# FDPS講習会
# サンプルコード解説(C++)

野村昴太郎 / Kentaro Nomura

理研RCCS リサーチアソシエイト

2018年8月3日

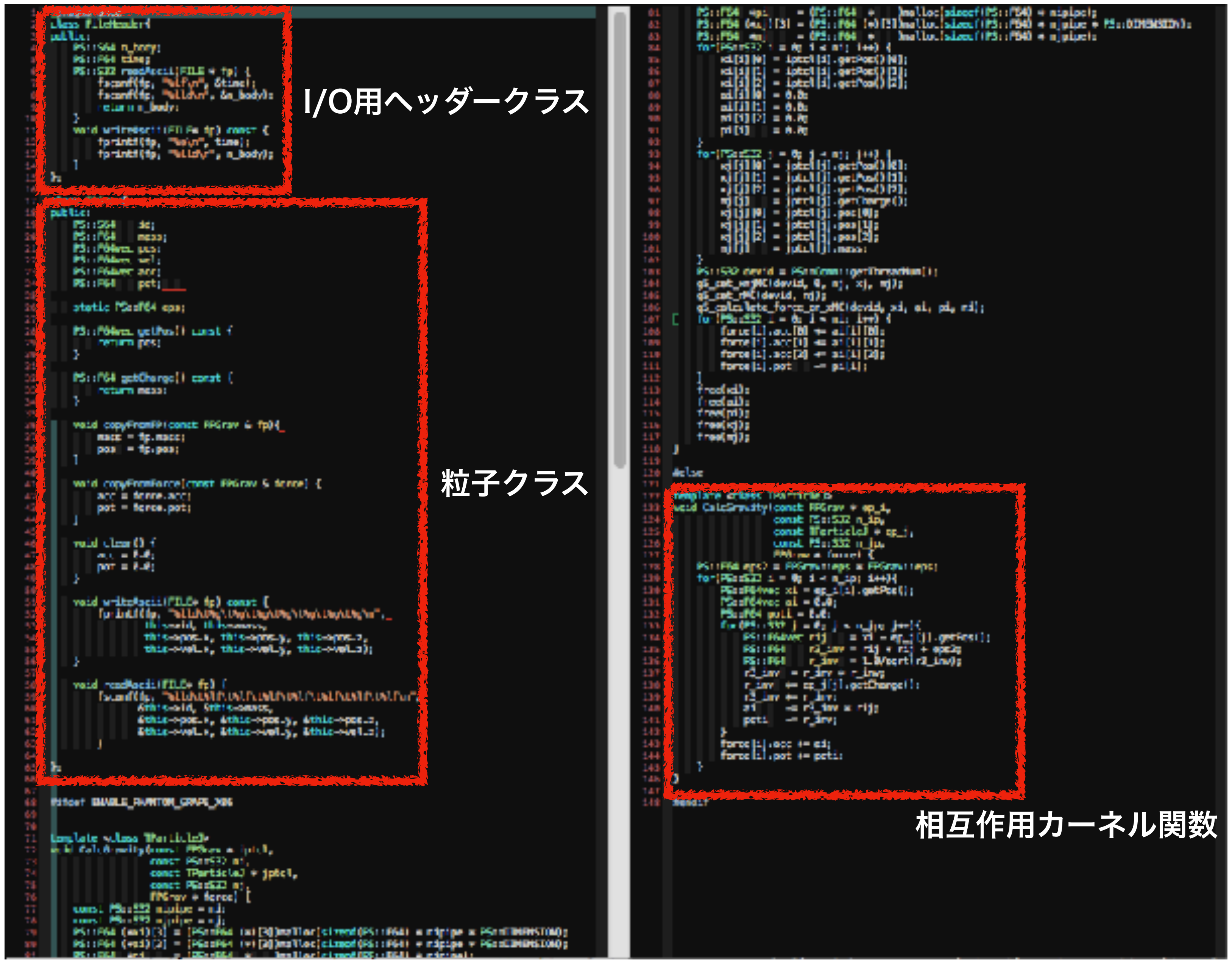
# コードの内容

- ヘッダーのインクルード(#include <particle_simulator.h>)
- 粒子クラスとそのメンバ関数の定義
- 相互作用カーネルの定義
- 時間積分
- I/Oと解析

# 紹介するサンプルコード

- ディレクトリは$FDPS_DIR/sample/c++/nbody
- ファイルはnbody.cppとuser-defined.hpp
- 重力多体問題コード
    - 時間積分はleap-frog法
    - 初期条件はファイル読み込みではなく，その場で生成

# user-defined.hpp全景

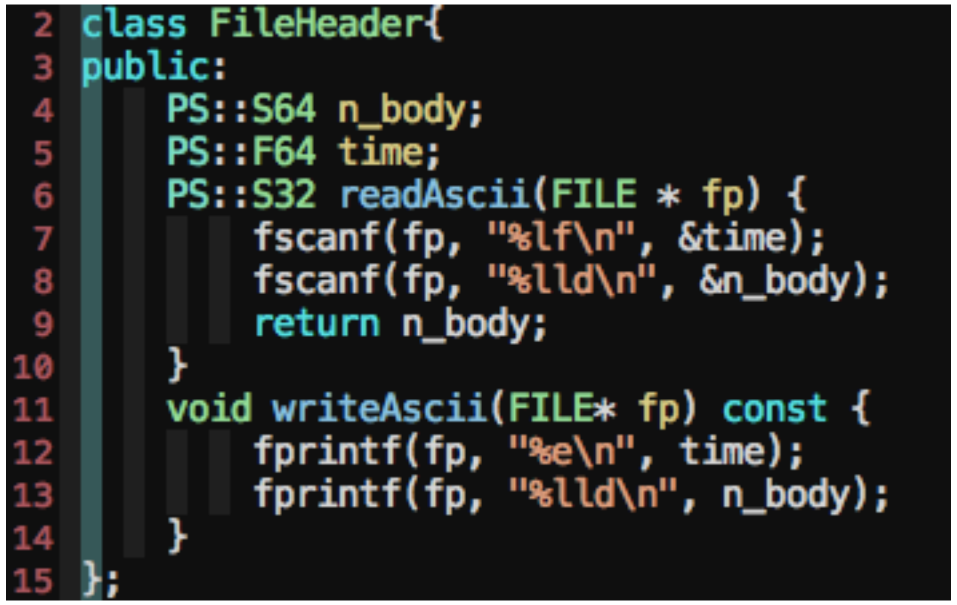


```
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n",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

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 *mj      = (PS::F64 *    )malloc(sizeof(PS::F64) * njpipe);
    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;
        }
        force[i].acc += ai;
        force[i].pot += poti;
    }
}
#endif
```

148行

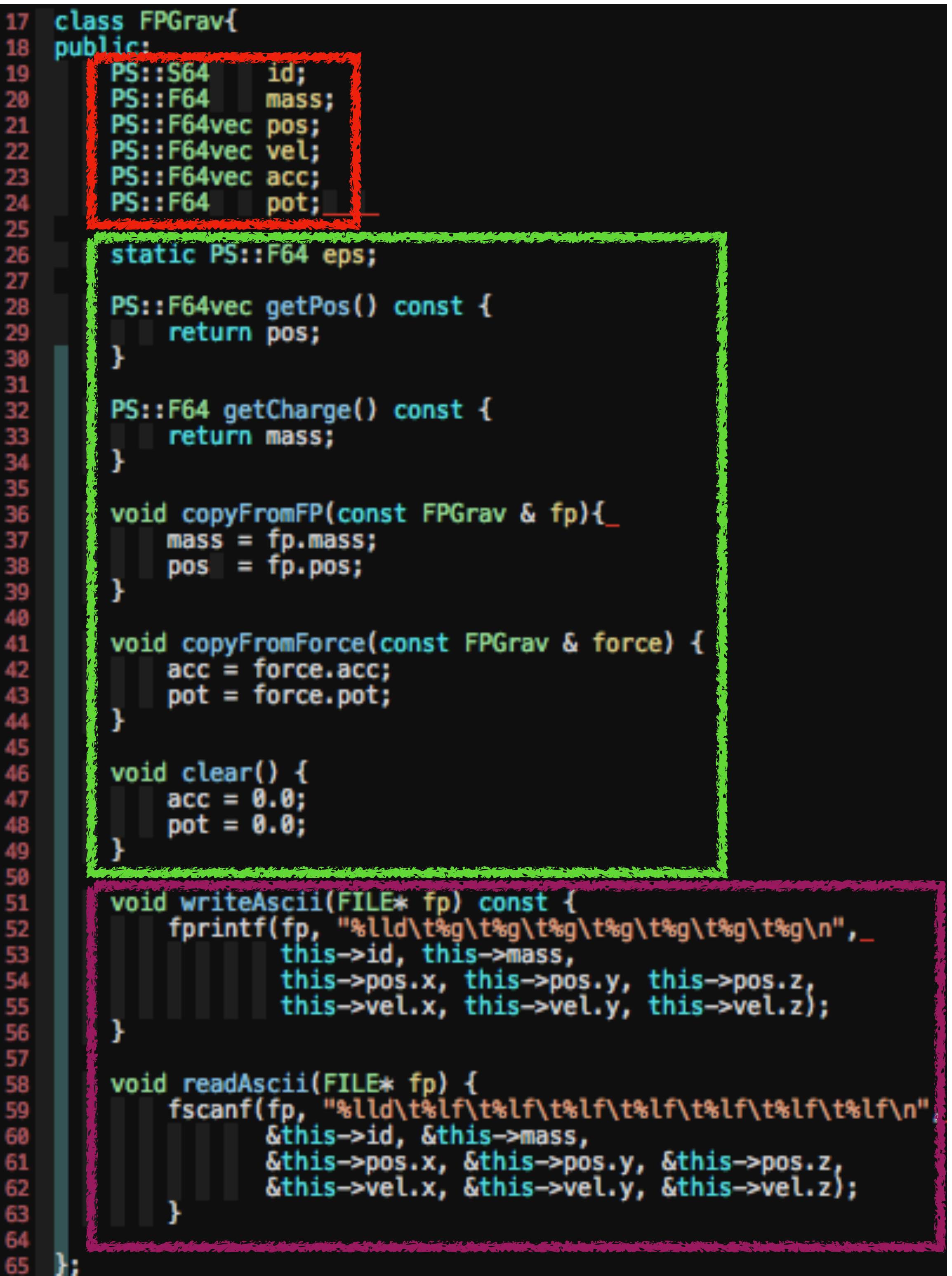
# I/O用ヘッダークラス (FileHeaderクラス)



```
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);
    }
};
```

- ParticleSystem::writePraticleAscii/Binaryで必要
- 粒子データを単一/分散ファイルに書き込む
- readAsciiとwriteAsciiメンバ関数が必要

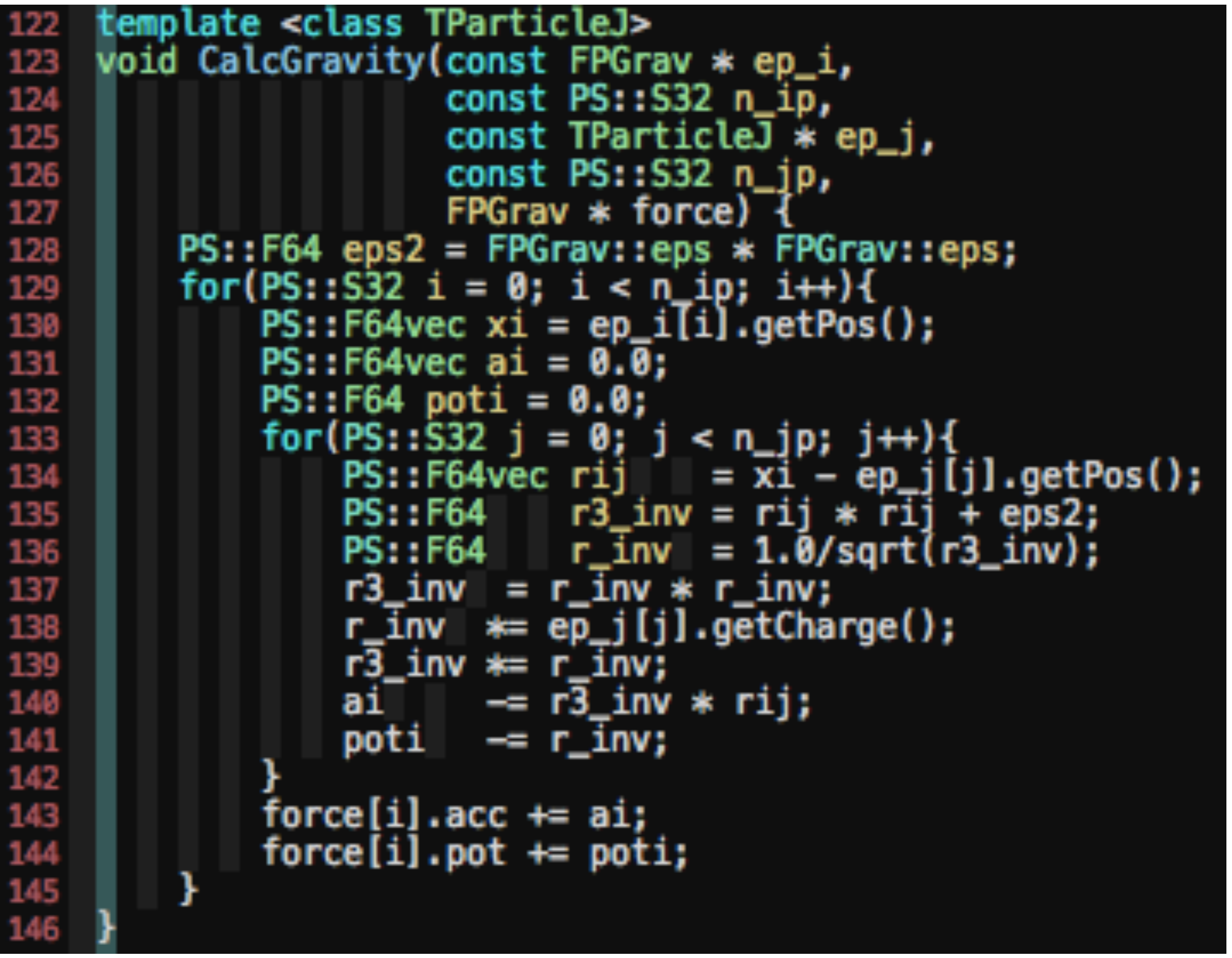
# 粒子クラス (FPGrav)



```
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);
                }
};
```

- メンバ変数に粒子の持つ物理量を記述
- FDPSが粒子の情報を得るのに必要なメンバ関数を定義(getPos, copyFromFP, copyFromForce, clearは必須)
- I/O用にreadAsciiとwriteAsciiメンバ関数を定義
- 必要に応じてEssentialParticle(EP)クラスやForceクラスを作ることが可能．本サンプルではFPGravが全てを兼ねている

# 相互作用カーネル関数



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

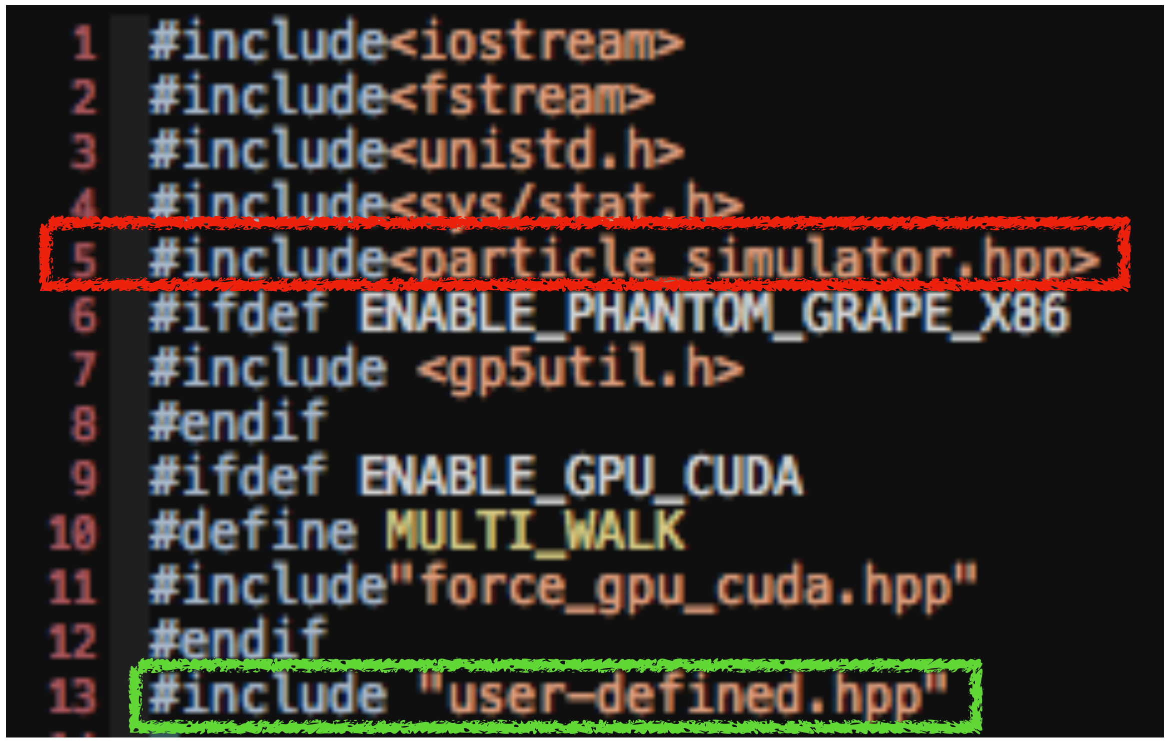
- 本来は普通の粒子(EP)と超粒子(SP)用にそれぞれカーネル関数が必要だが，本サンプルではテンプレート関数化することで同じ関数を利用
- 引数は固定(EPI*, $N_{EPI}$,EPJ*/SPJ*, $N_{EPJ/SPJ}$, Force*)
- getPos()メンバ関数で粒子の座標を取得
- Force(ここではFPGrav)の配列に計算結果を格納

# nbody.cpp全景

```
#include<iostream>
#include<fstream>
[illegible]
#include<particle_simulator.hpp>
[illegible]
#include <unistd.h>
#endif
[illegible]
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"

void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {

    assert(eng < 0.0);
    {
        PS::MTTS mt;
        mt.init_genrand(0);
        for(PS::S32 i = 0; i < n_loc; i++){
            mass[i] = mass_glb / n_glb;
            const PS::F64 radius = 3.0;
            do {
                pos[i][0] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][1] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][2] = (2. * mt.genrand_res53() - 1.) * radius;
            }while(pos[i] * pos[i] >= radius * radius);
            vel[i][0] = 0.0;
            vel[i][1] = 0.0;
            vel[i][2] = 0.0;
[illegible]
```

348行

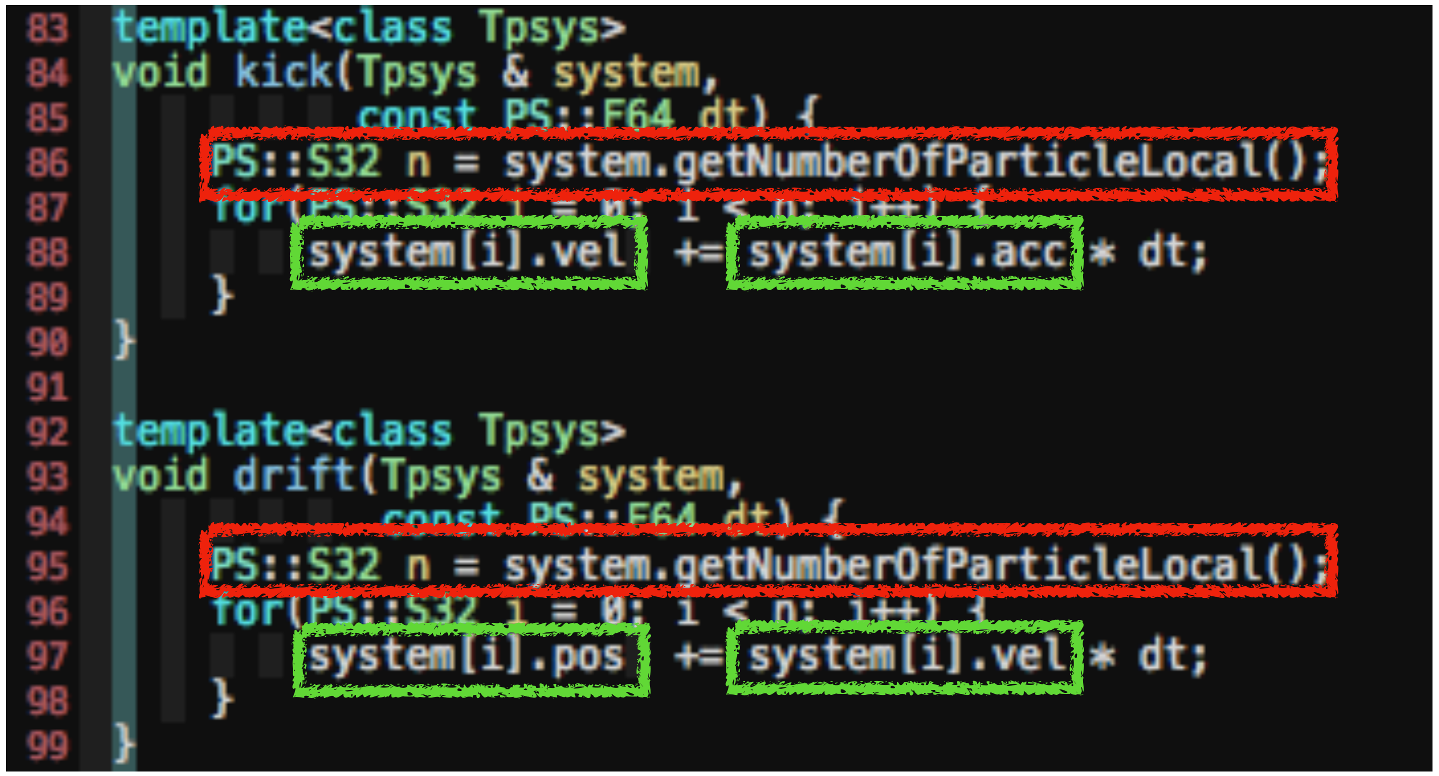
# FDPSヘッダーファイルのインクルード



```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"
```

- particle_simulator.hppをインクルード
- 粒子クラス等の定義が別ファイル(user-defined.hpp)の場合はインクルード

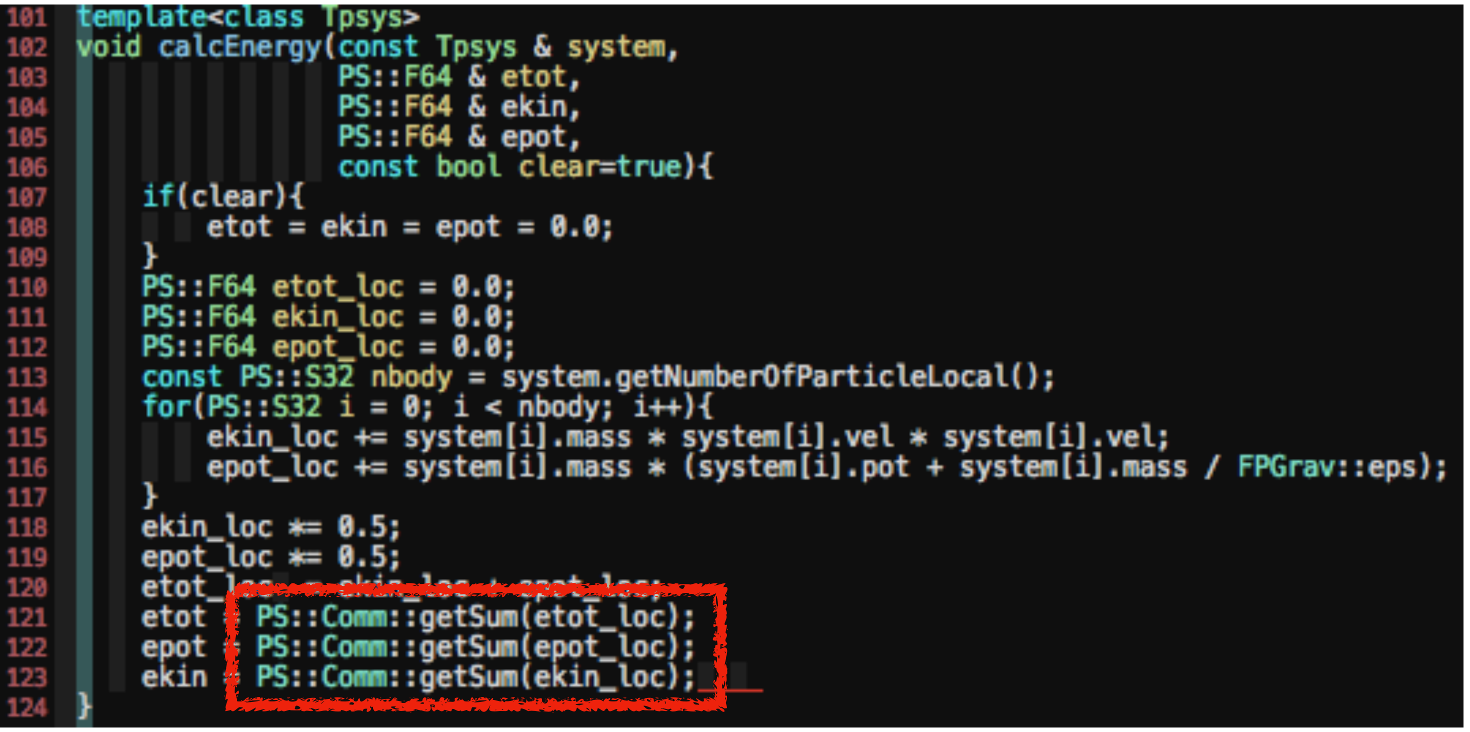
# 時間積分関数の定義



```
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;
    }
}

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;
    }
}
```

- Leap-frog法の速度と座標の更新の関数を定義
- ParticleSystem::getNumberOfParticleLocal()関数でプロセスが担当する粒子の数を取得
- []演算子で粒子データにアクセス

# エネルギーの計算(総和など)



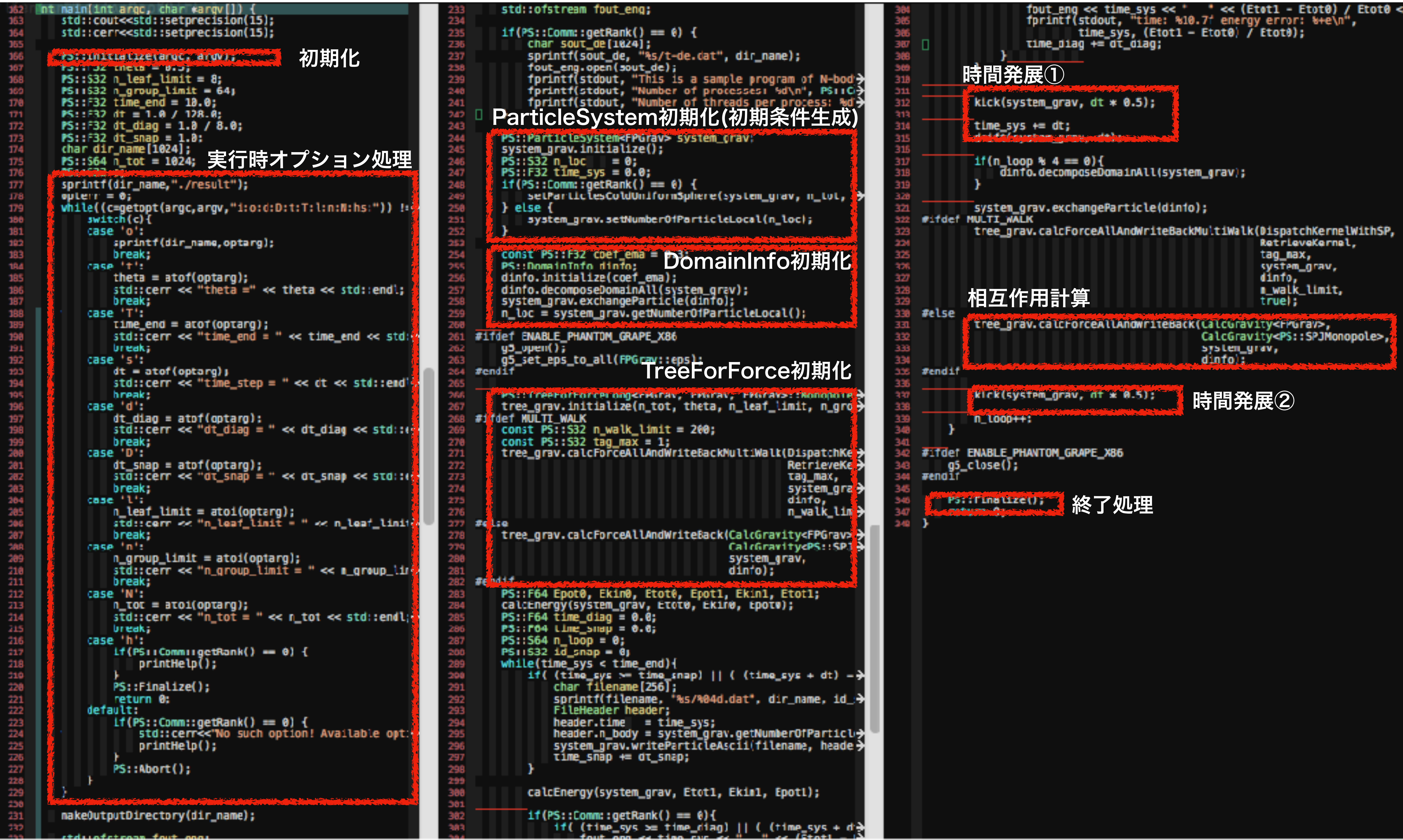
```
template<class Tpsys>
void calcEnergy(const Tpsys & system,
                PS::F64 & etot,
                PS::F64 & ekin,
                PS::F64 & epot,
                const bool clear=true){
    if(clear){
        etot = ekin = epot = 0.0;
    }
    PS::F64 etot_loc = 0.0;
    PS::F64 ekin_loc = 0.0;
    PS::F64 epot_loc = 0.0;
    const PS::S32 nbody = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < nbody; i++){
        ekin_loc += system[i].mass * system[i].vel * system[i].vel;
        epot_loc += system[i].mass * (system[i].pot + system[i].mass / FPGrav::eps);
    }
    ekin_loc *= 0.5;
    epot_loc *= 0.5;
    etot_loc = ekin_loc + epot_loc;
    etot = PS::Comm::getSum(etot_loc);
    epot = PS::Comm::getSum(epot_loc);
    ekin = PS::Comm::getSum(ekin_loc);
}
```

- ポテンシャルや運動エネルギーを計算する場合，プロセス間の総和が必要

- PS::Comm::getSum()関数でプロセス間の総和をとることが可能

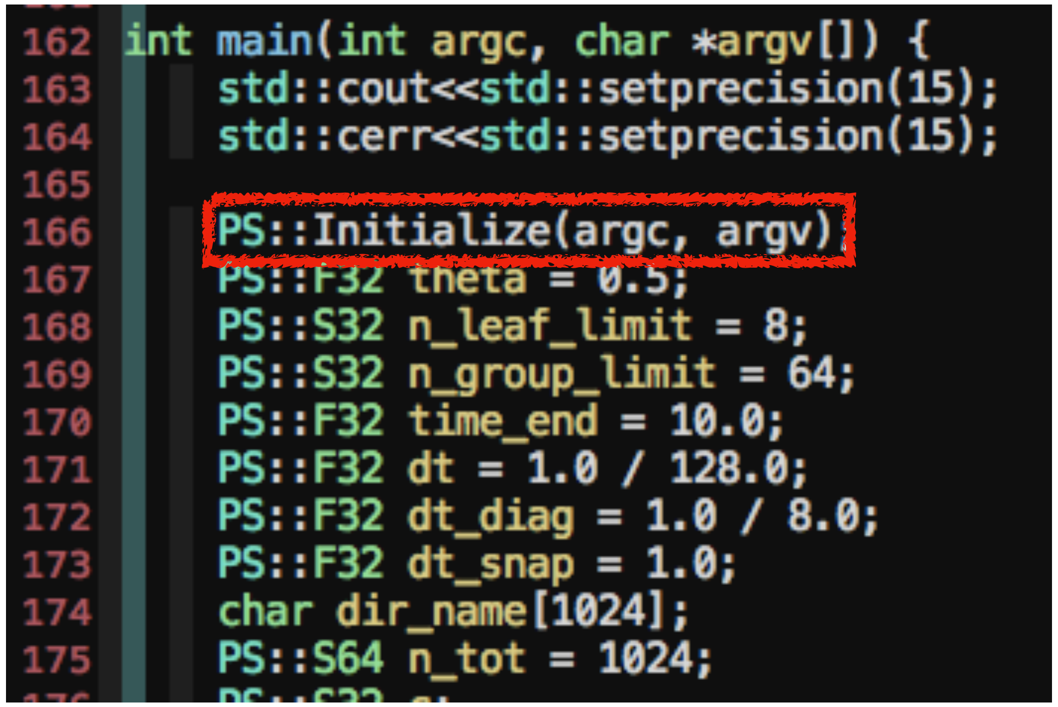
- MPIが使われない場合でもコードの変更が不要

# main関数全景

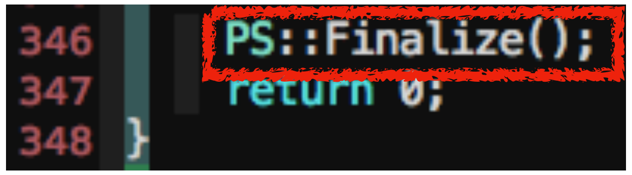


```
int main(int argc, char *argv[]) {
    std::cout<<std::setprecision(15);
    std::cerr<<std::setprecision(15);

    PS::Initialize(argc, argv);
    PS::F32 theta = 0.5;
    PS::S32 n_leaf_limit = 8;
    PS::S32 n_group_limit = 64;
    PS::F32 time_end = 10.0;
    PS::F32 dt = 1.0 / 128.0;
    PS::F32 dt_diag = 1.0 / 8.0;
    PS::F32 dt_snap = 1.0;
    char dir_name[1024];
    PS::S64 n_tot = 1024;
    [illegible]
    sprintf(dir_name,"./result");
    opterr = 0;
    while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) !=
        switch(c){
        case 'o':
            sprintf(dir_name,optarg);
            break;
        case 't':
            theta = atof(optarg);
            std::cerr << "theta =" << theta << std::endl;
            break;
        case 'T':
            time_end = atof(optarg);
            std::cerr << "time_end = " << time_end << std:
            break;
        case 's':
            dt = atof(optarg);
            std::cerr << "time_step = " << dt << std::endl
            break;
        case 'd':
            dt_diag = atof(optarg);
            std::cerr << "dt_diag = " << dt_diag << std::e
            break;
        case 'D':
            dt_snap = atof(optarg);
            std::cerr << "dt_snap = " << dt_snap << std::e
            break;
        case 'l':
            n_leaf_limit = atoi(optarg);
            std::cerr << "n_leaf_limit = " << n_leaf_limit
            break;
        case 'n':
            n_group_limit = atoi(optarg);
            std::cerr << "n_group_limit = " << n_group_lir
            break;
        case 'N':
            n_tot = atoi(optarg);
            std::cerr << "n_tot = " << n_tot << std::endl;
            break;
        case 'h':
            if(PS::Comm::getRank() == 0) {
                printHelp();
            }
            PS::Finalize();
            return 0;
        default:
            if(PS::Comm::getRank() == 0) {
                std::cerr<<"No such option! Available opt:
                printHelp();
            }
            PS::Abort();
        }
    }

    makeOutputDirectory(dir_name);

    std::ofstream fout_eng;

    if(PS::Comm::getRank() == 0) {
        char sout_de[1024];
        sprintf(sout_de, "%s/t-de.dat", dir_name);
        fout_eng.open(sout_de);
        fprintf(stdout, "This is a sample program of N-bod
        fprintf(stdout, "Number of processes: %d\n", PS::C
        fprintf(stdout, "Number of threads per process: %d
    }

    PS::ParticleSystem<FPGrav> system_grav;
    system_grav.initialize();
    PS::S32 n_loc    = 0;
    PS::F32 time_sys = 0.0;
    if(PS::Comm::getRank() == 0) {
        setParticlesColdUniformSphere(system_grav, n_tot,
    } else {
        system_grav.setNumberOfParticleLocal(n_loc);
    }

    const PS::F32 coef_ema = 0.3;
    PS::DomainInfo dinfo;
    dinfo.initialize(coef_ema);
    dinfo.decomposeDomainAll(system_grav);
    system_grav.exchangeParticle(dinfo);
    n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif

    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_gro
#ifdef MULTI_WALK
    const PS::S32 n_walk_limit = 200;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKe
                                                RetrieveKe
                                                tag_max,
                                                system_gra
                                                dinfo,
                                                n_walk_lin
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>
                                       CalcGravity<PS::SPJ
                                       system_grav,
                                       dinfo);
#endif
    PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
    calcEnergy(system_grav, Etot0, Ekin0, Epot0);
    PS::F64 time_diag = 0.0;
    PS::F64 time_snap = 0.0;
    PS::S64 n_loop = 0;
    PS::S32 id_snap = 0;
    while(time_sys < time_end){
        if( (time_sys >= time_snap) || ( (time_sys + dt) -
            char filename[256];
            sprintf(filename, "%s/%04d.dat", dir_name, id_
            FileHeader header;
            header.time   = time_sys;
            header.n_body = system_grav.getNumberOfParticl
            system_grav.writeParticleAscii(filename, heade
            time_snap += dt_snap;
        }

        calcEnergy(system_grav, Etot1, Ekin1, Epot1);

        if(PS::Comm::getRank() == 0){
            if( (time_sys >= time_diag) || ( (time_sys + d
                fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 <
                fprintf(stdout, "time: %10.7f energy error: %+e\n",
                        time_sys, (Etot1 - Etot0) / Etot0);
                time_diag += dt_diag;
            }
        }

        kick(system_grav, dt * 0.5);

        time_sys += dt;
        drift(system_grav, dt);

        if(n_loop % 4 == 0){
            dinfo.decomposeDomainAll(system_grav);
        }

        system_grav.exchangeParticle(dinfo);
#ifdef MULTI_WALK
        tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
                                                    RetrieveKernel,
                                                    tag_max,
                                                    system_grav,
                                                    dinfo,
                                                    n_walk_limit,
                                                    true);
#else
        tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                           CalcGravity<PS::SPJMonopole>,
                                           system_grav,
                                           dinfo);
#endif

        kick(system_grav, dt * 0.5);

        n_loop++;
    }

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_close();
#endif

    PS::Finalize();
    return 0;
}
```

# FDPSの初期化と終了処理



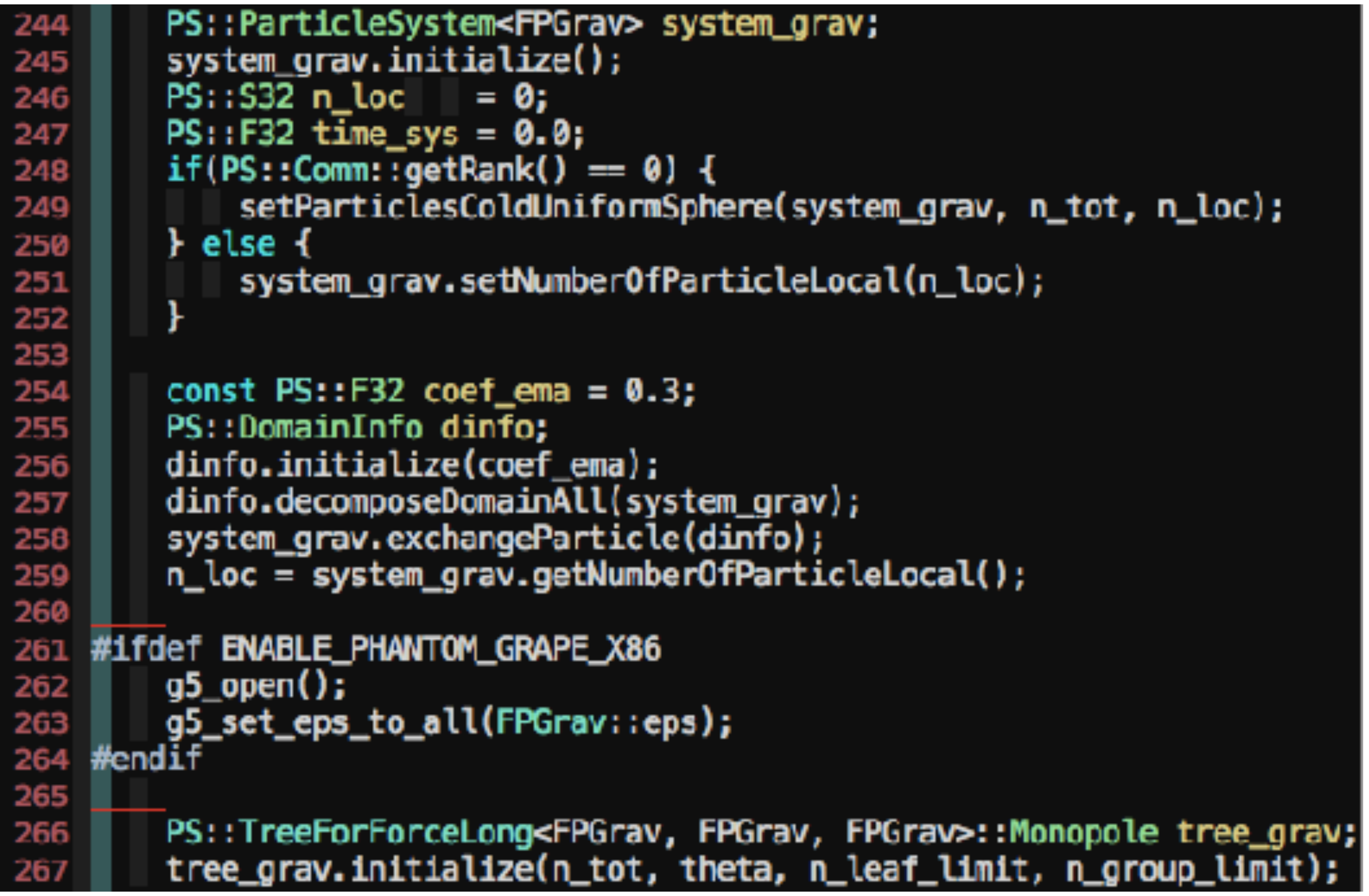
```
int main(int argc, char *argv[]) {
    std::cout<<std::setprecision(15);
    std::cerr<<std::setprecision(15);

    PS::Initialize(argc, argv);
    PS::F32 theta = 0.5;
    PS::S32 n_leaf_limit = 8;
    PS::S32 n_group_limit = 64;
    PS::F32 time_end = 10.0;
    PS::F32 dt = 1.0 / 128.0;
    PS::F32 dt_diag = 1.0 / 8.0;
    PS::F32 dt_snap = 1.0;
    char dir_name[1024];
    PS::S64 n_tot = 1024;
```



```
    PS::Finalize();
    return 0;
}
```

- 必ず最初にPS::Initialize()と最後にPS::Finalize()関数を呼ぶ

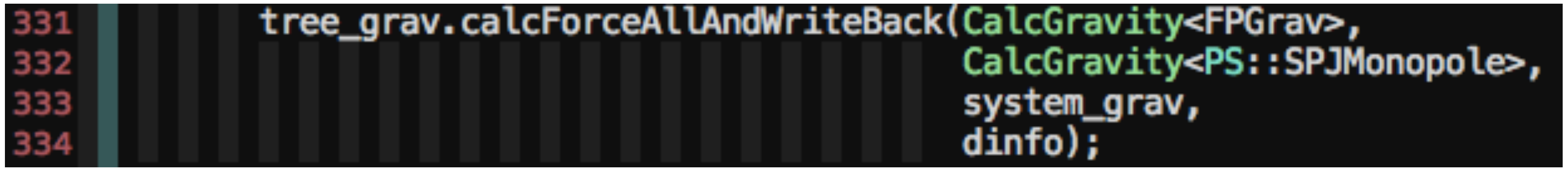
# FDPS各クラスの初期化



```
    PS::ParticleSystem<FPGrav> system_grav;
    system_grav.initialize();
    PS::S32 n_loc     = 0;
    PS::F32 time_sys = 0.0;
    if(PS::Comm::getRank() == 0) {
        setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
    } else {
        system_grav.setNumberOfParticleLocal(n_loc);
    }

    const PS::F32 coef_ema = 0.3;
    PS::DomainInfo dinfo;
    dinfo.initialize(coef_ema);
    dinfo.decomposeDomainAll(system_grav);
    system_grav.exchangeParticle(dinfo);
    n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif

    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
```

- 各クラスでは必ずInitialize()メンバ関数を呼ぶ必要がある
- ParticleSystemクラスは初期化後に初期条件の生成/読み込みを行う
- DomainInfoクラスは初期化後に最初の領域分割を行う(decomposeDomainAll)
- TreeForForceクラスは相互作用計算の形式によって呼び出すクラスが変わる(今回はTree法を使ったモノポールの長距離相互作用計算を行うもの)

# 相互作用の計算



```
tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                   CalcGravity<PS::SPJMonopole>,
                                   system_grav,
                                   dinfo);
```

- TreeForForce::calcForceAllAndWriteBackメンバ関数を呼ぶ

- 引数に普通の粒子の相互作用計算関数と超粒子の相互作用計算関数(本サンプルではテンプレート関数で両方に同じ関数を使用), ParticleSystem, DomainInfoを入れる

- その他オプションがあるがここでは割愛

## まとめ

- 500行程度でI/Oまでを含むTree法を用いた重力多体問題シミュレーションコードを記述
- OpenMPやMPIの並列化を意識するところ(ほぼ)ない
- コンパイルオプションを変更することで，MPIやOpenMPを利用可能