

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;
24     PS::F64 pot;
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     }
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\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\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 #ifdef ENABLE_PHANTOM_GRAPE_X86
68
69
70
71 template <class TParticleJ>
72 void CalcGravity(const FPGrav * iptcl,

```

user-defined.hpp

```

73         const PS::S32 ni,
74         const TParticleJ * jptcl,
75         const PS::S32 nj,
76         FPGrav * force) {
77     const PS::S32 npipe = ni;
78     const PS::S32 njpipe = nj;
79     PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * npipe * PS
::DIMENSION);
80     PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * npipe * PS
::DIMENSION);
81     PS::F64 *pi = (PS::F64 *)malloc(sizeof(PS::F64) * npipe);
82     PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpipe * PS
::DIMENSION);
83     PS::F64 *mj = (PS::F64 *)malloc(sizeof(PS::F64) * njpipe);
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    PS::S32 devid = PS::Comm::getThreadNum();
104    g5_set_xmJMC(devid, 0, nj, xj, mj);
105    g5_set_nMC(devid, nj);
106    g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
107    for(PS::S32 i = 0; i < ni; i++) {
108        force[i].acc[0] += ai[i][0];
109        force[i].acc[1] += ai[i][1];
110        force[i].acc[2] += ai[i][2];
111        force[i].pot -= pi[i];
112    }
113    free(xi);
114    free(ai);
115    free(pi);
116    free(xj);
117    free(mj);
118 }
119
120 #else
121
122 template <class TParticleJ>
123 void CalcGravity(const FPGrav * ep_i,
124     const PS::S32 n_ip,
125     const TParticleJ * ep_j,
126     const PS::S32 n_jp,
127     FPGrav * force) {
128     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
129     for(PS::S32 i = 0; i < n_ip; i++){
130         PS::F64vec xi = ep_i[i].getPos();
131         PS::F64vec ai = 0.0;
132         PS::F64 poti = 0.0;
133         for(PS::S32 j = 0; j < n_jp; j++){
134             PS::F64vec rij = xi - ep_j[j].getPos();
135             PS::F64 r3_inv = rij * rij + eps2;
136             PS::F64 r_inv = 1.0/sqrt(r3_inv);
137             r3_inv = r_inv * r_inv;
138             r_inv *= ep_j[j].getCharge();
139             r3_inv *= r_inv;
140             ai -= r3_inv * rij;
141             poti -= r_inv;

```

user-defined.hpp

```
142     }  
143     force[i].acc += ai;  
144     force[i].pot += poti;  
145   }  
146 }  
147  
148 #endif
```