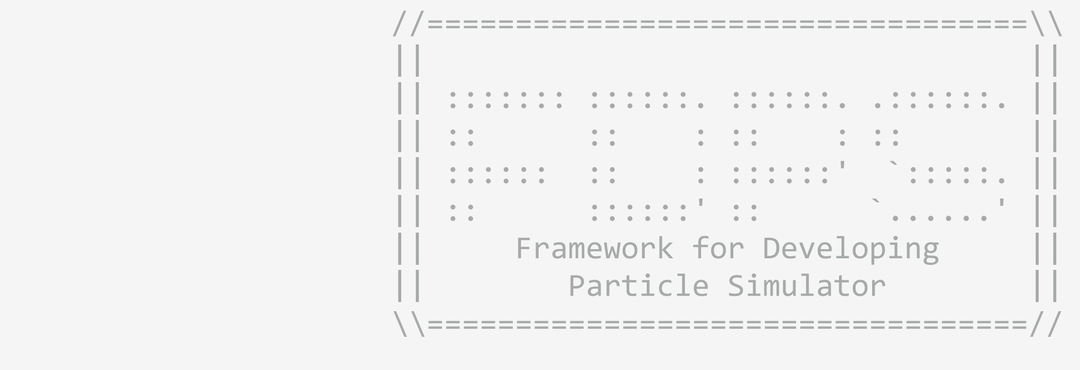
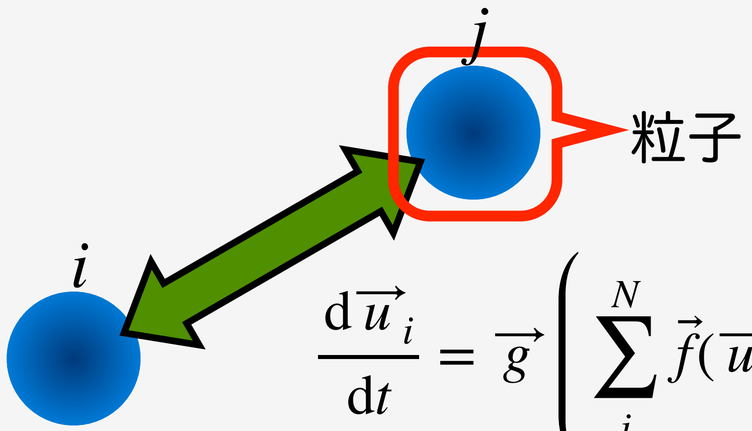
FDPS 講習会 実践編 (C++)

細野 七月 (JAMSTEC/RIKEN CCS)



FDPSが扱うもの

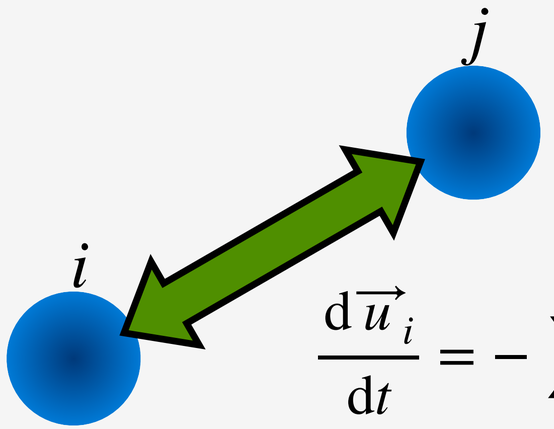
◆個々の要素(粒子)に作用する力が、粒子間相互作用の重ね合わせで記述できるもの



$$\frac{\mathrm{d}\vec{u}_i}{\mathrm{d}t} = \vec{g}\left(\sum_j^N \vec{f}(\vec{u}_i, \vec{u}_j), \vec{u}_i \right)$$

FDPSが扱うもの

◆個々の要素(粒子)に作用する力が、粒子間相互作用の重ね合わせで記述できるもの



$$\frac{\mathrm{d}\vec{u}_i}{\mathrm{d}t} = -\sum_j^N \frac{m_j}{|\vec{x}_i - \vec{x}_j|^3}\left(\vec{x}_i - \vec{x}_j\right)$$

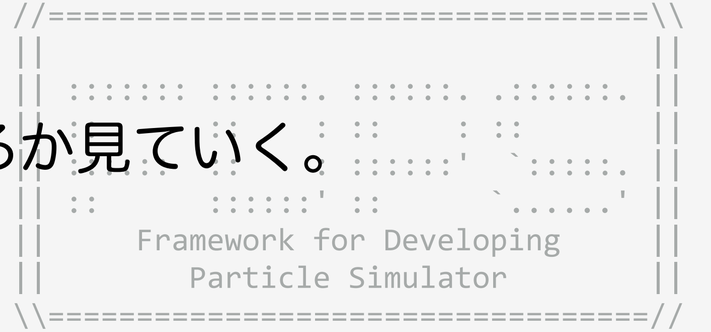
例　ニュートン重力

復習　習得しておきたいC++の機能

◆FDPS用いるにおいて、知っておきたいC++の機能は、

- 名前空間
  - PS::F64 など
- クラス
  - メンバ変数とメンバ関数
- テンプレート
  - template<typename T> など
- 標準ライブラリ
  - std:: など
  - ただし今回のサンプルコードではI/O用のstd::coutなどが使われている程度なので省略。

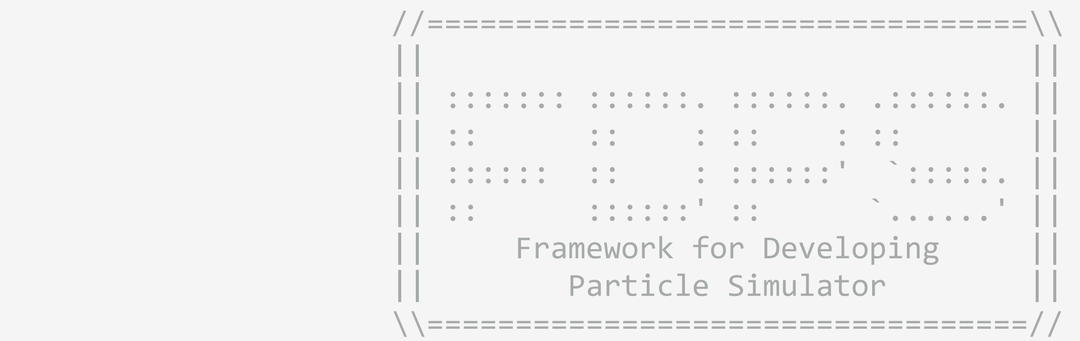
◆今から、サンプルコードでこれらが具体的にどのように使われているか見ていく。



コード構成

◆ユーザーが書くべきものは、

- #include <particle_simulator.h>
- 粒子クラスと必要なメンバ関数
- 相互作用関数
- 時間積分ルーチン
- I/O (粒子クラスのI/Oと、FileHeaderクラス)



サンプルコード

◆今回のサンプルコードの内容
- 粒子間相互作用は、重力
  - Phantom-GRAPE (Tanikawa+, 2011; 2012) ありとなし両方
- 時間積分法はleap-frog法
- 初期条件はその場生成
  - ファイル読み込みではない

◆ファイル構成
- user-defined.hpp
- nbody.cpp

他にも色々入っているが必要なものはすべてこの2つの中にある

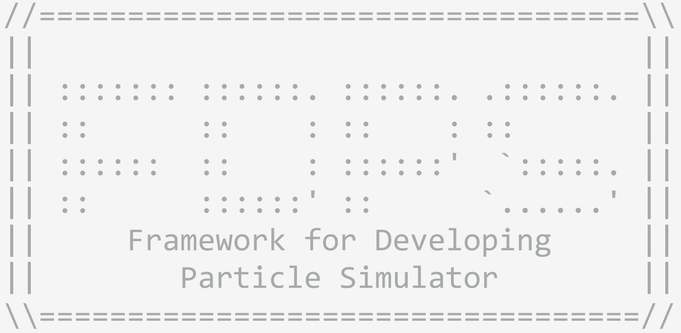
サンプルコード used_defined.hpp

```
#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;
```

サンプルコード used_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;
```

クラス



サンプルコード used_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);
    }
};
```

ファイルヘッダークラス

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
```

サンプルコード used_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);
    }
};
```

名前空間

ファイルヘッダークラス

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
```

```
//==================================\\
||                                  ||
||  ::::::: ::::::. ::::::. .:::::. ||
||  ::      ::    : ::    : ::      ||
||  ::::::  ::    : ::::::'  `::::. ||
||  ::      ::::::' ::      `....' ||
||     Framework for Developing     ||
||        Particle Simulator        ||
\\==================================//
```

サンプルコード used_defined.hpp

```
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;
```

粒子クラス

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
||  ::     ::    : ::    : ::      ||
||  ::::   ::    : ::::::'  `::::. ||
||  ::     ::::::' ::      `....' ||
||     Framework for Developing     ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード used_defined.hpp

```
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;
```

物理量

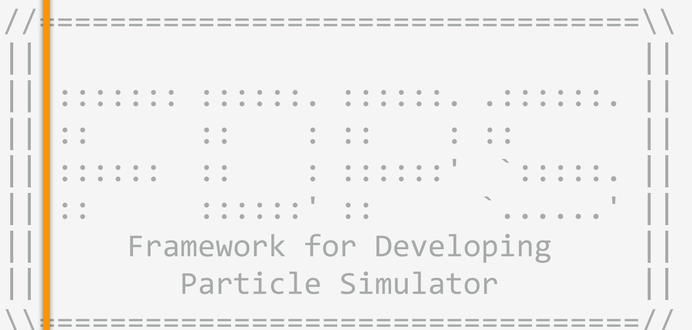
粒子クラス

```
//================================\\
||                                ||
||  ::::::: ::::::. ::::::. .::::::. ||
||  ::      ::    : ::    : ::      ||
||  ::::::  ::    : ::::::'  `::::.  ||
||  ::      ::::::' ::      `....'  ||
||     Framework for Developing     ||
||       Particle Simulator         ||
\\================================//
```

サンプルコード used_defined.hpp

```
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,
```

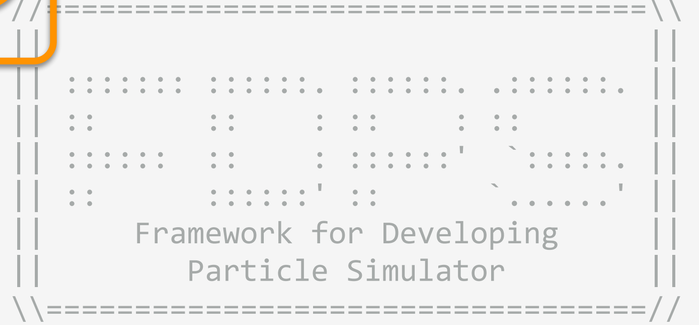
メンバー関数



サンプルコード used_defined.hpp

I/O用メンバー関数

```
        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 EPGrav * iptcl,
                 const PS::S32 ni,
```



サンプルコード used_defined.hpp

```
    }
};

#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;
```

関数テンプレート

Phantom GRAPEあり

サンプルコード used_defined.hpp

```
            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();
```

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `....' ||
||     Framework for Developing     ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード used_defined.hpp

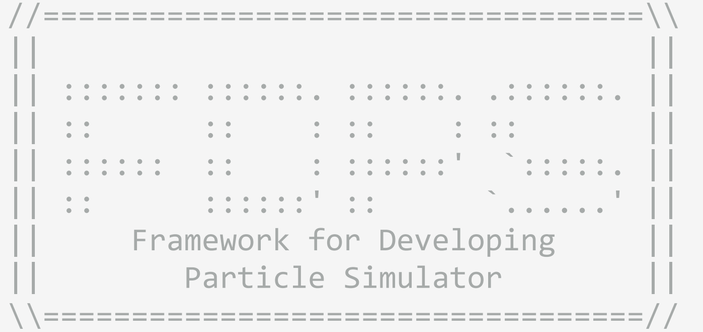
```
template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParticleJ * ep_j,
                 const PS::S32 n_jp,
                 FPGrav * force) {
    PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
    for(PS::S32 i = 0; i < n_ip; i++){
        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;
```

相互作用関数テンプレート

メンバ関数呼び出し

サンプルコード used_defined.hpp

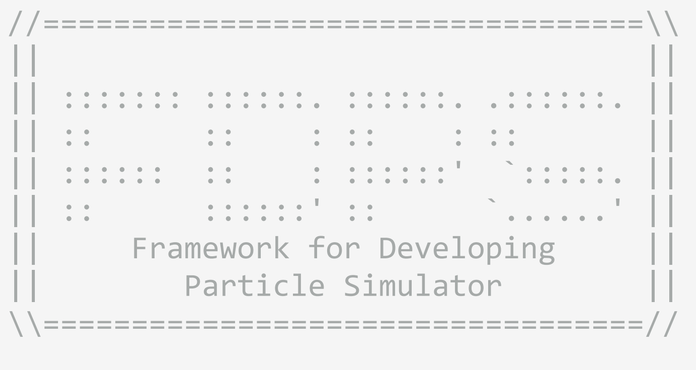
```
                PS::F64    r3_inv = rij * rij + eps2;
                PS::F64    r_inv  = 1.0/sqrt(r3_inv);
                r3_inv  = r_inv * r_inv;
                r_inv  *= ep_j[j].getCharge();
                r3_inv *= r_inv;
                ai     -= r3_inv * rij;
                poti   -= r_inv;
            }
            force[i].acc += ai;
            force[i].pot += poti;
        }
    }

    #endif
```



サンプルコード user_defined.hpp

◆user_defined.hppはこれだけ。

おおよそ150行。



サンプルコード nbody.cpp

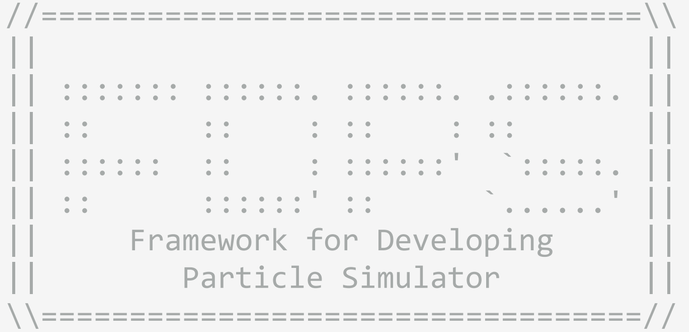
```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include “user-defined.hpp"

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);
```

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .::::::. ||
|| ::      ::    : ::    : ::       ||
|| ::::::  ::    : ::::::'  `::::.  ||
|| ::      ::::::' ::      `....'  ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
#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"

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);
```

FDPSのヘッダー読み込み



サンプルコード nbody.cpp

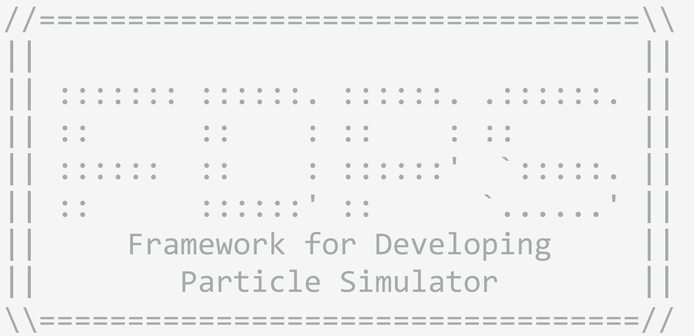
```
}

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;
    }
}

template<class Tpsys>
void calcEnergy(const Tpsys & system,
                PS::F64 & etot,
                PS::F64 & ekin,
                PS::F64 & epot,
                const bool clear=true){
    if(clear){
```

粒子数が取得可能

サンプルコード nbody.cpp

```
}

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;
    }
}

template<class Tpsys>
void calcEnergy(const Tpsys & system,
                PS::F64 & etot,
                PS::F64 & ekin,
                PS::F64 & epot,
                const bool clear=true){
    if(clear){
```

粒子群クラスに[i]をつけると粒子データにアクセス可能



サンプルコード nbody.cpp

メイン関数開始

```
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::S32 c;
    sprintf(dir_name,"./result");
    opterr = 0;
    while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
        switch(c){
        case 'o':
            sprintf(dir_name,optarg);
            break;
        case 't':
            theta = atof(optarg);
            std::cerr << "theta =" << theta << std::endl;
            break;
```

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
||    ::    ::    : ::    : ::      ||
||    ::    ::    : ::::::'  `::::. ||
||    ::    ::::::' ::      `....' ||
||     Framework for Developing     ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
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::S32 c;
    sprintf(dir_name,"./result");
    opterr = 0;
    while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
        switch(c){
        case 'o':
            sprintf(dir_name,optarg);
            break;
        case 't':
            theta = atof(optarg);
            std::cerr << "theta =" << theta << std::endl;
            break;
```

FDPSの初期化

```
//=================================\\
||                                 ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `....' ||
||    Framework for Developing     ||
||       Particle Simulator        ||
\\=================================//
```

サンプルコード nbody.cpp

```
        fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
    }

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

粒子群クラスの生成と初期化

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .::::::. ||
|| ::      ::    : ::    : ::       ||
|| ::::::  ::    : ::::::'  `::::::. ||
|| ::      ::::::' ::      `......' ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
        fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
    }

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

ドメイン情報クラスの生成と初期化、領域分割

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::::. ||
|| ::      ::    : ::    : ::       ||
|| ::::::  ::    : ::::::'  `::::::. ||
|| ::      ::::::' ::      `......' ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
        fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
    }

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

粒子交換

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .::::::. ||
|| ::      ::    : ::    : ::       ||
|| ::::::  ::    : ::::::'  `::::.  ||
|| ::      ::::::' ::      `....'  ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
    n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endi
    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
#ifdef MULTI_WALK
    const PS::S32 n_walk_limit = 200;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
                                                RetrieveKernel,
                                                tag_max,
                                                system_grav,
                                                dinfo,
                                                n_walk_limit);
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                       CalcGravity<PS::SPJMonopole>,
                                       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;
```

相互作用ツリークラスの
生成と初期化

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `....' ||
||    Framework for Developing     ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
                                                    tag_max,
                                                    system_grav,
                                                    dinfo,
                                                    n_walk_limit);
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                       CalcGravity<PS::SPJMonopole>,
                                       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) - time_snap ) > (time_snap - time_sys) ){
            char filename[256];
            sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
            FileHeader header;
            header.time   = time_sys;
            header.n_body = system_grav.getNumberOfParticleGlobal();
            system_grav.writeParticleAscii(filename, header);
            time_snap += dt_snap;
        }
        calcEnergy(system_grav, Etot1, Ekin1, Epot1);
```

相互作用関数を用いた力の計算

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `....' ||
||     Framework for Developing     ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
    PS::S32 id_snap = 0;
    while(time_sys < time_end){
        if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
            char filename[256];
            sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
            FileHeader header;
            header.time   = time_sys;
            header.n_body = system_grav.getNumberOfParticleGlobal();
            system_grav.writeParticleAscii(filename, header);
            time_snap += dt_snap;
        }
        calcEnergy(system_grav, Etot1, Ekin1, Epot1);
        if(PS::Comm::getRank() == 0){
            if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys) ){
                fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 << std::endl;
                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);
```

時間積分

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `.....' ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

```
            tree_grav.calcForceAllAndWriteBack(CalcGravity<PFGrav>,
                                               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;
}
```

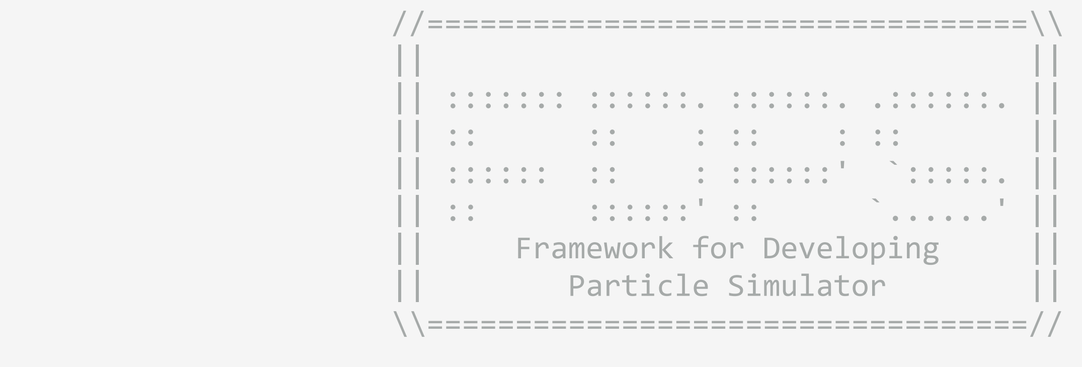
FDPS終了

```
//==================================\\
||                                  ||
|| ::::::: ::::::. ::::::. .:::::. ||
|| ::      ::    : ::    : ::      ||
|| ::::::  ::    : ::::::'  `::::. ||
|| ::      ::::::' ::      `....' ||
||    Framework for Developing      ||
||       Particle Simulator         ||
\\==================================//
```

サンプルコード nbody.cpp

◆nbody.cppはこれだけ。

おおよそ350行。

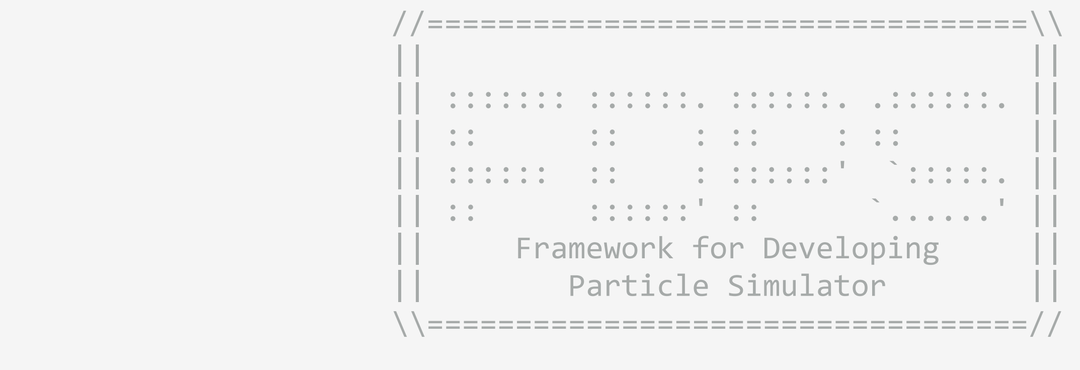


最後に

◆ユーザーが書かなくてはいけないのは、150+350 = 500行 程度。

この500行の中には、コマンドライン引数などの解析も含むため、
そのようなものが必要なければ実際には更に少なくて済む。

◆コード内に、並列化を意識しなくてはならないような場所は無かった。
つまり、コンパイルの方法だけでOpenMPやMPIを切り替えられる。



実習の流れ

◆実習用のFOCUSスパコンにログイン後、サンプルコードを
(1)並列化なし (2)OpenMP (3)OpenMP + MPI
の3パターンに関してコンパイルし、実行。
サンプルコードは以下の2つ。
- 重力 cold collapse
- 流体 (Smoothed Particle Hydrodynamics法) adiabatic sphere collapse
実行が終わったら、結果の解析

◆詳しくは以下の資料をご覧ください。
http://v1.jmlab.jp/~makino/wordpress-3.5.1/wp-content/uploads/2015/06/resume.pdf