```
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;
    PS::F64  xi[3], ai[3];
    PS::F64  *mj     = (PS::F64  *    )malloc(sizeof(PS::F64) *
     ↪    n_jp);
    PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * n_jp
     ↪    * 3);
    for (PS::S32 j=0; j<n_jp; j++) {
        mj[j]    = ep_j[j].getCharge();
        xj[j][0] = ep_j[j].getPos()[0];
        xj[j][1] = ep_j[j].getPos()[1];
        xj[j][2] = ep_j[j].getPos()[2];
    }
    for(PS::S32 i = 0; i < n_ip; i++){
        xi[0] = ep_i[i].getPos()[0];
        xi[1] = ep_i[i].getPos()[1];
        xi[2] = ep_i[i].getPos()[2];
        ai[0] = 0.0;
        ai[1] = 0.0;
        ai[2] = 0.0;
        PS::F64 poti = 0.0;
        for(PS::S32 j = 0; j < n_jp; j++){
            PS::F64 rij[3];
            rij[0] = xi[0] - xj[j][0];
            rij[1] = xi[1] - xj[j][1];
            rij[2] = xi[2] - xj[j][2];
            PS::F64 r3_inv = (rij[0] * rij[0]
                              +rij[1] * rij[1]
                              +rij[2] * rij[2])
                           + eps2;
            PS::F64 r_inv  = 1.0/sqrt(r3_inv);
            r3_inv  = r_inv * r_inv;
            r_inv  *= mj[j];
            r3_inv *= r_inv;
            ai[0]  -= r3_inv * rij[0];
            ai[1]  -= r3_inv * rij[1];
            ai[2]  -= r3_inv * rij[2];
            poti   -= r_inv;
        }
        force[i].acc.x += ai[0];
        force[i].acc.y += ai[1];
        force[i].acc.z += ai[2];
        force[i].pot += poti;
    }
    free(mj);
    free(xj);
}
```