user−defined.hpp

```
#pragma once
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};

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;
    }

    void clear() {
        acc = 0.0;
        pot = 0.0;
    }

    void writeAscii(FILE* fp) const {
        fprintf(fp, "%lld\t%g\t%g\t%g\t%g\t%g\t%g\t%g\n",
                this->id, this->mass,
                this->pos.x, this->pos.y, this->pos.z,
                this->vel.x, this->vel.y, this->vel.z);
    }

    void readAscii(FILE* fp) {
        fscanf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
               &this->id, &this->mass,
               &this->pos.x, &this->pos.y, &this->pos.z,
               &this->vel.x, &this->vel.y, &this->vel.z);
        }

};


#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;
        pi[i]    = 0.0;
    }
    for(PS::S32 j = 0; j < nj; j++) {
        xj[j][0] = jptcl[j].getPos()[0];
        xj[j][1] = jptcl[j].getPos()[1];
        xj[j][2] = jptcl[j].getPos()[2];
        mj[j]    = jptcl[j].getCharge();
        xj[j][0] = jptcl[j].pos[0];
        xj[j][1] = jptcl[j].pos[1];
        xj[j][2] = jptcl[j].pos[2];
        mj[j]    = jptcl[j].mass;
    }
    PS::S32 devid = PS::Comm::getThreadNum();
    g5_set_xmjMC(devid, 0, nj, xj, mj);
    g5_set_nMC(devid, nj);
    g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
    for(PS::S32 i = 0; i < ni; i++) {
        force[i].acc[0] += ai[i][0];
        force[i].acc[1] += ai[i][1];
        force[i].acc[2] += ai[i][2];
        force[i].pot    -= pi[i];
    }
    free(xi);
    free(ai);
    free(pi);
    free(xj);
    free(mj);
}

#else

template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParticleJ * ep_j,
                 const PS::S32 n_jp,
                 FPGrav * force) {
    PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
    for(PS::S32 i = 0; i < n_ip; i++){
        PS::F64vec xi = ep_i[i].getPos();
        PS::F64vec ai = 0.0;
        PS::F64 poti = 0.0;
        for(PS::S32 j = 0; j < n_jp; j++){
            PS::F64vec rij    = xi − ep_j[j].getPos();
            PS::F64    r3_inv = rij * rij + eps2;
            PS::F64    r_inv  = 1.0/sqrt(r3_inv);
            r3_inv  = r_inv * r_inv;
            r_inv  *= ep_j[j].getCharge();
            r3_inv *= r_inv;
            ai     −= r3_inv * rij;
            poti   −= r_inv;
```

user−defined.hpp

```
        }
        force[i].acc += ai;
        force[i].pot += poti;
    }
}

#endif
```