

FDPS講習会

サンプルコード編

細野 七月

京都大学, 理研 AICS

コード構成

●ユーザーが書くべきもの

- #include <particle_simulator.h>
- 粒子クラスと必要なメンバ関数
- 相互作用関数
- 時間積分ルーチン
- I/O (粒子クラスのI/Oと、FileHeaderクラス)

サンプルコード

- 今回のサンプルコードの内容
 - 流体 (Smoothed Particle Hydrodynamics)
 - 重力 (w/ and w/o Phantom-GRAPE (Tanikawa+, 2011; 2012))
- 本スライドでは、重力を計算するコードを取り上げて解説する。
 - ・ 時間積分法はleap-frog法
 - ・ 初期条件はその場生成(ファイル読み込みではない)
 - ・ ファイル構成
 - user-defined.hpp
 - nbody.cpp

user-defined.hpp

Lines (151-310) | 3.95 KB

```
1  #pragma once
2  class FileHeader{
3  public:
4      PS::S64 n_body;
5      PS::F64 time;
6      PS::S32 readAscii(FILE * fp) {
7          fscanf(fp, "%lf\n", &time);
8          fscanf(fp, "%lld\n", &n_body);
9          return n_body;
10     }
11     void writeAscii(FILE* fp) const {
12         fprintf(fp, "%e\n", time);
13         fprintf(fp, "%lld\n", n_body);
14     }
15 };
16
17 class FPGrav{
18 public:
19     PS::S64 id;
20     PS::F64 mass;
21     PS::F64vec pos;
22     PS::F64vec vel;
23     PS::F64vec acc;
24     PS::F64
```

user-defined.hpp

Lines (131 slots) | 3.95 KB

```
1 #pragma once
2 class FileHeader{
3 public:
4     PS::S64 n_body;
5     PS::F64 time;
6     PS::S32 readAscii(FILE * fp) {
7         fscanf(fp, "%lf\n", &time);
8         fscanf(fp, "%lld\n", &n_body);
9         return n_body;
10    }
11    void writeAscii(FILE* fp) const {
12        fprintf(fp, "%e\n", time);
13        fprintf(fp, "%lld\n", n_body);
14    }
15 };
16
17 class FPGrav{
18 public:
19     PS::S64 id;
20     PS::F64 mass;
21     PS::F64vec pos;
22     PS::F64vec vel;
23     PS::F64vec acc;
24     PS::F64vec pos;
```

クラス

user-defined.hpp

```
1 #pragma once
2 class FileHeader{
3 public:
4     PS::S64 n_body;
5     PS::F64 time;
6     PS::S32 readAscii(FILE * fp) {
7         fscanf(fp, "%lf\n", &time);
8         fscanf(fp, "%lld\n", &n_body);
9         return n_body;
10    }
11    void writeAscii(FILE* fp) const {
12        fprintf(fp, "%e\n", time);
13        fprintf(fp, "%lld\n", n_body);
14    }
15 };
16
17 class FPGrav{
18 public:
19     PS::S64 id;
20     PS::F64 mass;
21     PS::F64vec pos;
22     PS::F64vec vel;
23     PS::F64vec acc;
```

FileHeaderクラス

```

16 class FPGrav{
17 public:
18     PS::S64    id;
19     PS::F64    mass;
20     PS::F64vec pos;
21     PS::F64vec vel;
22     PS::F64vec acc;
23     PS::F64    pot;
24
25
26     static PS::F64 eps;
27
28     PS::F64vec getPos() const {
29         return pos;
30     }
31
32     PS::F64 getCharge() const {
33         return mass;
34     }
35
36     void copyFromFP(const FPGrav & fp){
37         mass = fp.mass;
38         pos  = fp.pos;
39     }
40
41     void copyFromForce(const FPGrav & force) {
42         acc = force.acc;
43         pot = force.pot;
44     }

```

粒子クラス

16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

```
class FPGrav{  
public:  
    PS::S64    id;  
    PS::F64    mass;  
    PS::F64vec pos;  
    PS::F64vec vel;  
    PS::F64vec acc;  
    PS::F64    pot;  
  
    static PS::F64 eps;  
  
    PS::F64vec getPos() const {  
        return pos;  
    }  
  
    PS::F64 getCharge() const {  
        return mass;  
    }  
  
    void copyFromFP(const FPGrav & fp){  
        mass = fp.mass;  
        pos  = fp.pos;  
    }  
  
    void copyFromForce(const FPGrav & force) {  
        acc = force.acc;  
        pot = force.pot;  
    }  
};
```

粒子物理量


```
19 PS::S64 id;  
20 PS::F64 mass;  
21 PS::F64vec pos;  
22 PS::F64vec vel;  
23 PS::F64vec acc;  
24 PS::F64 pot;  
25  
26 static PS::F64 eps;
```

```
28 PS::F64vec getPos() const {  
29     return pos;  
30 }
```

```
32 PS::F64 getCharge() const {  
33     return mass;  
34 }
```

```
36 void copyFromFP(const FPGrav & fp){  
37     mass = fp.mass;  
38     pos = fp.pos;  
39 }
```

```
41 void copyFromForce(const FPGrav & force) {  
42     acc = force.acc;  
43     pot = force.pot;  
44 }
```

```
46 void clear() {  
47     acc = 0.0;
```

FDPSにデータを渡すための
メンバ関数

```

44 }
45
46 void clear() {
47     acc = 0.0;
48     pot = 0.0;
49 }
50
51 void writeAscii(FILE* fp) const {
52     fprintf(fp, "%lld\t%g\t%g\t%g\t%g\t%g\t%g\n",
53             this->id, this->mass,
54             this->pos.x, this->pos.y, this->pos.z,
55             this->vel.x, this->vel.y, this->vel.z);
56 }
57
58 void readAscii(FILE* fp) {
59     fscanf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
60            &this->id, &this->mass,
61            &this->pos.x, &this->pos.y, &this->pos.z,
62            &this->vel.x, &this->vel.y, &this->vel.z);
63 }
64
65 };
66
67
68 #ifdef ENABLE_PHANTOM_GRAPE_X86
69
70
71 template <class TParticle>

```

I/Oメンバ関数

相互作用

テンプレート関数

(w/ Phantom-GRAPE)

```
#ifdef ENABLE_PHANTOM_GRAPE_X86
```

```
template <class TParticleJ>
```

```
void CalcGravity(const FPGrav * iptcl,
```

```
    const PS::S32 ni,
```

```
    const TParticleJ * jptcl,
```

```
    const PS::S32 nj,
```

```
    FPGrav * force) {
```

```
    const PS::S32 nipipe = ni;
```

```
    const PS::S32 njpipe = nj;
```

```
    PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
```

```
    PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
```

```
    PS::F64 *pi      = (PS::F64 *)malloc(sizeof(PS::F64) * nipipe);
```

```
    PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpipe * PS::DIMENSION);
```

```
    PS::F64 *mj      = (PS::F64 *)malloc(sizeof(PS::F64) * njpipe);
```

```
    for(PS::S32 i = 0; i < ni; i++) {
```

```
        xi[i][0] = iptcl[i].getPos()[0];
```

```
        xi[i][1] = iptcl[i].getPos()[1];
```

```
        xi[i][2] = iptcl[i].getPos()[2];
```

```
        ai[i][0] = 0.0;
```

```
        ai[i][1] = 0.0;
```

```
        ai[i][2] = 0.0;
```

```

96     xj[j][2] = jptcl[j].getPos()[2];
97     mj[j]    = jptcl[j].getCharge();
98     xj[j][0] = jptcl[j].pos[0];
99     xj[j][1] = jptcl[j].pos[1];
00     xj[j][2] = jptcl[j].pos[2];
01     mj[j]    = jptcl[j].mass;
02 }
03 PS::S32 devid = PS::Comm::getThreadNum();
04 g5_set_xmjMC(devid, 0, nj, xj, mj);
05 g5_set_nMC(devid, nj);
06 g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
07 for(PS::S32 i = 0; i < ni; i++) {
08     force[i].acc[0] += ai[i][0];
09     force[i].acc[1] += ai[i][1];
10     force[i].acc[2] += ai[i][2];
11     force[i].pot    -= pi[i];
12 }
13 free(xi);
14 free(ai);
15 free(pi);
16 free(xj);
17 free(mj);
18 }

```

```

20 #else

```

```

21
22 template <class TParticle>
23 void CalcGravity(const FPGrav * ep_i,
24                 const PS::S32 n_in

```

```

22 template <class TParticle>
23 void CalcGravity(const FPGrav * ep_i,
24                 const PS::S32 n_ip,
25                 const TParticleJ * ep_j,
26                 const PS::S32 n_jp,
27                 FPGrav * force) {
28     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
29     for(PS::S32 i = 0; i < n_ip; i++){
30         PS::F64vec xi = ep_i[i].getPos();
31         PS::F64vec ai = 0.0;
32         PS::F64 poti = 0.0;
33         for(PS::S32 j = 0; j < n_jp; j++){
34             PS::F64vec rij = xi - ep_j[j].getPos();
35             PS::F64 r3_inv = rij * rij + eps2;
36             PS::F64 r_inv = 1.0/sqrt(r3_inv);
37             r3_inv = r_inv * r_inv;
38             r_inv *= ep_j[j].getCharge();
39             r3_inv *= r_inv;
40             ai -= r3_inv * rij;
41             poti -= r_inv;
42         }
43         force[i].acc += ai;
44         force[i].pot += poti;
45     }
46 }
47
48 #endif

```

相互作用

テンプレート関数

(w/o Phantom-GRAPE)

```

22 template <class TParticle>
23 void CalcGravity(const FPGrav * ep_i,
24                 const PS::S32 n_ip,
25                 const TParticleJ * ep_j,
26                 const PS::S32 n_jp,
27                 FPGrav * force) {
28     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
29     for(PS::S32 i = 0; i < n_ip; i++){
30         PS::F64vec xi = ep_i[i].getPos();
31         PS::F64vec ai = 0.0;
32         PS::F64 poti = 0.0;
33         for(PS::S32 j = 0; j < n_jp; j++){
34             PS::F64vec rij = xi - ep_j[j].getPos();
35             PS::F64 r3_inv = rij * rij + eps2;
36             PS::F64 r_inv = 1.0/sqrt(r3_inv);
37             r3_inv = r_inv * r_inv;
38             r_inv *= ep_j[j].getCharge();
39             r3_inv *= r_inv;
40             ai -= r3_inv * rij;
41             poti -= r_inv;
42         }
43         force[i].acc += ai;
44         force[i].pot += poti;
45     }
46 }
47
48 #endif

```

相互作用

テンプレート関数

(w/o Phantom-GRAPE)


```

22 template <class TParticleJ>
23 void CalcGravity(const FPGrav * ep_i,
24                 const PS::S32 n_ip,
25                 const TParticleJ * ep_j,
26                 const PS::S32 n_jp,
27                 FPGrav * force) {
28     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
29     for(PS::S32 i = 0; i < n_ip; i++){
30         PS::F64vec xi = ep_i[i].getPos();
31         PS::F64vec ai = 0.0;
32         PS::F64 poti = 0.0;
33         for(PS::S32 j = 0; j < n_jp; j++){
34             PS::F64vec rij = xi - ep_j[j].getPos();
35             PS::F64 r3_inv = rij * rij + eps2;
36             PS::F64 r_inv = 1.0/sqrt(r3_inv);
37             r3_inv = r_inv * r_inv;
38             r_inv *= ep_j[j].getCharge();
39             r3_inv *= r_inv;
40             ai -= r3_inv * rij;
41             poti -= r_inv;
42         }
43         force[i].acc += ai;
44         force[i].pot += poti;
45     }
46 }
47
48 #endif

```

相互作用
テンプレート関数
(w/o Phantom-GRAPE)

相互作用template関数

- 本当は2つ(Super Particleからの寄与と通常Particleからの寄与)書く必要がある。
- 今回はどちらも相互作用形としては同じなので、templateを使って1つで済みます。

nbody.cpp

55 lines (520 SLOC) | 11.4 KB

```
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);
```

nbody.cpp

55 lines (520 SLOC) | 11.4 KB

```
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(mass > 0.0);
```

FDPSのinclude

```
delete [] mass;  
79 delete [] pos;  
80 delete [] vel;  
81 }
```

```
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 }
```

```
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 }
```

```
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){
```

leap-frog法による
時間積分

```
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){
```

getNumberOfParticleLocal()

で粒子数が取得できる

```
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;
```

```
    }
```

```
}
```

i粒子のデータが取得できる

```
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){
```

```
163 PS::F64 FPGrav::eps = 1.0/32.0;
164
165 int main(int argc, char *argv[]) {
```

メイン関数開始

```
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':
```



```

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':

```

FDPS初期化

コマンドライン

引数の解析

```
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':
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;

```

粒子群クラスの 生成・初期化

```
246 PS::ParticleSystem<FPGrav> system_grav;
247 system_grav.initialize();
248
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.collectSampleParticle(system_grav);
261 dinfo.decomposeDomain();
262 system_grav.exchangeParticle(dinfo);
263 n_loc = system_grav.getNumberOfParticleLocal();
264
265 #ifdef ENABLE_PHANTOM_GRAPE_X86
266     g5_open();
267     g5_set_eps_to_all(FPGrav::eps);
268 #endif
269
270 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
271 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
272 #ifdef MULTI_WALK
273     const PS::S32 n_walk_limit = 200;
274     const PS::S32 tag_max = 1;
```

```

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.collectSampleParticle(system_grav);
261 dinfo.decomposeDomain();
262 system_grav.exchangeParticle(dinfo);
263 n_loc = system_grav.getNumberOfParticleLocal();
264
265 #ifdef ENABLE_PHANTOM_GRAPE_X86
266     g5_open();
267     g5_set_eps_to_all(FPGrav::eps);
268 #endif
269
270 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
271 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
272 #ifdef MULTI_WALK
273     const PS::S32 n_walk_limit = 200;
274     const PS::S32 tag_max = 1;

```

ドメイン情報の
生成・初期化
領域分割

```

268 #endif
269
270 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
271 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
272 #ifdef MULTI_WALK
273     const PS::S32 n_walk_limit = 200;
274     const PS::S32 tag_max = 1;
275     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
276                                                  RetrieveKernel,
277                                                  tag_max,
278                                                  system_grav,
279                                                  dinfo,
280                                                  n_walk_limit);
281 #else
282     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
283                                       CalcGravity<PS::SPJMonopole>,
284                                       system_grav,
285                                       dinfo);
286 #endif
287 PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
288 calcEnergy(system_grav, Etot0, Ekin0, Epot0);
289 PS::F64 time_diag = 0.0;
290 PS::F64 time_snap = 0.0;
291 PS::S64 n_loop = 0;
292 PS::S32 id_snap = 0;
293 while(time_sys < time_end){
294     if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) )
295         char filename[256];

```

相互作用ツリークラスの
生成・初期化


```

268 #endif
269
270 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
271 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
272 #ifdef MULTI_WALK
273     const PS::S32 n_walk_limit = 200;
274     const PS::S32 tag_max = 1;
275     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
276                                                  RetrieveKernel,
277                                                  tag_max,
278                                                  system_grav,
279                                                  dinfo,
280                                                  n_walk_limit);
281 #else
282     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
283                                       CalcGravity<PS::SPJMonopole>,
284                                       system_grav,
285                                       dinfo);
286 #endif
287 PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
288 calcEnergy(system_grav, Etot0, Ekin0, Epot0);
289 PS::F64 time_diag = 0.0;
290 PS::F64 time_snap = 0.0;
291 PS::S64 n_loop = 0;
292 PS::S32 id_snap = 0;
293 while(time_sys < time_end){
294     if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) )
295         char filename[256];

```

相互作用関数を用いた 力の計算

時間積分

```
289 PS::F64 time_diag = 0.0;
290 PS::F64 time_snap = 0.0;
291 PS::S64 n_loop = 0;
292 PS::S32 id_snap = 0;
293 while(time_sys < time_end){
294     if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) )
295         char filename[256];
296         sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
297         FileHeader header;
298         header.time = time_sys;
299         header.n_body = system_grav.getNumberOfParticleGlobal();
300         system_grav.writeParticleAscii(filename, header);
301         time_snap += dt_snap;
302     }
303
304     calcEnergy(system_grav, Etot1, Ekin1, Epot1);
305
306     if(PS::Comm::getRank() == 0){
307         if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys) )
308             fout_eng << time_sys << " " << (Etot1 - Etot0) / Etot0 << std::endl;
309             fprintf(stderr, "time: %10.7f energy error: %+e\n",
310                     time_sys, (Etot1 - Etot0) / Etot0);
311             time_diag += dt_diag;
312         }
313     }
314
315     kick(system_grav, dt * 0.5);
```

```
316 kick(system_grav, dt * 0.5);
317
318 time_sys += dt;
319 drift(system_grav, dt);
320
321 if(n_loop % 4 == 0){
322     dinfo.decomposeDomainAll(system_grav);
323 }
324
325 system_grav.exchangeParticle(dinfo);
326 #ifdef MULTI_WALK
327     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
328                                                    RetrieveKernel,
329                                                    tag_max,
330                                                    system_grav,
331                                                    dinfo,
332                                                    n_walk_limit,
333                                                    true);
334 #else
335     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
336                                        CalcGravity<PS::SPJMonopole>,
337                                        system_grav,
338                                        dinfo);
339 #endif
340
341 kick(system_grav, dt * 0.5);
342
343 n_loop++;
```

```

326     system_grav.exchangeParticle(dinfo);
327     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
328                                                    RetrieveKernel,
329                                                    tag_max,
330                                                    system_grav,
331                                                    dinfo,
332                                                    n_walk_limit,
333                                                    true);
334 #else
335     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
336                                        CalcGravity<PS::SPJMonopole>,
337                                        system_grav,
338                                        dinfo);
339 #endif
340
341     kick(system_grav, dt * 0.5);
342
343     n_loop++;
344 }
345
346 #ifdef ENABLE_PHANTOM_GRAPE_X86
347     g5_close();
348 #endif
349
350     PS::Finalize();
351     return 0;
352 }

```

FDPS終了

```
system_grav.exchangerArticle(dinfo);
326 #ifdef MULTI_WALK
327     tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
328                                                  RetrieveKernel,
329                                                  tag_max,
330                                                  system_grav,
331                                                  dinfo,
332                                                  n_walk_limit,
333                                                  true);
334 #else
335     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
336                                       CalcGravity<PS::SPJMonopole>,
337                                       system_grav,
338                                       dinfo);
339 #endif
340
341     kick(system_grav, dt * 0.5);
342
343     n_loop++;
344 }
345
346 #ifdef ENABLE_PHANTOM_GRAPE_X86
347     g5_close();
348 #endif
349
350     PS::Finalize();
351     return 0;
```

352 352行！

最後に

- ユーザーが書かなければならないのは大体これくらい。
→重力の場合は352+148行で終わる。
- コード内に並列化を意識するようなところは無かった。
→コンパイルの方法を切り替えるだけで、
OpenMPやMPIを切り替えられる。

実習の流れ

- 詳しくはFDPS講習会の手引を御覧ください。
(<http://www.jmlab.jp/?p=650>)
- 実習用のFOCUSスパコンにログインし、サンプルコードを
(1)並列化無し (2)OpenMP (3)OpenMP + MPI
の3パターンについてコンパイル・実行
【計算内容】 重力
cold collapse
流体 (Smoothed Particle Hydrodynamics法)
adiabatic sphere collapse
その後結果の解析