**nbody.cpp**

```cpp
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"

void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {

    assert(eng < 0.0);
    {
        PS::MTTS mt;
        mt.init_genrand(0);
        for(PS::S32 i = 0; i < n_loc; i++){
            mass[i] = mass_glb / n_glb;
            const PS::F64 radius = 3.0;
            do {
                pos[i][0] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][1] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][2] = (2. * mt.genrand_res53() - 1.) * radius;
            }while(pos[i] * pos[i] >= radius * radius);
            vel[i][0] = 0.0;
            vel[i][1] = 0.0;
            vel[i][2] = 0.0;
        }
    }

    PS::F64vec cm_pos  = 0.0;
    PS::F64vec cm_vel  = 0.0;
    PS::F64    cm_mass = 0.0;
    for(PS::S32 i = 0; i < n_loc; i++){
        cm_pos  += mass[i] * pos[i];
        cm_vel  += mass[i] * vel[i];
        cm_mass += mass[i];
    }
    cm_pos /= cm_mass;
    cm_vel /= cm_mass;
    for(PS::S32 i = 0; i < n_loc; i++){
        pos[i] -= cm_pos;
        vel[i] -= cm_vel;
    }
}

template<class Tpsys>
void setParticlesColdUniformSphere(Tpsys & psys,
                                   const PS::S32 n_glb,
                                   PS::S32 & n_loc) {

    n_loc = n_glb;
    psys.setNumberOfParticleLocal(n_loc);

    PS::F64    * mass = new PS::F64[n_loc];
    PS::F64vec * pos  = new PS::F64vec[n_loc];
    PS::F64vec * vel  = new PS::F64vec[n_loc];
    const PS::F64 m_tot = 1.0;
    const PS::F64 eng   = -0.25;
    makeColdUniformSphere(m_tot, n_glb, n_loc, mass, pos, vel, eng);
    for(PS::S32 i = 0; i < n_loc; i++){
```

**nbody.cpp**

```cpp
        psys[i].mass = mass[i];
        psys[i].pos  = pos[i];
        psys[i].vel  = vel[i];
        psys[i].id   = i;
    }
    delete [] mass;
    delete [] pos;
    delete [] vel;
}

template<class Tpsys>
void kick(Tpsys & system,
          const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].vel  += system[i].acc * dt;
    }
}

template<class Tpsys>
void drift(Tpsys & system,
           const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].pos  += system[i].vel * dt;
    }
}

template<class Tpsys>
void calcEnergy(const Tpsys & system,
                PS::F64 & etot,
                PS::F64 & ekin,
                PS::F64 & epot,
                const bool clear=true){
    if(clear){
        etot = ekin = epot = 0.0;
    }
    PS::F64 etot_loc = 0.0;
    PS::F64 ekin_loc = 0.0;
    PS::F64 epot_loc = 0.0;
    const PS::S32 nbody = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < nbody; i++){
        ekin_loc += system[i].mass * system[i].vel * system[i].vel;
        epot_loc += system[i].mass * (system[i].pot + system[i].mass / FPGrav::eps);
    }
    ekin_loc *= 0.5;
    epot_loc *= 0.5;
    etot_loc  = ekin_loc + epot_loc;
#ifdef PARTICLE_SIMULATOR_MPI_PARALLEL
    etot = PS::Comm::getSum(etot_loc);
    epot = PS::Comm::getSum(epot_loc);
    ekin = PS::Comm::getSum(ekin_loc);
#else
    etot = etot_loc;
    epot = epot_loc;
    ekin = ekin_loc;
#endif
}

void printHelp() {
    std::cerr<<"o: dir name of output (default: ./result)"<<std::endl;
    std::cerr<<"t: theta (default: 0.5)"<<std::endl;
    std::cerr<<"T: time_end (default: 10.0)"<<std::endl;
    std::cerr<<"s: time_step (default: 1.0 / 128.0)"<<std::endl;
    std::cerr<<"d: dt_diag (default: 1.0 / 8.0)"<<std::endl;
    std::cerr<<"D: dt_snap (default: 1.0)"<<std::endl;
    std::cerr<<"l: n_leaf_limit (default: 8)"<<std::endl;
    std::cerr<<"n: n_group_limit (default: 64)"<<std::endl;
    std::cerr<<"N: n_tot (default: 1024)"<<std::endl;
    std::cerr<<"h: help"<<std::endl;
}
```

## nbody.cpp

```cpp

void makeOutputDirectory(char * dir_name) {
    struct stat st;
    if(stat(dir_name, &st) != 0) {
        PS::S32 ret_loc = 0;
        PS::S32 ret     = 0;
        if(PS::Comm::getRank() == 0)
            ret_loc = mkdir(dir_name, 0777);
        PS::Comm::broadcast(&ret_loc, ret);
        if(ret == 0) {
            if(PS::Comm::getRank() == 0)
                fprintf(stderr, "Directory \"%s\" is successfully made.\n", dir_name);
        } else {
            fprintf(stderr, "Directory %s fails to be made.\n", dir_name);
            PS::Abort();
        }
    }
}

PS::F64 FPGrav::eps = 1.0/32.0;

int main(int argc, char *argv[]) {
    std::cout<<std::setprecision(15);
    std::cerr<<std::setprecision(15);

    PS::Initialize(argc, argv);
    PS::F32 theta = 0.5;
    PS::S32 n_leaf_limit = 8;
    PS::S32 n_group_limit = 64;
    PS::F32 time_end = 10.0;
    PS::F32 dt = 1.0 / 128.0;
    PS::F32 dt_diag = 1.0 / 8.0;
    PS::F32 dt_snap = 1.0;
    char dir_name[1024];
    PS::S64 n_tot = 1024;
    PS::S32 c;
    sprintf(dir_name,"./result");
    opterr = 0;
    while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
        switch(c){
        case 'o':
            sprintf(dir_name,optarg);
            break;
        case 't':
            theta = atof(optarg);
            std::cerr << "theta =" << theta << std::endl;
            break;
        case 'T':
            time_end = atof(optarg);
            std::cerr << "time_end = " << time_end << std::endl;
            break;
        case 's':
            dt = atof(optarg);
            std::cerr << "time_step = " << dt << std::endl;
            break;
        case 'd':
            dt_diag = atof(optarg);
            std::cerr << "dt_diag = " << dt_diag << std::endl;
            break;
        case 'D':
            dt_snap = atof(optarg);
            std::cerr << "dt_snap = " << dt_snap << std::endl;
            break;
        case 'l':
            n_leaf_limit = atoi(optarg);
            std::cerr << "n_leaf_limit = " << n_leaf_limit << std::endl;
            break;
        case 'n':
            n_group_limit = atoi(optarg);
            std::cerr << "n_group_limit = " << n_group_limit << std::endl;
            break;
        case 'N':
```

## nbody.cpp

```cpp
            n_tot = atoi(optarg);
            std::cerr << "n_tot = " << n_tot << std::endl;
            break;
        case 'h':
            if(PS::Comm::getRank() == 0) {
                printHelp();
            }
            PS::Finalize();
            return 0;
        default:
            if(PS::Comm::getRank() == 0) {
                std::cerr<<"No such option! Available options are here."<<std::endl;
                printHelp();
            }
            PS::Abort();
        }
    }

    makeOutputDirectory(dir_name);

    std::ofstream fout_eng;

    if(PS::Comm::getRank() == 0) {
        char sout_de[1024];
   sprintf(sout_de, "%s/t-de.dat", dir_name);
        fout_eng.open(sout_de);
        fprintf(stdout, "This is a sample program of N-body simulation on FDPS!\n");
        fprintf(stdout, "Number of processes: %d\n", PS::Comm::getNumberOfProc());
        fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
    }

    PS::ParticleSystem<FPGrav> system_grav;
    system_grav.initialize();
    PS::S32 n_loc    = 0;
    PS::F32 time_sys = 0.0;
    if(PS::Comm::getRank() == 0) {
        setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
    } else {
        system_grav.setNumberOfParticleLocal(n_loc);
    }

    const PS::F32 coef_ema = 0.3;
    PS::DomainInfo dinfo;
    dinfo.initialize(coef_ema);
    dinfo.decomposeDomainAll(system_grav);
    system_grav.exchangeParticle(dinfo);
    n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif

    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
#ifdef MULTI_WALK
    const PS::S32 n_walk_limit = 200;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
                                                RetrieveKernel,
                                                tag_max,
                                                system_grav,
                                                dinfo,
                                                n_walk_limit);
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                       CalcGravity<PS::SPJMonopole>,
                                       system_grav,
                                       dinfo);
#endif
```

**nbody.cpp**

```cpp
    PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
    calcEnergy(system_grav, Etot0, Ekin0, Epot0);
    PS::F64 time_diag = 0.0;
    PS::F64 time_snap = 0.0;
    PS::S64 n_loop = 0;
    PS::S32 id_snap = 0;
    while(time_sys < time_end){
        if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
            char filename[256];
            sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
            FileHeader header;
            header.time   = time_sys;
            header.n_body = system_grav.getNumberOfParticleGlobal();
            system_grav.writeParticleAscii(filename, header);
            time_snap += dt_snap;
        }

        calcEnergy(system_grav, Etot1, Ekin1, Epot1);

        if(PS::Comm::getRank() == 0){
            if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys) ){
                fout_eng << time_sys << " " << (Etot1 - Etot0) / Etot0 << std::endl;
                fprintf(stdout, "time: %10.7f energy error: %+e\n",
                        time_sys, (Etot1 - Etot0) / Etot0);
                time_diag += dt_diag;
            }
        }


        kick(system_grav, dt * 0.5);

        time_sys += dt;
        drift(system_grav, dt);

        if(n_loop % 4 == 0){
            dinfo.decomposeDomainAll(system_grav);
        }

        system_grav.exchangeParticle(dinfo);
#ifdef MULTI_WALK
        tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
                                                    RetrieveKernel,
                                                    tag_max,
                                                    system_grav,
                                                    dinfo,
                                                    n_walk_limit,
                                                    true);
#else
        tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                           CalcGravity<PS::SPJMonopole>,
                                           system_grav,
                                           dinfo);
#endif

        kick(system_grav, dt * 0.5);

        n_loop++;
    }

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_close();
#endif

    PS::Finalize();
    return 0;
}
```