
" double dt,drmin,drmin2,g;\n"
" double x0,y0,vx0,vy0,xt,yt,mass,f,ex,ey;\n"
" double Ax0,Ay0,rsq,r,dr,fint,exi,eyi;\n"
" uint j,i,n,nt,nb,jb,k;       \n"
" drmin=1.0;\n"
" drmin2=0.1;\n"
" g = 4.517017e-12;\n"
" dt=1.0E-5;\n"
" i = get_local_id(0); \n"
" j = get_global_id(0);       \n"
" n = get_global_size(0); \n"
" nt = get_local_size(0); \n"
" nb=n/nt;\n"
" mass=mt[j];\n"
" x0=x[j];\n"
" y0=y[j];\n"
" vx0=vx[j];\n"
" vy0=vy[j];\n"
" Ax0=Ax[j];\n"
" Ay0=Ay[j];\n"
" vx0+= 0.5*Ax0*dt; \n"
" vy0+= 0.5*Ay0*dt; \n"
" rsq=x0*x0 + y0*y0 + drmin2;\n"
" r=sqrt(rsq);\n"
" f = (g*mass)/rsq;                          \n"
" ex = x0 / r;                                   \n"
" ey = y0 / r;                                   \n"
" Ax0 = -f * ex;                                   \n"
" Ay0 = -f * ey;                                   \n"

" for(jb=0;jb<nb;jb++) { \n"
"   gxb[i] = x[i+jb*nt]; \n"
"   gyb[i] = y[i+jb*nt]; \n"
"   gmb[i] = m[i+jb*nt]; \n"
"   barrier(CLK_LOCAL_MEM_FENCE); \n"
```

```
"   for (k=0;k<nt;k++) {                                        \n"
"      dr=(x0 - gxb[k])*(x0 - gxb[k]) + (y0-gyb[k])*(y0-gyb[k]) + drmin;   \n"
"      fint=g*gmb[k]/dr;                                        \n"
"      dr=sqrt(dr);                                            \n"
"      exi = (x0-gxb[k]) / dr;                                  \n"
"      eyi = (y0-gyb[k]) / dr;                                  \n"
"      Ax0+=-fint*exi;                                  \n"
"      Ay0+=-fint*eyi;                                  \n"
"      }\n"
"   barrier(CLK_LOCAL_MEM_FENCE); \n"
"}\n"

" vx0+= 0.5*Ax0*dt; \n"
" vy0+= 0.5*Ay0*dt; \n"
" xt = x0 + vx0*dt + 0.5*Ax0*dt*dt; \n"
" yt = y0 + vy0*dt + 0.5*Ay0*dt*dt; \n"
" xnw[j]= xt;\n"
" ynw[j]= yt;\n"
" vx[j] = vx0;                                          \n"
" vy[j] = vy0;                                           \n"
" Ax[j] = Ax0;                                           \n"
" Ay[j] = Ay0;                                           \n"
  "}\n";

// Print vector routine
void printVector(const cl_double *arrayData,
                 const unsigned int length)
{
  int i;
  for (i = 0;i<length;++i)
    printf("i= %d %f\n",i,arrayData[i]);
}

void initHost()
{
  int i;
  size_t sizeInBytes = length * sizeof(cl_double);
  x=(cl_double *) malloc(sizeInBytes);
  if (x == NULL)
    printf("ERROR: Failed to allocate input memory x on host\n");

  y=(cl_double *) malloc(sizeInBytes);
  if (y == NULL)
    printf("ERROR: Failed to allocate input memory y on host\n");

  m=(cl_double *) malloc(sizeInBytes);
  if (m == NULL)
```

```
   printf("ERROR: Failed to allocate input memory m on host\n");

  xnw=(cl_double *) malloc(sizeInBytes);
  if (xnw == NULL)
   printf("ERROR: Failed to allocate input memory xnw on host\n");

  ynw=(cl_double *) malloc(sizeInBytes);
  if (ynw == NULL)
   printf("ERROR: Failed to allocate input memory ynw on host\n");

  Ax=(cl_double *) malloc(sizeInBytes);
  if (Ax == NULL)
   printf("ERROR: Failed to allocate input memory Ax on host\n");

  Ay=(cl_double *) malloc(sizeInBytes);
  if (Ay == NULL)
   printf("ERROR: Failed to allocate input memory Ay on host\n");

  vx=(cl_double *) malloc(sizeInBytes);
  if (vx == NULL)
   printf("ERROR: Failed to allocate input memory vx on host\n");

  vy=(cl_double *) malloc(sizeInBytes);
  if (vy == NULL)
   printf("ERROR: Failed to allocate input memory vy on host\n");

  mt=(cl_double *) malloc(sizeInBytes);
  if (mt == NULL)
   printf("ERROR: Failed to allocate input memory mt on host\n");


  //    printf ("X\n");
    //    printVector(X,length);
  //    printf ("Y\n");
  //    printVector(Y,length);
}


/*    ******************  MAIN *********************************/

void main (void)
{
  double year,sm,day,dt,myr,oneau,tmax,Mdisc,Mcore;
double mv,mvc,alpha,rdiskmin,alpha2,g,rcoremin;
 double rc,PI,vx2,vy2,v2x,v2y,v2,twoau,kpc;
 double rv,rv2;
 double dr,dro,math,rkpc,delvt,mass,vm,dro2;
 double x2,y2,restart;
```

```
 double delv,rad,rn,En[1],En0;
 double vm2,U,t,fr,ex,ey,f,rsq,vxo,vyo,rs;

 int galaxy,dnvs,jp;
double Fintx,Finty,fint;
double exi,eyi,Fx,Fy,Axn,Ayn,dx,dy;

clock_t t1,t2;
double elapsed;


 double * r = NULL;
 double * mcdm = NULL;


 /* Allocate memory for arrays r and mcdm */
 size_t sizeInBytes = NMAX * sizeof(double);
r=(double *) malloc(sizeInBytes);
mcdm=(double *) malloc(sizeInBytes);


 if (r == NULL) {
  printf("ERROR: Failed to allocate input memory r on host\n");
 }

 if (mcdm == NULL) {
  printf("ERROR: Failed to allocate input memory mcdm on host\n");
 }


double th;
double vs[120],rvs[120];
int  n0,idump,ik,iv,nvs,starran,flag,idumpmax,inn;
int  i,j,k,n,ind,ii;
int i1,i2,i3,i4;

static char *digit[]={"0","1","2","3","4","5","6","7","8","9"};
char fn[4];
char fname[14];
//static char version[14];

/* File pointers for data output */
 FILE *fp1,*fpE;

cl_int ierr;
cl_uint nplat;
```

```
/* Initialize memory */
 initHost();
/* ==================================*/


/* OPENCL STUFF STARTS HERE */
 cl_platform_id platform;
 ierr=clGetPlatformIDs(1, &platform, &nplat);
 printf ("nplatform = %d\n",nplat);

 if (ierr != 0 ) {
  printf ("Error: check environment settings\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}

 cl_device_id device;
 ierr=clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
 // ierr=clGetDeviceIDs(platform, CL_DEVICE_TYPE_CPU, 1, &device, NULL);

 if (ierr != 0 ) {
  printf ("ERROR: in GetDeviceIDs\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}

 cl_context context;
 context = clCreateContext(NULL,1,&device,NULL,NULL,&ierr);

 if (ierr != 0 ) {
  printf ("ERROR: in CreateContext\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}

 cl_command_queue queue = clCreateCommandQueue(context,device,0,&ierr);

 if (ierr != 0 ) {
  printf ("ERROR: in CreateCommandQueue\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}

 cl_program program = clCreateProgramWithSource(context,1,&source,NULL,&ierr);

 if (ierr != 0 ) {
  printf ("ERROR: in CreateProgramWithSource\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}


 ierr=clBuildProgram(program, 1, &device, NULL, NULL, NULL);
```

```
 if (ierr != 0 ) {
  printf ("ERROR: in BuildProgram\n");
  printf ("ierr = %d\n",ierr);
  char buffer[4096];
  size_t length;
  clGetProgramBuildInfo(
                        program,
                device,
                        CL_PROGRAM_BUILD_LOG,
                        sizeof(buffer),
                        buffer,
                        &length);
  printf("%s\n",buffer);
  exit(1);
 }
 // exit(1);}


 cl_kernel kernel = clCreateKernel(program,"nbody",&ierr);

 if (ierr != 0 ) {
  printf ("ERROR: in CreateKernel\n");
  printf ("ierr = %d\n",ierr);
  exit(1);}
 /* BUFFERS CREATED BELOW AFTER INITIALIZE x,y and m */

 /* ================================================*/
 /* Galaxy Code starts HERE */

 /* inn=1 is Nearest Neighbor */
 /* inn=0 is N-Body */
inn=0;



 /* galaxy = 2 is NGC 2403 */
 /* galaxy = 1 is NGC 3198 */
 /* galaxy = 0 is ANDROMEDA */
 galaxy = 0;

 /*version info */
 // version = "galaxy_v5.1.c";

/* one solar mass in kg*/
sm = 1.98892e30;
/* Length of day in seconds*/
day = 24.0*3600.0;
```

```
/* Length of a year in seconds*/
year = 365.25*day;
/* 1 million years */
myr=1.0E6*year;
/* one AU in meters*/
oneau = 149597871.0E3;
twoau = 229388960.0E3;
/* one kilo parsec*/
kpc = 2.06e8*oneau;
/*----------------------------------------*/
//dt = year*10.0;
//Times are now in myr
 dt=1.0E-5;
/* Maximum simulation time  in myr */
tmax = 1000.0;
/* Kernel call does 2 time steps per idump */
/* so 50000 is 100000 time steps */
idumpmax=50000;
// idumpmax=100;
idump = 0;
/*------------------------------*/
/*DISC AND CORE MASSES in kg!!*/
 if(galaxy == 2) {
   Mdisc=1.18e10*.88;
   Mcore=.72e10*0.75;
 }
 if(galaxy == 1) {
   Mdisc=.98e10;
   Mcore=1.03e10;
 }
 if(galaxy == 0) {
   Mdisc=7.0e10;
   Mcore=2.5e10;
 }
/*------------------------------*/
//n=128;
//n0=100;
//n=1024;
//n0=800;
//n=4352;
//n0=4000;
n=2048;
n0=1800;
/*------------------------------*/
/*Mass of each "star" in DISC AND CORE in kg!!*/
/*-------------------*/
//printf("%d\n",version);
 printf("galaxy_v5.2.c\n");
```

```
 printf("==================================\n");
 printf("galaxy=%d  (0 = Andromeda, 1 = NGC3198, 2 = NGC 2403)\n",galaxy);
 printf("==================================\n");
 printf("\n");
 printf("==========Init Variables==========\n");
 printf("n=%d\n",n);
 printf("n0=%d\n",n0);
 printf("dt=%e myr  |  %f yrs\n",dt,dt*myr/year);
 printf("Mdisc=%e solar masses\n",Mdisc);
 printf("Mcore=%e solar masses\n",Mcore);
 printf("tmax=%e myr\n",tmax);
 printf("idump=%d\n",idump);
 printf("idumpmax=%d\n",idumpmax);
/*Mass of each "star"*/
mv = (Mdisc / n0);
mvc = (Mcore / (n-n0));
/*----------------------*/

/*BLACK HOLE mass + 1% of the core*/

 printf("mv=%e solar masses\n",mv);
 printf("mvc=%e solar masses\n",mvc);

/* DM core density*/
 if(galaxy == 2) {
   mcdm0=3.2e11*0.8;
   m0 = 0.0;
 }
 if(galaxy == 1) {
   mcdm0=3.2e11;
   m0 = 0.0;
     }
 if(galaxy == 0) {
   mcdm0=7.2e11;
   m0 = 1.0e8;
 }
printf("m0=%e solar masses\n",m0);
//printf("mcdm0=%e solar masses\n",mcdm);
printf("A=%e kpc\n",A);

/*gravity constant: N m^2 kg-2*/
/* so masses are in kg, and distances are in meters*/
// g = 6.67428e-11;

// Converted to sm,kpc,myr  for single precision code
 g = 4.517017e-12;
 printf("g = %le Sm,kpc,myr units\n",g);
```

```
/* constant PI */
PI = 4.0*atan(1.0);

/* Initialize star masses in DISK and CORE */
//printf("n=%d n0=%d\n",n,n0);
// printf("setting masses..");
for (i=0;i<n0;i++){
/* set all star masses the same in kg in DISC*/
   m[i] = mv;
}

for (i=n0;i<n;i++){
  /* set all star masses the same in kg in CORE*/
    m[i] = mvc;
}


//for (i=0;i<n;i++){
//  printf("i= %d m= %e \n",i,m[i]);}


/* ========================================*/
/* INitialize star locations and velocities*/
/* ========================================*/

/* Minimum radius in kpc for DISK mass */
//rdiskmin = 2.0;
rdiskmin = 1.5;

/* Minimum radius in kpc for CORE mass */
 rcoremin = 0.25;

/*DISK region goes from rdiskmin out to around 50kpc */
/* This is for an exponential: exp(-alpha*r) mass distribution */
//DISK region
alpha = 0.15;
//Minimum distance in disk region
dro=0.25;
// CORE region
alpha2=2.0;
//Minimum distance in core region
dro2=0.025;

 printf("alpha=%f  alpha2= %f\n",alpha,alpha2);
 printf("dro=%f  dro2= %f\n",dro,dro2);
 printf("rdiskmin=%f  rcoremin= %f\n",rdiskmin,rcoremin);
 printf("================================\n");
 printf("\n");
```

```
printf("==========Unit Variables==========\n");
printf("sm=%e\n",sm);
printf("day=%e\n",day);
printf("year=%e\n",year);
printf("au=%e\n",oneau);
printf("kpc=%e\n",kpc);
printf("==================================\n");
printf("\n");
printf("==========Flag Variables==========\n");
printf("inn=%d\n",inn);
printf("cdm=%f\n",cdm);
printf("==================================\n");
printf("\n");
   printf("Initializing mass distribution: Disk\n");
   for (i=0;i<n0;i++)
   {
    rv = ((double) (rand() %10000))/10000.1;
    //          printf("i= %d rv= %f\n",i,rv);
    rv2=rdiskmin + (-1.0/alpha)*log(1.0-rv);
     //         printf("i= %d r= %f\n",i,r[i]);
      rv = ((double) (rand() %10000))/10000.1;
      th=rv*2.0*PI;
      x[i]=rv2*cos(th);
      y[i]=rv2*sin(th);
     r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
     //         printf("%f  %f \n",x[i],y[i]);
   }

/* Minimum distance in meters between two mass points in disk region*/

   flag = 1;
   while(flag==1)
   {
      flag=0;
      printf("Scanning stars if too close\n");

      for(i=0;i<n0;i++)
      {
          for(j=0;j<n0;j++)
          {
              if(j!=i)
                  {
                  dr=sqrt((x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]));

                  if(dr<dro)
                          {
                              rv = ((double) (rand() %10000))/10000.1;
                     rv2=rdiskmin + (-1.0/alpha)*log(1.0-rv);
```

```
                    rv = ((double) (rand() %10000))/10000.1;
                th=rv*2.0*PI;
                x[i]=rv2*cos(th);
                y[i]=rv2*sin(th);
                    r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
                flag=1;
                    }
            }
        }
    }
}


/* CORE region starts at 0.25kpc and goes out to 2.0kpc */
/* This is for an exponential: exp(-alpha*r) mass distribution */
    printf("Initializing mass distribution: Core\n");

    for(i=n0;i<n;i++)
    {
        rv = ((double) (rand() %10000))/10000.1;
       //       printf("i= %d rv= %f\n",i,rv);
        rv2=rcoremin + (-1.0/alpha2)*log(1.0-rv);
        rv = ((double) (rand() %10000))/10000.1;
        th=rv*2.0*PI;
        x[i]=rv2*cos(th);
        y[i]=rv2*sin(th);
        r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
    }

    //Minimum distance between two mass points in core region

    flag = 1;
    while(flag==1)
    {
        flag=0;
        printf("Scanning stars if too close\n");

        for(i=n0;i<n;i++)
       {
            for(j=n0;j<n;j++)
           {
                if(j!=i)
                    {
                    dr=sqrt((x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]));

                    if(dr<dro2)
                            {
                                rv = ((double) (rand() %10000))/10000.1;
```

```
                    rv2=rcoremin + (-1.0/alpha2)*log(1.0-rv);
                            rv = ((double) (rand() %10000))/10000.1;
                    th=rv*2.0*PI;
                    x[i]=rv2*cos(th);
                    y[i]=rv2*sin(th);
                            r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
                    flag=1;
                         }
                  }
          }
       }

   }

/*=============================================*/
/* Dump Mass distributons for plotting */
/*=============================================*/
    fp1=fopen("mass.mtv","w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",0.0);
    fprintf(fp1,"%%xmin= -100\n");
    fprintf(fp1,"%%xmax= +100\n");
    fprintf(fp1,"%%ymin= -100\n");
    fprintf(fp1,"%%ymax= +100\n");
    fprintf(fp1,"%%equalscale= T\n");
    fprintf(fp1,"%%xlabel= \"x (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"y (kpc)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linetype= 0\n");
    for (i=0;i<n;i++)

     {
      fprintf(fp1,"%f %f\n",x[i],y[i]);
     }
    fprintf(fp1,"\n");
    fclose(fp1);


/*=============================================*/
/* Compute Effective central mass = All Mass within a radius rkpc */
/*=============================================*/
for(i=0;i<n;i++)
{
  mt[i]=0.0;
/*THIS DARK MATTER DIST is NFW*/
/*static, spherically symmetric mass*/
/* mcdm is the integrated NFW CDM density x volume = total CDM within radius rkpc */
```

```
    rkpc=r[i];
    /* this gives an approximately linear mass increase in DISK region */
    /* it becomes flat at very large radius*/
    mcdm[i]=cdm*mcdm0*(-rkpc/(A + rkpc) - log(A) + log(A+rkpc));
    //     printf("i= %d mcdm = %e solar masses\n",i,mcdm[i]);

/* Compute total galaxy mass within radius r[i] */
/*n0 to n is only CORE!!
/*0 to n0 is disk only
/*0 to n is BOTH core + disc*/
    for(j=0;j<n;j++)
    {
        if(r[j]<r[i])
       {
           mt[i]=m[j]+mt[i];
       }
    }
//Combine masses to get total
mt[i]=m0 + mt[i] + mcdm[i];

// FOR TESTING ONLY (ignore other masses)
//mt[i]=m0 +  mcdm[i];
}


/*=============================================*/
/* Initialize the mass velocities consistent with uniform */
/* circular motion about a central mass point */
/*=============================================*/
for(i=0;i<n;i++)
{

//Total mass is now mt
  mass=mt[i];

  ///* total mass is black hole + core + disk + CDM*/
  //    mass=m0 + mt[i] + mcdm[i];

/* DARK MATTER ONLY*/
  //  mass=mcdm[i];
/* BLACK HOLE + DISK AND/OR CORE*/
   //mass=m0+mt[i];
/* DISK only*/
//  mass=mt[i];

/* Equilibrium velocity for uniform circular motion
   about the total mass */
//     vm=sqrt(g*mass/r[i]);
```

```
/* include drmin in order to set better initial velocities near center */
    dr=r[i]*r[i] + drmin2;
    vm=sqrt(r[i]*g*mass/dr);
    //     printf("i= %d vm= %e \n",i,vm);

/* Now Add in random radial velocity dispersion*/
/* to DISK only */
    rv = ((double) (rand() %10000))/10000.1;
    rv = rv * 2.0 - 1.0;
    if(r[i]>3.0 && r[i]< 60.0)
    {
      //         delv=rv*40000.0*(60.0 - r[i] )/(57.0);
      delv=rv*40000.0*(myr/kpc)*(60.0 - r[i] )/(57.0);
    }
    else
    {
        delv=0.0;
    }
/* Tangential dispersion */
/* to DISK only */
    rv = ((double) (rand() %10000))/10000.1;
    rv = rv * 2.0 -1.0;
    if(r[i]>3.0 && r[i]<60.0)
    {
      //         delvt=rv*40000.0*(60.0-r[i])/(57.0);
      delvt=rv*40000.0*(myr/kpc)*(60.0-r[i])/(57.0);
    }
    else
    {
        delvt=0.0;
    }

/* Total velocity + random radial + random tangential dispersion */

    vx[i]=vm*y[i]/r[i] + delv*x[i]/r[i] + delvt*y[i]/r[i];
    vy[i]=-vm*x[i]/r[i] + delv*y[i]/r[i] - delvt*x[i]/r[i];
    //     printf("i= %d vx= %e vy= %e \n",i,vx[i],vy[i]);
}



/*
fname[0]=0;
strcat(&fname[0],"data/vdat_");
strcat(fname,&fn[0]);
printf("%s\n",fname);

fp1=fopen(fname,"r");
```

```
for (i=0;i<n;i++)
   {
      fscanf(fp1,"%le %le %le %le\n",&x[i],&y[i],&vx[i],&vy[i]);
     //fprintf(fp1,"%le %le %le %le\n",&x[i],&y[i],&vx[i],&vy[i]);
     //      r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
    //            printf("%e %e %e %e %e\n",x[i],y[i],vx[i],vy[i],r[i]);
   }
 fclose(fp1);

*/

/*================================================*/
/* Compute Nearest Neighbor interactions if inn = 1*/
/*================================================*/
/* If nearest neighbor (inn=1), then */
/* update mass within radius r[i]. */
/* In order to speed up calculations, we */
/* don't do this every time step, only every idump */
/*    if (inn == 1) {
       for(i=0;i<n;i++)
       {
           rv=sqrt(x[i]*x[i] + y[i]*y[i]);
          rn = 0.5*(1.0 + rv/5.0);

           mt[i]=0.0;
           for(j=0;j<n;j++)
          {
                x2=(x[j] - x[i])*(x[j] - x[i]);
                y2=(y[j] - y[i])*(y[j] - y[i]);
                dr=sqrt(x2 + y2);
// if mass j is inside r[i] and OUTSIDE nearest neighbor radius rn
// then include it in the total mass within radius r[i]
              if(r[j] < r[i] && dr > rn)
                  {
                  mt[i] = m[j] + mt[i];
                   }
          }
          // Add in other mass contributions to get total
          mt[i]=m0 + mt[i] + mcdm[i];
       }
   }
*/


/*================================================*/
/* Compute velocity rotation curve*/
/*================================================*/
```

```
    /* use a finer grid from 1 - 5 kpc */
    for(j=0;j<16;j++)
    {
    /* Add one to put on 1 - 5 kpc range */
     rad = ((double) j + 1.0)*0.25;
     *(rvs+j)=rad;
    /* nvs counts the number of stars */
        nvs=0;
    /* initialize velocity to zero */
        *(vs+j)=0;
    /* loop over all stars and test if within +/- 0.5kpc of rad */
        for(i=0;i<n;i++)
        {
    /* is star j within +/- 0.125 kpc of rad?*/
         if ((*(r+i) > rad - 0.125) && (*(r+i) < rad + 0.125)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+j)=*(vs+j)+v2;
    /* count the number of stars in this range*/
          nvs=nvs+1;
           }
        }

        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
            dnvs = (double) nvs;
            *(vs+j)=*(vs+j)/dnvs;
    /* Now convert from kpc/myr to km/s */
            *(vs+j)=*(vs+j)*kpc/(myr*1000.0);
        }
        else {
    /* set to previous value if no particles were inside this radius range */
            *(vs+j)=*(vs+j-1);
         }
    }

    /* use a coarser grid from 5-30kpc */
    for(j=0;j<26;j++)
    {
     jp=j+16;
     rad = ((double) j)*1.0 + 5.0;
     *(rvs+jp)=rad;
    /* nvs counts the number of stars */
```

```
        nvs=0;
    /* initialize velocity to zero */
        *(vs+jp)=0;
    /* loop over all stars and test if within +/- 0.5kpc of rad */
        for(i=0;i<n;i++)
        {
    /* is star j within +/- 0.125 kpc of rad?*/
          if ((*(r+i) > rad - 0.5) && (*(r+i) < rad + 0.5)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+jp)=*(vs+jp)+v2;
    /* count the number of stars in this range*/
           nvs=nvs+1;
           }
        }

        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
           dnvs = (double) nvs;
           *(vs+jp)=*(vs+jp)/dnvs;
    /* Now convert from kpc/myr to km/s */
           *(vs+jp)=*(vs+jp)*kpc/(myr*1000.0);

        }
        else {
    /* Set to previous value if no stars are in this range of rad */
          *(vs+jp)=*(vs+jp-1);
        }
      }


    //velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

        fp1=fopen("velrot.mtv","w");
        fprintf(fp1,"%%xmin= 0\n");
        fprintf(fp1,"%%xmax= +30\n");
        fprintf(fp1,"%%ymin= 0\n");
        if(galaxy == 0) {
          fprintf(fp1,"%%ymax= +300\n");
        }
        if(galaxy == 1) {
          fprintf(fp1,"%%ymax= +200\n");
        }
```

```
    if(galaxy == 2) {
      fprintf(fp1,"%%ymax= +200\n");
    }
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linecolor= 3\n");
    fprintf(fp1,"%%linetype= 1\n");

    for (i=0;i<41;i++)
      {
       fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
      }
    fprintf(fp1,"\n");
    fclose(fp1);



/* Compute total energy */
/* don't put doubles in argument list!! */
/* Make them globals if unchanged inputs or arrays or function return if single value */

En[0]=0.0;
for(j=0;j<n;j++)
{
  vx2=(*(vx+j))*(*(vx+j));
  vy2=(*(vy+j))*(*(vy+j));
  vm2=vx2+vy2;

// Reset mt to be just m0 and mcdm for N-BODY!!!
  mt[j]=m0 + mcdm[j];

//CENTRAL mass = BLACK hole + CDM
  mass=mt[j];

  //  printf("r[j] = %e | drmin = %e\n",r[j],drmin);
  ///* Potential energy due to large central mass
  ///* use drmin to limit force/potential when r -> 0
  ///* central force drmin is 1/4 drmin to better model core
  U=-g*(*(m+j))*mass/sqrt((*(r+j))*(*(r+j)) + drmin2);

  //  printf("j= %d r= %e m = %e drmin= %f\n",j,*(r+j),*(m+j),drmin);

  //   printf("vm2= %f, mass= %f, U= %f\n",vm2,mass,U);
  ///* Add up potential energies due to all other mass points
    for(i=0;i<n;i++)
    {
```

```
        if (i != j)
        {
         x2=(*(x+j) - *(x+i))*(*(x+j) - *(x+i));
         y2=(*(y+j) - *(y+i))*(*(y+j) - *(y+i));
         dr=x2 + y2 + drmin;
          U=U-g*(*(m+j))*(*(m+i))/sqrt(dr);
         // printf("i=%d U= %e\n",i,U);
        }
    }
    En[0]=En[0] + 0.5*(*(m+j))*vm2 + U;
 }


//    eng(&En,&vx,&vy,&mt,&mcdm,&r,&x,&y,&m,n);
    En0=En[0];
    printf("Initial Energy En0= %e in Sm, kpc, myr units\n",En0);


/* Open file to store energy vs time */
    fpE=fopen("En.mtv","w");
    fprintf(fpE,"%%xmin= 0\n");
    fprintf(fpE,"%%xmax= +1000\n");
    fprintf(fpE,"%%ymin= 0\n");
    fprintf(fpE,"%%ymax= +2.0\n");
    fprintf(fpE,"%%xlabel= \"t (myr)\"\n");
    fprintf(fpE,"%%ylabel= \"E/E0\"\n");
    fprintf(fpE,"%%markertype= 13\n");
    fprintf(fpE,"%%markercolor= 0\n");
    fprintf(fpE,"%%linecolor= 0\n");
    fprintf(fpE,"%%linetype= 1\n");



/*=========================================*/
/* INITIALIZE OPENCL BUFFERS */
/*=========================================*/

  cl_mem bufx = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                               length*sizeof(cl_double), x, &ierr);

  cl_mem bufy = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                               length*sizeof(cl_double), y, &ierr);

  cl_mem bufm = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                               length*sizeof(cl_double), m, &ierr);

  cl_mem bufmt = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                               length*sizeof(cl_double), mt, &ierr);
```

```
 cl_mem bufxnw = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), xnw, &ierr);

 cl_mem bufynw = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), ynw, &ierr);

 cl_mem bufAx = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), Ax, &ierr);

 cl_mem bufAy = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), Ay, &ierr);

 cl_mem bufvx = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), vx, &ierr);

 cl_mem bufvy = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
                                length*sizeof(cl_double), vy, &ierr);


 if (ierr != 0 ) {
   printf ("ERROR: in CreateBuffer\n");
   printf ("ierr = %d\n",ierr);
   exit(1);}

 size_t global_work_size = length;
 size_t local_size = local;


ierr=clSetKernelArg(kernel, 4, sizeof(bufvx), (void *) &bufvx);
ierr=clSetKernelArg(kernel, 5, sizeof(bufvy), (void *) &bufvy);
ierr=clSetKernelArg(kernel, 6, sizeof(bufAx), (void *) &bufAx);
ierr=clSetKernelArg(kernel, 7, sizeof(bufAy), (void *) &bufAy);
ierr=clSetKernelArg(kernel, 8, sizeof(bufmt), (void *) &bufmt);
ierr=clSetKernelArg(kernel, 9, sizeof(bufm), (void *) &bufm);

/* LOCAL BUFFERS */
ierr=clSetKernelArg(kernel, 10, sizeof(cl_double)*local,NULL);
ierr=clSetKernelArg(kernel, 11, sizeof(cl_double)*local,NULL);
ierr=clSetKernelArg(kernel, 12, sizeof(cl_double)*local,NULL);



 if (ierr != 0 ) {
   printf ("ERROR: in SetKernelArg\n");
   printf ("ierr = %d\n",ierr);
   exit(1);}
```

```
/*=========================================*/
/* Start Galaxy evolution loop */
/*=========================================*/
t=0;
ik=0;

/* set file name counters to zero */
  i1=0;
  i2=0;
  i3=0;
  i4=0;

/* Initial Energy ratio = 1.0 */
fprintf(fpE,"%f %f\n",t,En0/En0);



  if (ierr != 0 ) {
    printf ("ERROR: in SetKernelArg\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}

/*=========================================*/
/* Start Galaxy evolution loop */
/*=========================================*/
t=0;
ik=0;

/* set file name counters to zero */
  i1=0;
  i2=0;
  i3=0;
  i4=0;

/* Initial Energy ratio = 1.0 */
fprintf(fpE,"%f %f\n",t,En0/En0);

/* Compute initial Acclerations of all mass points */

getaccel(n);

/* Need to go ahead and update the x values using the new v  and A */

    for(j=0;j<n;j++)
    {
        x[j] = x[j] + vx[j]*dt + 0.5*Ax[j]*dt*dt;
       y[j] = y[j] + vy[j]*dt + 0.5*Ay[j]*dt*dt;
    }
```

```
// ONLY NEED TO WRITE THE ONE THAT CHANGED!!
/* Write the Ax and Ay to global memory */
ierr=clEnqueueWriteBuffer(queue, bufAx, CL_TRUE, 0, length*sizeof(cl_double),Ax,0,NULL,NULL);
ierr=clEnqueueWriteBuffer(queue, bufAy, CL_TRUE, 0, length*sizeof(cl_double),Ay,0,NULL,NULL);

/* Write the Vx and Vy to global memory */
//ierr=clEnqueueWriteBuffer(queue, bufvx, CL_TRUE, 0, length*sizeof(cl_double),vx,0,NULL,NULL);
//ierr=clEnqueueWriteBuffer(queue, bufvy, CL_TRUE, 0, length*sizeof(cl_double),vy,0,NULL,NULL);

// ONLY NEED TO WRITE THE ONE THAT CHANGED!!
/* Write new x and y go global memory */
ierr=clEnqueueWriteBuffer(queue, bufx, CL_TRUE, 0, length*sizeof(cl_double),x,0,NULL,NULL);
ierr=clEnqueueWriteBuffer(queue, bufy, CL_TRUE, 0, length*sizeof(cl_double),y,0,NULL,NULL);

/* Write central masses to GPU global memory */
//ierr=clEnqueueWriteBuffer(queue, bufmt, CL_TRUE, 0, length*sizeof(cl_double),mt,0,NULL,NULL);


  if (ierr != 0 ) {
    printf ("ERROR: in EnqueueWriteBuffer\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}

  //TEST
/*   printf("NEW values:\n");
   for(j=0;j<n;j++)
     {
       printf("j= %d Ax= %e  Ay= %e \n",j,Ax[j],Ay[j]);
       printf("j= %d x= %e  y= %e \n",j,x[j],y[j]);
     }

return;
*/


printf(">running program..\n");

while (t<tmax)
{

t1=clock();

for(ii=0;ii<idumpmax;ii++) {

 ierr=clSetKernelArg(kernel, 0, sizeof(bufx), (void *) &bufx);
 ierr=clSetKernelArg(kernel, 1, sizeof(bufy), (void *) &bufy);
 ierr=clSetKernelArg(kernel, 2, sizeof(bufxnw), (void *) &bufxnw);
```

```
 ierr=clSetKernelArg(kernel, 3, sizeof(bufynw), (void *) &bufynw);

ierr=clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global_work_size, &local_size, 0, NULL,
NULL);


  if (ierr != 0 ) {
    printf ("ERROR: in Kernal\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}

  // clFinish(queue);


ierr=clSetKernelArg(kernel, 2, sizeof(bufx), (void *) &bufx);
ierr=clSetKernelArg(kernel, 3, sizeof(bufy), (void *) &bufy);
ierr=clSetKernelArg(kernel, 0, sizeof(bufxnw), (void *) &bufxnw);
ierr=clSetKernelArg(kernel, 1, sizeof(bufynw), (void *) &bufynw);

ierr=clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global_work_size, &local_size, 0, NULL,
NULL);


  if (ierr != 0 ) {
    printf ("ERROR: in Kernal\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}

  //    clFinish(queue); */

 }

printf("All kernels running..\n");

t=t+dt*idumpmax*2;

  if (ierr != 0 ) {
    printf ("ERROR: in EnqueueNDRangeKernel\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}

/* WAIT FOR ALL KERNELS TO FINISH */
  clFinish(queue);

/* GET DATA X, Y, VX and VY */

  ierr=clEnqueueReadBuffer(queue, bufx, CL_TRUE, 0, length*sizeof(cl_double),x,0,NULL,NULL);
  ierr=clEnqueueReadBuffer(queue, bufy, CL_TRUE, 0, length*sizeof(cl_double),y,0,NULL,NULL);
```

```
  ierr=clEnqueueReadBuffer(queue, bufvx, CL_TRUE, 0, length*sizeof(cl_double),vx,0,NULL,NULL);
  ierr=clEnqueueReadBuffer(queue, bufvy, CL_TRUE, 0, length*sizeof(cl_double),vy,0,NULL,NULL);

  if (ierr != 0 ) {
    printf ("ERROR: in EnqueueReadBuffer\n");
    printf ("ierr = %d\n",ierr);
    exit(1);}


t2=clock();
elapsed = ((double) (t2-t1))/CLOCKS_PER_SEC;
printf("Time= %f (sec)\n",elapsed);

//return;

  /* DUMP DATA to DISK */

       idump=0;
         printf("t (myr)= %f %%= %f\n",t,100.0*t/tmax);
         printf("Updating Mass...\n");
/* file name counter update */
      ik=ik+1;

      i1++;
      if (i1 % 10 == 0) {
        i1=0;
       i2++;
      if (i2 % 10 == 0) {
        i2=0;
       i3++;
      if (i3 % 10 == 0) {
        i3=0;
       i4++;
      }}}

//    printf ("File # %d = %d %d %d %d\n",ik,i4,i3,i2,i1);
/* make+

 string file name out of time step counters */
/* fn = 0000,00001,00002, etc */
    fn[0]=0;
    strcat(&fn[0],digit[i4]);
    strcat(&fn[0],digit[i3]);
    strcat(&fn[0],digit[i2]);
    strcat(&fn[0],digit[i1]);
    printf ("Fstring= %s\n",fn);

/* If nearest neighbor (inn=1), then */
```

```
/* update mass within radius r[i]. */
/* In order to speed up calculations, we */
/* don't do this every time step, only every idump */
/*=============================================*/
/* UPdate Nearest Neighbors every idump */
/*=============================================*/


/*=============================================*/
/* Compute Effective central mass = All Mass within a radius rkpc */
/*=============================================*/
for(i=0;i<n;i++)
{
/*THIS DARK MATTER DIST is NFW*/
/*static, spherically symmetric mass*/
/* mcdm is the integrated NFW CDM density x volume = total CDM within radius rkpc */
    r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
    rkpc=r[i];
    /* this gives an approximately linear mass increase in DISK region */
    /* it becomes flat at very large radius*/
    mcdm[i]=cdm*mcdm0*(-rkpc/(A + rkpc) - log(A) + log(A+rkpc));
    //     printf("i= %d mcdm = %e solar masses\n",i,mcdm[i]);

    //Total Mass for N-body is black hole + dark matter
    mt[i]=m0 + mcdm[i];

 }

/* UPDATE central masses in GPU global memory */
ierr=clEnqueueWriteBuffer(queue, bufmt, CL_TRUE, 0, length*sizeof(cl_double),mt,0,NULL,NULL);

/*=========================================*/
/* compute and save total energy*/
/* Total energy conservation      */

En[0]=0.0;
for(j=0;j<n;j++)
{
  vx2=(*(vx+j))*(*(vx+j));
  vy2=(*(vy+j))*(*(vy+j));
  vm2=vx2+vy2;

//CENTRAL mass = BLACK hole + CDM
  mass=m0 + *(mcdm+j);
  //  printf("r[j] = %e | drmin = %e\n",r[j],drmin);
  ///* Potential energy due to large central mass
  ///* use drmin to limit force/potential when r -> 0
  ///* central force drmin is 1/4 drmin to better model core
```

```
 U=-g*(*(m+j))*mass/sqrt((*(r+j))*(*(r+j)) + drmin2);

 //  printf("j= %d r= %e m = %e drmin= %f\n",j,*(r+j),*(m+j),drmin);

 //   printf("vm2= %f, mass= %f, U= %f\n",vm2,mass,U);
 ///* Add up potential energies due to all other mass points
   for(i=0;i<n;i++)
   {
       if (i != j)
       {
        x2=(*(x+j) - *(x+i))*(*(x+j) - *(x+i));
        y2=(*(y+j) - *(y+i))*(*(y+j) - *(y+i));
        dr=x2 + y2 + drmin;
         U=U-g*(*(m+j))*(*(m+i))/sqrt(dr);
        // printf("i=%d U= %e\n",i,U);
       }
   }
   En[0]=En[0] + 0.5*(*(m+j))*vm2 + U;
}

// eng(&En,&vx,&vy,&mt,&mcdm,&r,&x,&y,&m,n);

printf("Energy En/En0= %f\n",En[0]/En0);
fprintf(fpE,"%f %f\n",t,En[0]/En0);
fflush(fpE);

/*=============================================*/
/*Compute rotational velocity at this time step*/
//velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

/* use a finer grid from 1 - 5 kpc */
for(j=0;j<16;j++)
{
/* Add one to put on 1 - 5 kpc range */
 rad = ((double) j + 1.0)*0.25;
 *(rvs+j)=rad;
/* nvs counts the number of stars */
   nvs=0;
/* initialize velocity to zero */
   *(vs+j)=0;
/* loop over all stars and test if within +/- 0.5kpc of rad */
   for(i=0;i<n;i++)
   {
/* is star j within +/- 0.125 kpc of rad?*/
     if ((*(r+i) > rad - 0.125) && (*(r+i) < rad + 0.125)){
/* Project velocity of star j onto tangent direction */
   v2x=(*(vx+i))*(*(y+i))/(*(r+i));
   v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
```

```
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+j)=*(vs+j)+v2;
    /* count the number of stars in this range*/
          nvs=nvs+1;
           }
        }

        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
           dnvs = (double) nvs;
           *(vs+j)=*(vs+j)/dnvs;
    /* Now convert from kpc/myr to km/s */
           *(vs+j)=*(vs+j)*kpc/(myr*1000.0);
        }
        else {
    /* set to previous value if no particles were inside this radius range */
           *(vs+j)=*(vs+j-1);
         }
    }

    /* use a coarser grid from 5-30kpc */
    for(j=0;j<26;j++)
    {
     jp=j+16;
     rad = ((double) j)*1.0 + 5.0;
     *(rvs+jp)=rad;
    /* nvs counts the number of stars */
        nvs=0;
    /* initialize velocity to zero */
        *(vs+jp)=0;
    /* loop over all stars and test if within +/- 0.5kpc of rad */
        for(i=0;i<n;i++)
        {
    /* is star j within +/- 0.125 kpc of rad?*/
          if ((*(r+i) > rad - 0.5) && (*(r+i) < rad + 0.5)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+jp)=*(vs+jp)+v2;
    /* count the number of stars in this range*/
          nvs=nvs+1;
           }
        }
```

```
        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
            dnvs = (double) nvs;
            *(vs+jp)=*(vs+jp)/dnvs;
    /* Now convert from kpc/myr to km/s */
            *(vs+jp)=*(vs+jp)*kpc/(myr*1000.0);

        }
        else {
    /* Set to previous value if no stars are in this range of rad */
          *(vs+jp)=*(vs+jp-1);
        }
      }



        fname[0]=0;
        strcat(&fname[0],"data/vrot_");
        strcat(fname,&fn[0]);
        printf("%s\n",fname);

        fp1=fopen(fname,"w");
        fprintf(fp1,"$DATA=CURVE2D\n");
        fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",t);
        fprintf(fp1,"%%xmin= 0\n");
        fprintf(fp1,"%%xmax= +30\n");
        fprintf(fp1,"%%ymin= 0\n");
        fprintf(fp1,"%%ymax= +300\n");
        fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
        fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
        fprintf(fp1,"%%markertype= 13\n");
        fprintf(fp1,"%%linetype= 1\n");
        for (i=0;i<41;i++)
         {
          fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
         }
        fprintf(fp1,"\n");
        fclose(fp1);


    /*=======================================*/
    /* Dump data to disk here */
    /*=======================================*/

        fname[0]=0;
```

```
    strcat(&fname[0],"data/mass_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"Time = %4.2f (myr)\"\n",t);
    fprintf(fp1,"%%xmin= -100\n");
    fprintf(fp1,"%%xmax= +100\n");
    fprintf(fp1,"%%ymin= -100\n");
    fprintf(fp1,"%%ymax= +100\n");
    fprintf(fp1,"%%equalscale= T\n");
    fprintf(fp1,"%%xlabel= \"x (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"y (kpc)\"\n");
    fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markercolor= 3\n");
    fprintf(fp1,"%%linetype= 0\n");
    for (i=0;i<n;i++)
     {
      fprintf(fp1,"%f %f\n",x[i],y[i]);
     }
    fprintf(fp1,"\n");
    fclose(fp1);

/* Dump velocities  to disk */
    fname[0]=0;
    strcat(&fname[0],"data/vdat_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    for (i=0;i<n;i++)
     {
      fprintf(fp1,"%e %e %e %e\n",x[i],y[i],vx[i],vy[i]);
     }
    fprintf(fp1,"\n");
    fclose(fp1);


/*=======================================*/

/* END of time loop */
}


/*=========================================*/
/*END OF MAIN PROGRAM*/
/*=========================================*/
```

```
}


/* ======================================================== *
/* SUBROUTINES BELOW */
/* ======================================================== */

/*================================================*/
/* Compute Nearest Neighbor lists */
/* nn(i) is the number of nearest neighbors about ith mass point
/* jn stores the index for each neighbor about ith mass point
/*================================================*/

getaccel(int n)
{

    int i,j,iv,ind;
    double rsq,mass,g,f;
    double ex,ey,exi,eyi,dr,Fintx,Finty,fint;
    double Fx,Fy,mcdm,r;
/*gravity constant: N m^2/kg-2*/
//g = 6.673e-11;
//gravity in kpc, Sm, myr
g = 4.517017e-12;

   for(j=0;j<n;j++)
   {
       rsq = x[j]*x[j] + y[j]*y[j] + drmin2;
       r=sqrt(rsq);

/*THIS DARK MATTER DIST is NFW*/
/*static, spherically symmetric mass*/
/* mcdm is the integrated NFW CDM density x volume = total CDM within radius rkpc */
/* this gives an approximately linear mass increase in DISK region */
/* it becomes flat at very large radius to give a finite mass*/
//        mcdm=cdm*mcdm0*(-r/(A + r) - log(A) + log(A+r));

/*Total mass inside radius rkpc= Black hole + core + disK + CDM*/
/* if nearest neighbor (inn=1), then central mass includes mt */
/* if N-body (inn=0), then don't include it since it is in the full interaction term */
        mass=mt[j];

/* force due to gravity between mass point m[j] and central total mass, use drmin to cut off force if
rsq ->0 */
/* central force drmin is 1/4 drmin to better model core */
        f = (g*mass)/rsq;
```

```
/* cosine and sine */
        ex = x[j] / r;
        ey = y[j] / r;

/* Compute NEAREST NEIGHBOR force INTERACTIONS if inn == 1 else do N-Body (all interactions)*/
/* Add up ALL interactions N-Body */
         Fintx=0.0;
         Finty=0.0;
            for(i=0;i<n;i++)
            {
                dr=(x[j] - x[i])*(x[j] - x[i]) + (y[j]-y[i])*(y[j]-y[i]) + drmin;
                fint=g*m[i]/dr;
                dr=sqrt(dr);
                exi = (x[j]-x[i]) / dr;
                eyi = (y[j]-y[i]) / dr;
                Fintx=-fint*exi + Fintx;
                Finty=-fint*eyi + Finty;

            }


/* Total force due to Gravity of central mass + NN interactions*/
/* Fx and Fy really are acclerations since we have already divided by m[j] */
        Fx = -f * ex   + Fintx;
        Fy = -f * ey   + Finty;
/* Total acceleration of mass m[j] */
/* xn and yn is used to pass ax and ay to GPU at FIRST TIME STEP ONLY! */
        Ax[j] = Fx;
        Ay[j] = Fy;
    }
}
```

vcolor.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#define NMAX 4096


void main (void)
{
 double vx[NMAX],vy[NMAX],x[NMAX],y[NMAX],r[NMAX];
 double vmax,vmag,vsum,vtx,vty;
 double xdiff,ydiff,vxdiff,vydiff;
 double oneau,kpc,t;
 int  i,i1,i2,i3,i4,n,ik,ic;

static char *digit[]={"0","1","2","3","4","5","6","7","8","9"};
char fn[4];
char fname[14];

/* File pointers for data output */
 FILE *fp1,*fpE;

oneau = 149597871.0E3;
/* one kilo parsec*/
kpc = 2.06e8*oneau;

 n=4096;
// n=1100;
// n=260;
 i1=0;
 i2=0;
 i3=0;
 i4=0;

 // File Name Counter

 for (ik=1;ik<401;ik++) {

   t=(double) ik;

 /* set file to read */
     i1++;
     if (i1 % 10 == 0) {
       i1=0;
      i2++;
     if (i2 % 10 == 0) {
       i2=0;
```

```
      i3++;
     if (i3 % 10 == 0) {
       i3=0;
      i4++;
     }}}

   fn[0]=0;
   strcat(&fn[0],digit[i4]);
   strcat(&fn[0],digit[i3]);
   strcat(&fn[0],digit[i2]);
   strcat(&fn[0],digit[i1]);
   printf ("Fstring= %s\n",fn);

/* Read in velocities and positions */
    fname[0]=0;
    strcat(&fname[0],"data/vdat_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"r");
    for (i=0;i<n;i++)
     {
      fscanf(fp1,"%le %le %le %le\n",&x[i],&y[i],&vx[i],&vy[i]);
      r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
      //          printf("%d %e %e %e %e %e\n",i,x[i],y[i],vx[i],vy[i],r[i]);
     }
    fclose(fp1);

    printf("============================\n");

    fname[0]=0;
    strcat(&fname[0],"data/masC_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);
    fp1=fopen(fname,"w");
    fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"WITH Dark Matter\"\n");
    //fprintf(fp1,"%%toplabel= \"NO Dark Matter\"\n");
    fprintf(fp1,"%%subtitle= \"Time = %4.2f (myr)\"\n",t);
    fprintf(fp1,"%%xmin= -50\n");
    fprintf(fp1,"%%xmax= +50\n");
    fprintf(fp1,"%%ymin= -50\n");
    fprintf(fp1,"%%ymax= +50\n");
    fprintf(fp1,"%%equalscale= T\n");
    fprintf(fp1,"%%xlabel= \"x (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"y (kpc)\"\n");
    fprintf(fp1,"%%linetype= 0\n");
```

```
    for (i=0;i<n;i++)
     {

//compute tangential component
       vtx=vx[i]*y[i]/r[i];
       vty=-vy[i]*x[i]/r[i];
// Add up vtx and vty to get total tangential velocity
// Divide by 1000 to get km/
       vmag=(vtx + vty)/1000.0;
       //          printf("vmag= %f\n",vmag);

       //          printf ("vmag= %f\n",vmag);
//Set color depending upon radial velocity
       if (vmag > 200.0) {
         ic=4;}
         else if (vmag > 125.0) {
           ic=4;}
         else if (vmag > 100.0) {
           ic=4;}
          else if (vmag > 50.0) {
            ic=5;}
          else if (vmag > 25.0) {
            ic=3;}
         else {
           ic=0;}

        fprintf(fp1,"%%markertype= 1\n");
        fprintf(fp1,"%%markercolor= %d\n",ic);
       //fprintf(fp1,"%f %f\n",x[i],y[i]);
        fprintf(fp1,"%f %f %f\n",x[i],y[i],vmag);
        //printf("vmag = %f\n",vmag);
        fprintf(fp1,"\n");
      }
    fprintf(fp1,"\n");
    fclose(fp1);


 //Next time step
 }

    //END OF PROGRAM
}
```

vrots.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#define NMAX 200
#define NNMAX 200

 double * x = NULL;
 double * y = NULL;
 double * vx = NULL;
 double * vy = NULL;
 double * r = NULL;

void initmem()
{
size_t sizeInBytes = NMAX * sizeof(double);
  x=(double *) malloc(sizeInBytes);
  vx=(double *) malloc(sizeInBytes);
  vy=(double *) malloc(sizeInBytes);
  y=(double *) malloc(sizeInBytes);
  r=(double *) malloc(sizeInBytes);
}

void main (void)
{
  // double vx[NMAX],vy[NMAX],x[NMAX],y[NMAX],r[NMAX];
 double vmax,vmag,vsum,vs[120],rvs[120];
 double vs_sum[120];
 double xdiff,ydiff,vxdiff,vydiff;
 double oneau,kpc,t;
 int  i,i1,i2,i3,i4,n,ik,ic;

static char *digit[]={"0","1","2","3","4","5","6","7","8","9"};
char fn[4];
char fname[14];

/* File pointers for data output */
 FILE *fp1,*fpE;
 initmem();
oneau = 149597871.0E3;
/* one kilo parsec*/
kpc = 2.06e8*oneau;

 n=16384;
 i1=0;
 i2=0;
```

```
 i3=0;
 i4=0;

//Zero out vs_sum
 for (i=0;i<61;i++) {
   vs_sum[i]=0.0;}

 for (ik=1;ik<58;ik++) {

   t=(double) ik;


 /* set file to read */
      i1++;
      if (i1 % 10 == 0) {
        i1=0;
       i2++;
      if (i2 % 10 == 0) {
        i2=0;
       i3++;
      if (i3 % 10 == 0) {
        i3=0;
       i4++;
      }}}

    fn[0]=0;
    strcat(&fn[0],digit[i4]);
    strcat(&fn[0],digit[i3]);
    strcat(&fn[0],digit[i2]);
    strcat(&fn[0],digit[i1]);
    printf ("Fstring= %s\n",fn);

/* Read in velocities and positions */
     fname[0]=0;
     strcat(&fname[0],"data/vdat_");
     strcat(fname,&fn[0]);
     printf("%s\n",fname);

     fp1=fopen(fname,"r");
     for (i=0;i<n;i++)
       {
        fscanf(fp1,"%le %le %le %le\n",&x[i],&y[i],&vx[i],&vy[i]);
        r[i]=sqrt(x[i]*x[i] + y[i]*y[i]);
        //          printf("%e %e %e %e %e\n",x[i],y[i],vx[i],vy[i],r[i]);
       }
     fclose(fp1);

     printf("============================\n");
```

```
    velrot(n,&r,&vx,&vy,&vs,&rvs,&x,&y);

    fname[0]=0;
    strcat(&fname[0],"data/vroC_");
    strcat(fname,&fn[0]);
    printf("%s\n",fname);

    fp1=fopen(fname,"w");
    //     fprintf(fp1,"$DATA=CURVE2D\n");
    fprintf(fp1,"%%toplabel= \"WITH Dark Matter\"\n");
    //fprintf(fp1,"%%toplabel= \"NO Dark Matter\"\n");
    fprintf(fp1,"%%subtitle= \"Time = %4.2f (myr)\"\n",t);
    fprintf(fp1,"%%xmin= 0\n");
    fprintf(fp1,"%%xmax= +30\n");
    fprintf(fp1,"%%ymin= 0\n");
    fprintf(fp1,"%%ymax= +200\n");
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    //     fprintf(fp1,"%%markertype= 13\n");
    fprintf(fp1,"%%markertype= 3\n");
    fprintf(fp1,"%%markercolor= 0\n");
    fprintf(fp1,"%%linetype= 1\n");
    fprintf(fp1,"%%linecolor= 0\n");

    for (i=0;i<61;i++)
     {

//Set color depending upon radial velocity
       if (vs[i] > 200.0) {
         ic=4;}
         else if (vs[i] > 150.0) {
           ic=3;}
         else if (vs[i] > 100.0) {
           ic=5;}
         else {
           ic=0;}

//NO COLOR CODE (B&W only)
       ic=0;

        fprintf(fp1,"%%linetype= 0\n");
       //fprintf(fp1,"%%markertype= 13\n");
       fprintf(fp1,"%%markertype= 4\n");
        fprintf(fp1,"%%markercolor= %d\n",ic);
       fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
        fprintf(fp1,"\n");
```

```
/*
       // TURN OFF lines
       if (i != 40) {
//Plot line half-way on RIGHT side of data point
        fprintf(fp1,"%%markertype= 0\n");
        fprintf(fp1,"%%linetype= 1\n");
        fprintf(fp1,"%%linewidth= 2\n");
        fprintf(fp1,"%%linecolor= %d\n",ic);
       fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
       fprintf(fp1,"%f %f\n",0.5*(rvs[i]+rvs[i+1]),0.5*(vs[i]+vs[i+1]));
        fprintf(fp1,"\n");
       }

       if (i != 0) {
//Plot line half-way on LEFT side of data point
        fprintf(fp1,"%%markertype= 0\n");
        fprintf(fp1,"%%linetype= 1\n");
        fprintf(fp1,"%%linewidth= 2\n");
        fprintf(fp1,"%%linecolor= %d\n",ic);
       fprintf(fp1,"%f %f\n",rvs[i],vs[i]);
       fprintf(fp1,"%f %f\n",0.5*(rvs[i]+rvs[i-1]),0.5*(vs[i]+vs[i-1]));
        fprintf(fp1,"\n");
       }
//
*/

/* Add up to compute average */
       vs_sum[i]=vs_sum[i] + vs[i];


     }

   fprintf(fp1,"\n");
   fclose(fp1);


 //Next time step
 }

//Write Avg of data for plotting
 printf("dumping avg to vavg.mtv");
   fp1=fopen("vavg.mtv","w");
   fprintf(fp1,"\n");
   //    fprintf(fp1,"%%toplabel= \"Average Rotational Velocity\"\n");
   //    fprintf(fp1,"%%subtitle= \"Time = %4.2f (myr)\"\n",t);
   fprintf(fp1,"%%xmin= 0\n");
   fprintf(fp1,"%%xmax= +30\n");
   fprintf(fp1,"%%ymin= 0\n");
```

```
    fprintf(fp1,"%%ymax= +350\n");
    fprintf(fp1,"%%xlabel= \"r (kpc)\"\n");
    fprintf(fp1,"%%ylabel= \"v (km/s)\"\n");
    fprintf(fp1,"%%markertype= 0\n");
    fprintf(fp1,"%%linetype= 1\n");
    fprintf(fp1,"%%linecolor= 0\n");
    fprintf(fp1,"%%linewidth= 3\n");

    for (i=0;i<61;i++)
     {
    fprintf(fp1,"%f %f\n",rvs[i],vs_sum[i]/400.0);
     }

    fprintf(fp1,"\n");
    fclose(fp1);

    //END OF PROGRAM
}

/*================================================*/
/* Compute velocity rotation curve*/
/* Average all the star tangential velocities within +0.5 and -0.5 kpc */
/* of radius rad=1-30, returned in vs(0-29) */
/*================================================*/

velrot(int n,double *r,double *vx,double *vy,double *vs,double *rvs,double *x,double *y)
{
int j,i,nvs,jp;
double rad,oneau,kpc,dnvs,v2,v2x,v2y;

/* one kilo parsec*/
oneau = 149597871.0E3;
kpc = 2.06e8*oneau;
/* set to 1.0 since x and y are already in kpc here */
 kpc=1.0;

for(j=0;j<61;j++)
{
  jp=j;
  rad = ((double) j)*0.5;
  *(rvs+jp)=rad;
/* nvs counts the number of stars */
    nvs=0;
/* initialize velocity to zero */
    *(vs+jp)=0;
/* loop over all stars and test if within +/- 0.5kpc of rad */
    for(i=0;i<n;i++)
    {
```

```
    /* is star j within +/- 0.125 kpc of rad?*/
          if ((*(r+i) / kpc > rad - 0.25) && (*(r+i) / kpc < rad + 0.25)){
    /* Project velocity of star j onto tangent direction */
        v2x=(*(vx+i))*(*(y+i))/(*(r+i));
        v2y=-(*(vy+i))*(*(x+i))/(*(r+i));
        v2=v2x + v2y;
    /* Add up the tangential velocities */
        *(vs+jp)=*(vs+jp)+v2;
    /* count the number of stars in this range*/
          nvs=nvs+1;
           }
        }

        if(nvs > 0)
        {
    /* COMPUTE AVERAGE tangential VELOCITY at radius rad by dividing by the total number stars
there*/
    /* divide by 1000 to get km/sec*/
            dnvs = (double) nvs;
            *(vs+jp)=*(vs+jp)/(1000.0*dnvs);
        }
        else {
    /* Set to previous value if no stars are in this range of rad */
          *(vs+jp)=*(vs+jp-1);
        }
     }


    }
```