

## nbody.cpp

```

1  #include<iostream>
2  #include<fstream>
3  #include<unistd.h>
4  #include<sys/stat.h>
5  #include<particle_simulator.hpp>
6  #ifdef ENABLE_PHANTOM_GRAPE_X86
7  #include <gp5util.h>
8  #endif
9  #ifdef ENABLE_GPU_CUDA
10 #define MULTI_WALK
11 #include "force_gpu_cuda.hpp"
12 #endif
13 #include "user-defined.hpp"
14
15 void makeColdUniformSphere(const PS::F64 mass_glb,
16                           const PS::S64 n_glb,
17                           const PS::S64 n_loc,
18                           PS::F64 * & mass,
19                           PS::F64vec * & pos,
20                           PS::F64vec * & vel,
21                           const PS::F64 eng = -0.25,
22                           const PS::S32 seed = 0) {
23
24     assert(eng < 0.0);
25     {
26         PS::MTTS mt;
27         mt.init_genrand(0);
28         for(PS::S32 i = 0; i < n_loc; i++){
29             mass[i] = mass_glb / n_glb;
30             const PS::F64 radius = 3.0;
31             do {
32                 pos[i][0] = (2. * mt.genrand_res53() - 1.) * radius;
33                 pos[i][1] = (2. * mt.genrand_res53() - 1.) * radius;
34                 pos[i][2] = (2. * mt.genrand_res53() - 1.) * radius;
35             }while(pos[i] * pos[i] >= radius * radius);
36             vel[i][0] = 0.0;
37             vel[i][1] = 0.0;
38             vel[i][2] = 0.0;
39         }
40     }
41
42     PS::F64vec cm_pos = 0.0;
43     PS::F64vec cm_vel = 0.0;
44     PS::F64 cm_mass = 0.0;
45     for(PS::S32 i = 0; i < n_loc; i++){
46         cm_pos += mass[i] * pos[i];
47         cm_vel += mass[i] * vel[i];
48         cm_mass += mass[i];
49     }
50     cm_pos /= cm_mass;
51     cm_vel /= cm_mass;
52     for(PS::S32 i = 0; i < n_loc; i++){
53         pos[i] -= cm_pos;
54         vel[i] -= cm_vel;
55     }
56 }
57
58 template<class Tpsys>
59 void setParticlesColdUniformSphere(Tpsys & psys,
60                                   const PS::S32 n_glb,
61                                   PS::S32 & n_loc) {
62
63     n_loc = n_glb;
64     psys.setNumberOfParticleLocal(n_loc);
65
66     PS::F64 * mass = new PS::F64[n_loc];
67     PS::F64vec * pos = new PS::F64vec[n_loc];
68     PS::F64vec * vel = new PS::F64vec[n_loc];
69     const PS::F64 m_tot = 1.0;
70     const PS::F64 eng = -0.25;
71     makeColdUniformSphere(m_tot, n_glb, n_loc, mass, pos, vel, eng);
72     for(PS::S32 i = 0; i < n_loc; i++){

```

## nbody.cpp

```

73     psys[i].mass = mass[i];
74     psys[i].pos = pos[i];
75     psys[i].vel = vel[i];
76     psys[i].id = i;
77 }
78 delete [] mass;
79 delete [] pos;
80 delete [] vel;
81 }
82
83 template<class Tpsys>
84 void kick(Tpsys & system,
85          const PS::F64 dt) {
86     PS::S32 n = system.getNumberOfParticleLocal();
87     for(PS::S32 i = 0; i < n; i++) {
88         system[i].vel += system[i].acc * dt;
89     }
90 }
91
92 template<class Tpsys>
93 void drift(Tpsys & system,
94           const PS::F64 dt) {
95     PS::S32 n = system.getNumberOfParticleLocal();
96     for(PS::S32 i = 0; i < n; i++) {
97         system[i].pos += system[i].vel * dt;
98     }
99 }
100
101 template<class Tpsys>
102 void calcEnergy(const Tpsys & system,
103                PS::F64 & etot,
104                PS::F64 & ekin,
105                PS::F64 & epot,
106                const bool clear=true){
107     if(clear){
108         etot = ekin = epot = 0.0;
109     }
110     PS::F64 etot_loc = 0.0;
111     PS::F64 ekin_loc = 0.0;
112     PS::F64 epot_loc = 0.0;
113     const PS::S32 nbody = system.getNumberOfParticleLocal();
114     for(PS::S32 i = 0; i < nbody; i++){
115         ekin_loc += system[i].mass * system[i].vel * system[i].vel;
116         epot_loc += system[i].mass * (system[i].pot + system[i].mass / FPGR
117 av::eps);
118     }
119     ekin_loc *= 0.5;
120     epot_loc *= 0.5;
121     etot_loc = ekin_loc + epot_loc;
122     #ifdef PARTICLE_SIMULATOR_MPI_PARALLEL
123     etot = PS::Comm::getSum(etot_loc);
124     epot = PS::Comm::getSum(epot_loc);
125     ekin = PS::Comm::getSum(ekin_loc);
126 #else
127     etot = etot_loc;
128     epot = epot_loc;
129     ekin = ekin_loc;
130 #endif
131 }
132
133 void printHelp() {
134     std::cerr<<"o: dir name of output (default: ./result)"<<std::endl;
135     std::cerr<<"t: theta (default: 0.5)"<<std::endl;
136     std::cerr<<"T: time_end (default: 10.0)"<<std::endl;
137     std::cerr<<"s: time_step (default: 1.0 / 128.0)"<<std::endl;
138     std::cerr<<"d: dt_diag (default: 1.0 / 8.0)"<<std::endl;
139     std::cerr<<"D: dt_snap (default: 1.0)"<<std::endl;
140     std::cerr<<"l: n_leaf_limit (default: 8)"<<std::endl;
141     std::cerr<<"n: n_group_limit (default: 64)"<<std::endl;
142     std::cerr<<"N: n_tot (default: 1024)"<<std::endl;
143     std::cerr<<"h: help"<<std::endl;

```

## nbody.cpp

```

144
145 void makeOutputDirectory(char * dir_name) {
146     struct stat st;
147     if(stat(dir_name, &st) != 0) {
148         PS::S32 ret_loc = 0;
149         PS::S32 ret = 0;
150         if(PS::Comm::getRank() == 0)
151             ret_loc = mkdir(dir_name, 0777);
152         PS::Comm::broadcast(&ret_loc, ret);
153         if(ret == 0) {
154             if(PS::Comm::getRank() == 0)
155                 fprintf(stderr, "Directory \"%s\" is successfully made.\n", dir_name);
156         } else {
157             fprintf(stderr, "Directory %s fails to be made.\n", dir_name);
158             PS::Abort();
159         }
160     }
161 }
162
163 PS::F64 FPGrav::eps = 1.0/32.0;
164
165 int main(int argc, char *argv[]) {
166     std::cout<<std::setprecision(15);
167     std::cerr<<std::setprecision(15);
168
169     PS::Initialize(argc, argv);
170     PS::F32 theta = 0.5;
171     PS::S32 n_leaf_limit = 8;
172     PS::S32 n_group_limit = 64;
173     PS::F32 time_end = 10.0;
174     PS::F32 dt = 1.0 / 128.0;
175     PS::F32 dt_diag = 1.0 / 8.0;
176     PS::F32 dt_snap = 1.0;
177     char dir_name[1024];
178     PS::S64 n_tot = 1024;
179     PS::S32 c;
180     sprintf(dir_name, "/result");
181     opterr = 0;
182     while((c=getopt(argc, argv, "i:o:d:D:t:T:l:n:N:hs:")) != -1){
183         switch(c){
184             case 'o':
185                 sprintf(dir_name, optarg);
186                 break;
187             case 't':
188                 theta = atof(optarg);
189                 std::cerr << "theta=" << theta << std::endl;
190                 break;
191             case 'T':
192                 time_end = atof(optarg);
193                 std::cerr << "time_end=" << time_end << std::endl;
194                 break;
195             case 's':
196                 dt = atof(optarg);
197                 std::cerr << "time_step=" << dt << std::endl;
198                 break;
199             case 'd':
200                 dt_diag = atof(optarg);
201                 std::cerr << "dt_diag=" << dt_diag << std::endl;
202                 break;
203             case 'D':
204                 dt_snap = atof(optarg);
205                 std::cerr << "dt_snap=" << dt_snap << std::endl;
206                 break;
207             case 'l':
208                 n_leaf_limit = atoi(optarg);
209                 std::cerr << "n_leaf_limit=" << n_leaf_limit << std::endl;
210                 break;
211             case 'n':
212                 n_group_limit = atoi(optarg);
213                 std::cerr << "n_group_limit=" << n_group_limit << std::endl;
214                 break;
215             case 'N':

```

## nbody.cpp

```

216         n_tot = atoi(optarg);
217         std::cerr << "n_tot=" << n_tot << std::endl;
218         break;
219     case 'h':
220         if(PS::Comm::getRank() == 0) {
221             printHelp();
222         }
223         PS::Finalize();
224         return 0;
225     default:
226         if(PS::Comm::getRank() == 0) {
227             std::cerr<<"No such option! Available options are here."<<std::endl;
228             printHelp();
229         }
230         PS::Abort();
231     }
232 }
233
234 makeOutputDirectory(dir_name);
235
236 std::ofstream fout_eng;
237
238 if(PS::Comm::getRank() == 0) {
239     char sout_de[1024];
240     sprintf(sout_de, "%s/t-de.dat", dir_name);
241     fout_eng.open(sout_de);
242     fprintf(stdout, "This is a sample program of N-body simulation on FDPs!\n");
243     fprintf(stdout, "Number of processes: %d\n", PS::Comm::getNumberOfProc());
244     fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfT
hread());
245 }
246
247 PS::ParticleSystem<FPGrav> system_grav;
248 system_grav.initialize();
249 PS::S32 n_loc = 0;
250 PS::F32 time_sys = 0.0;
251 if(PS::Comm::getRank() == 0) {
252     setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
253 } else {
254     system_grav.setNumberOfParticleLocal(n_loc);
255 }
256
257 const PS::F32 coef_ema = 0.3;
258 PS::DomainInfo dinfo;
259 dinfo.initialize(coef_ema);
260 dinfo.decomposeDomainAll(system_grav);
261 system_grav.exchangeParticle(dinfo);
262 n_loc = system_grav.getNumberOfParticleLocal();
263
264 #ifdef ENABLE_PHANTOM_GRAPE_X86
265     g5_open();
266     g5_set_eps_to_all(FPGrav::eps);
267 #endif
268
269 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
270 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
271 #ifdef MULTI_WALK
272     const PS::S32 n_walk_limit = 200;
273     const PS::S32 tag_max = 1;
274     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
RetrieveKernel,
tag_max,
system_grav,
dinfo,
n_walk_limit);
275 #else
276     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
CalcGravity<PS::SPJMonopole>,
system_grav,
dinfo);
277 #endif
278
279
280
281
282
283
284
285

```

## nbody.cpp

```
286     PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
287     calcEnergy(system_grav, Etot0, Ekin0, Epot0);
288     PS::F64 time_diag = 0.0;
289     PS::F64 time_snap = 0.0;
290     PS::S64 n_loop = 0;
291     PS::S32 id_snap = 0;
292     while(time_sys < time_end){
293         if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (t
ime_snap - time_sys) ){
294             char filename[256];
295             sprintf(filename, "%s%04d.dat", dir_name, id_snap++);
296             FileHeader header;
297             header.time = time_sys;
298             header.n_body = system_grav.getNumberOfParticleGlobal();
299             system_grav.writeParticleAscii(filename, header);
300             time_snap += dt_snap;
301         }
302
303         calcEnergy(system_grav, Etot1, Ekin1, Epot1);
304
305         if(PS::Comm::getRank() == 0){
306             if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag )
> (time_diag - time_sys) ){
307                 fout_eng << time_sys << " " << (Etot1 - Etot0) / Etot0 <<
std::endl;
308                 fprintf(stdout, "time: %10.7f energy error: %e\n",
309                     time_sys, (Etot1 - Etot0) / Etot0);
310                 time_diag += dt_diag;
311             }
312         }
313
314         kick(system_grav, dt * 0.5);
315
316         time_sys += dt;
317         drift(system_grav, dt);
318
319         if(n_loop % 4 == 0){
320             dinfo.decomposeDomainAll(system_grav);
321         }
322
323         system_grav.exchangeParticle(dinfo);
324 #ifdef MULTI_WALK
325         tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
326             RetrieveKernel,
327             tag_max,
328             system_grav,
329             dinfo,
330             n_walk_limit,
331             true);
332 #else
333         tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
334             CalcGravity<PS::SPJMonopole>,
335             system_grav,
336             dinfo);
337 #endif
338
339         kick(system_grav, dt * 0.5);
340
341         n_loop++;
342     }
343
344 #ifdef ENABLE_PHANTOM_GRAPE_X86
345     g5_close();
346 #endif
347
348     PS::Finalize();
349     return 0;
350 }
351 }
```