user−defined.hpp

```
 1  class FileHeader{
 2  public:
 3      PS::S64 n_body;
 4      PS::F64 time;
 5      PS::S32 readAscii(FILE * fp) {
 6            fscanf(fp, "%lf\n", &time);
 7            fscanf(fp, "%lld\n", &n_body);
 8            return n_body;
 9      }
10      void writeAscii(FILE* fp) const {
11    fprintf(fp, "%e\n", time);
12    fprintf(fp, "%lld\n", n_body);
13      }
14  };
15
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      static PS::F64 eps;
26
27      PS::F64vec getPos() const {
28          return pos;
29      }
30
31      PS::F64 getCharge() const {
32          return mass;
33      }
34
35      void copyFromFP(const FPGrav & fp){
36          mass = fp.mass;
37          pos  = fp.pos;
38      }
39
40      void copyFromForce(const FPGrav & force) {
41          acc = force.acc;
42          pot = force.pot;
43      }
44
45      void clear() {
46          acc = 0.0;
47          pot = 0.0;
48      }
49
50    void writeAscii(FILE* fp) const {
51            fprintf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
52                  this->id, this->mass,
53                  this->pos.x, this->pos.y, this->pos.z,
54                  this->vel.x, this->vel.y, this->vel.z);
55    }
56
57    void readAscii(FILE* fp) {
58            fscanf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
59                 &this->id, &this->mass,
60                 &this->pos.x, &this->pos.y, &this->pos.z,
61                 &this->vel.x, &this->vel.y, &this->vel.z);
62    }
63
64  };
65
66  PS::F64 FPGrav::eps = 1.0/32.0;
67
```

user−defined.hpp

```
 68  #ifdef ENABLE_PHANTOM_GRAPE_X86
 69
 70
 71  template <class TParticleJ>
 72  void CalcGravity(const FPGrav * iptcl,
 73                   const PS::S32 ni,
 74                   const TParticleJ * jptcl,
 75                   const PS::S32 nj,
 76                   FPGrav * force) {
 77      const PS::S32 nipipe = ni;
 78      const PS::S32 njpipe = nj;
 79      PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipi
pe * PS::DIMENSION);
 80      PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipi
pe * PS::DIMENSION);
 81      PS::F64  *pi     = (PS::F64  *    )malloc(sizeof(PS::F64) * nipi
pe);
 82      PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpi
pe * PS::DIMENSION);
 83      PS::F64  *mj     = (PS::F64  *    )malloc(sizeof(PS::F64) * njpi
pe);
 84      for(PS::S32 i = 0; i < ni; i++) {
 85          xi[i][0] = iptcl[i].getPos()[0];
 86          xi[i][1] = iptcl[i].getPos()[1];
 87          xi[i][2] = iptcl[i].getPos()[2];
 88          ai[i][0] = 0.0;
 89          ai[i][1] = 0.0;
 90          ai[i][2] = 0.0;
 91          pi[i]    = 0.0;
 92      }
 93      for(PS::S32 j = 0; j < nj; j++) {
 94          xj[j][0] = jptcl[j].getPos()[0];
 95          xj[j][1] = jptcl[j].getPos()[1];
 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];
100          xj[j][2] = jptcl[j].pos[2];
101          mj[j]    = jptcl[j].mass;
102      }
103  #ifdef PARTICLE_SIMULATOR_THREAD_PARALLEL
104      PS::S32 devid = omp_get_thread_num();
105  #else
106      PS::S32 devid = 0;
107  #endif
108      g5_set_xmjMC(devid, 0, nj, xj, mj);
109      g5_set_nMC(devid, nj);
110      g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
111      for(PS::S32 i = 0; i < ni; i++) {
112    force[i].acc[0] += ai[i][0];
113          force[i].acc[1] += ai[i][1];
114          force[i].acc[2] += ai[i][2];
115          force[i].pot    −= pi[i];
116      }
117      free(xi);
118      free(ai);
119      free(pi);
120      free(xj);
121      free(mj);
122  }
123
124  #else
125
126  template <class TParticleJ>
127  void CalcGravity(const FPGrav * ep_i,
128                   const PS::S32 n_ip,
129                   const TParticleJ * ep_j,
```

user−defined.hpp

```
130                     const PS::S32 n_jp,
131                     FPGrav * force) {
132       PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
133       for(PS::S32 i = 0; i < n_ip; i++){
134           PS::F64vec xi = ep_i[i].getPos();
135           PS::F64vec ai = 0.0;
136           PS::F64 poti = 0.0;
137           for(PS::S32 j = 0; j < n_jp; j++){
138               PS::F64vec rij    = xi − ep_j[j].getPos();
139               PS::F64    r3_inv = rij * rij + eps2;
140               PS::F64    r_inv  = 1.0/sqrt(r3_inv);
141               r3_inv  = r_inv * r_inv;
142               r_inv  *= ep_j[j].getCharge();
143               r3_inv *= r_inv;
144               ai     −= r3_inv * rij;
145               poti   −= r_inv;
146           }
147           force[i].acc += ai;
148           force[i].pot += poti;
149       }
150   }
151
152   #endif
```