FDPSの概要説明

岩澤全規
理化学研究所 計算科学研究センター
粒子系シミュレータ研究チーム
エクサスケールコンピューティング開発プロジェクト コデザインチーム

2018/08/03 FDPS 講習会

FDPSとは

- Framework for Developing Particle Simulator
- 大規模並列粒子シミュレーションコードの開発を支援するフレームワーク
- 重力N体、SPH、分子動力学、粉体、etc.
- 支配方程式

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

粒子データのベクトル

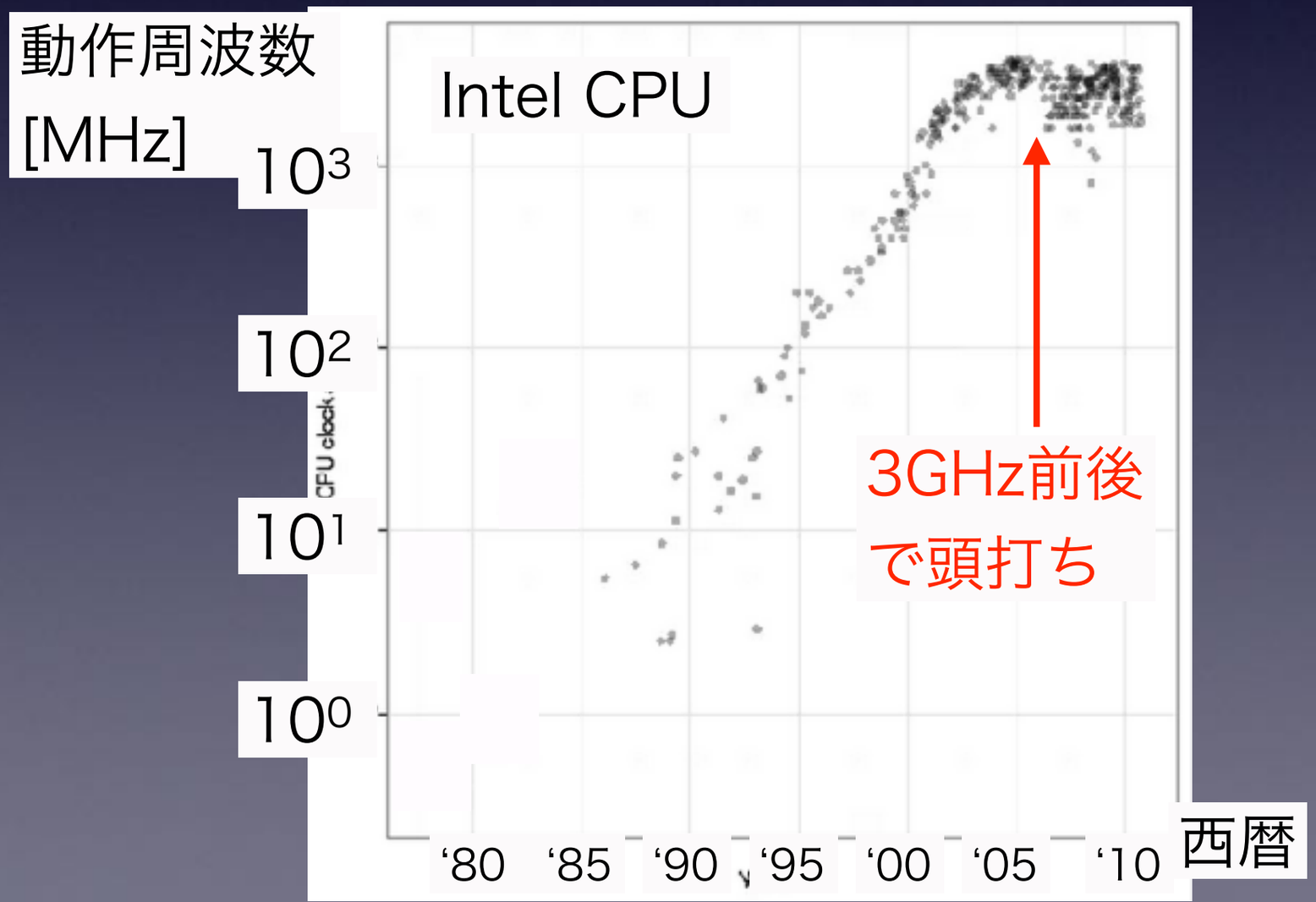
粒子の持つ物理量をその導関数に変換する関数

粒子間相互作用を表す関数

大規模並列粒子シミュレーションの必要性

- 大粒子数で積分時間の長いシミュレーション

- 逐次計算の速度はもう速くならない



大規模並列
粒子シミュレーションの困難

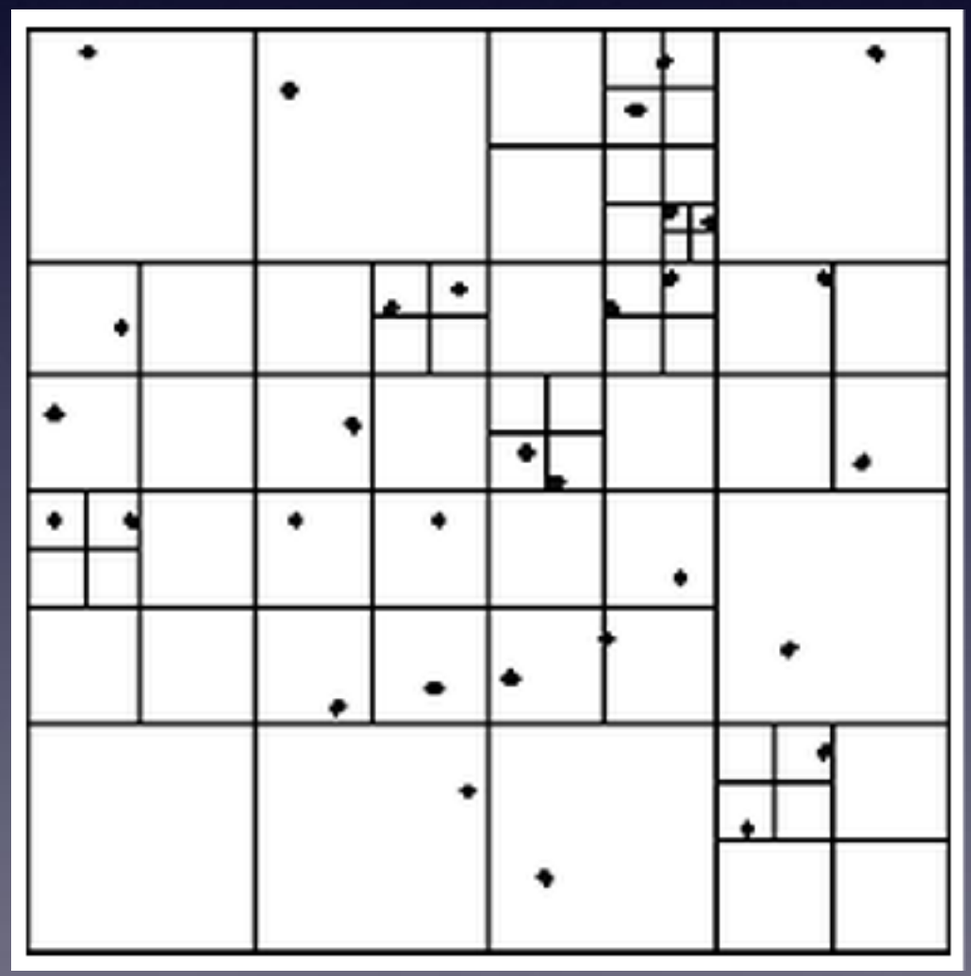
- 分散メモリ環境での並列化
  - 計算領域の分割と粒子データの交換
  - 相互作用計算のための粒子データの交換
- 共有メモリ環境での並列化
  - ツリー構造のマルチウォーク
  - 相互作用計算の負荷分散
- １コア内での並列化
  - SIMD演算器の有効利用

実は並列でなくても、、、

- キャッシュメモリの有効利用
- ツリー構造の構築

$$\frac{d\vec{u}_i}{dt} = \vec{g}\left(\sum_{j}^{N} \vec{f}(\vec{u}_i, \vec{u}_j), \vec{u}_i\right)$$

高速にNを小さい数に
減らす方法



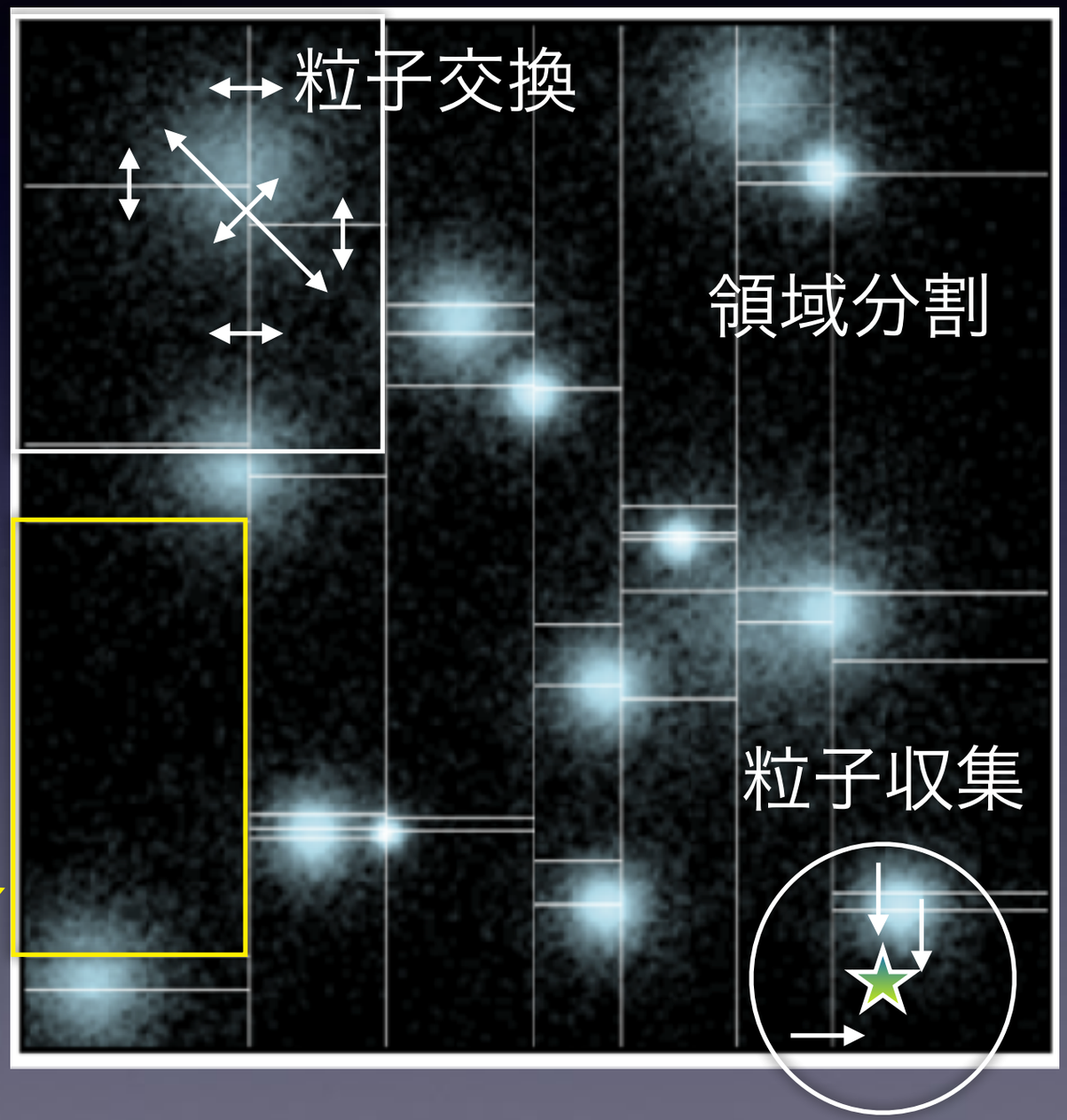
粒子シミュレーションの手順

FDPS

- 計算領域の分割
- 粒子データの交換
- 相互作用計算のための粒子データの収集

- 実際の相互作用の計算

- 粒子の軌道積分



1つのプロセスが
担当する領域

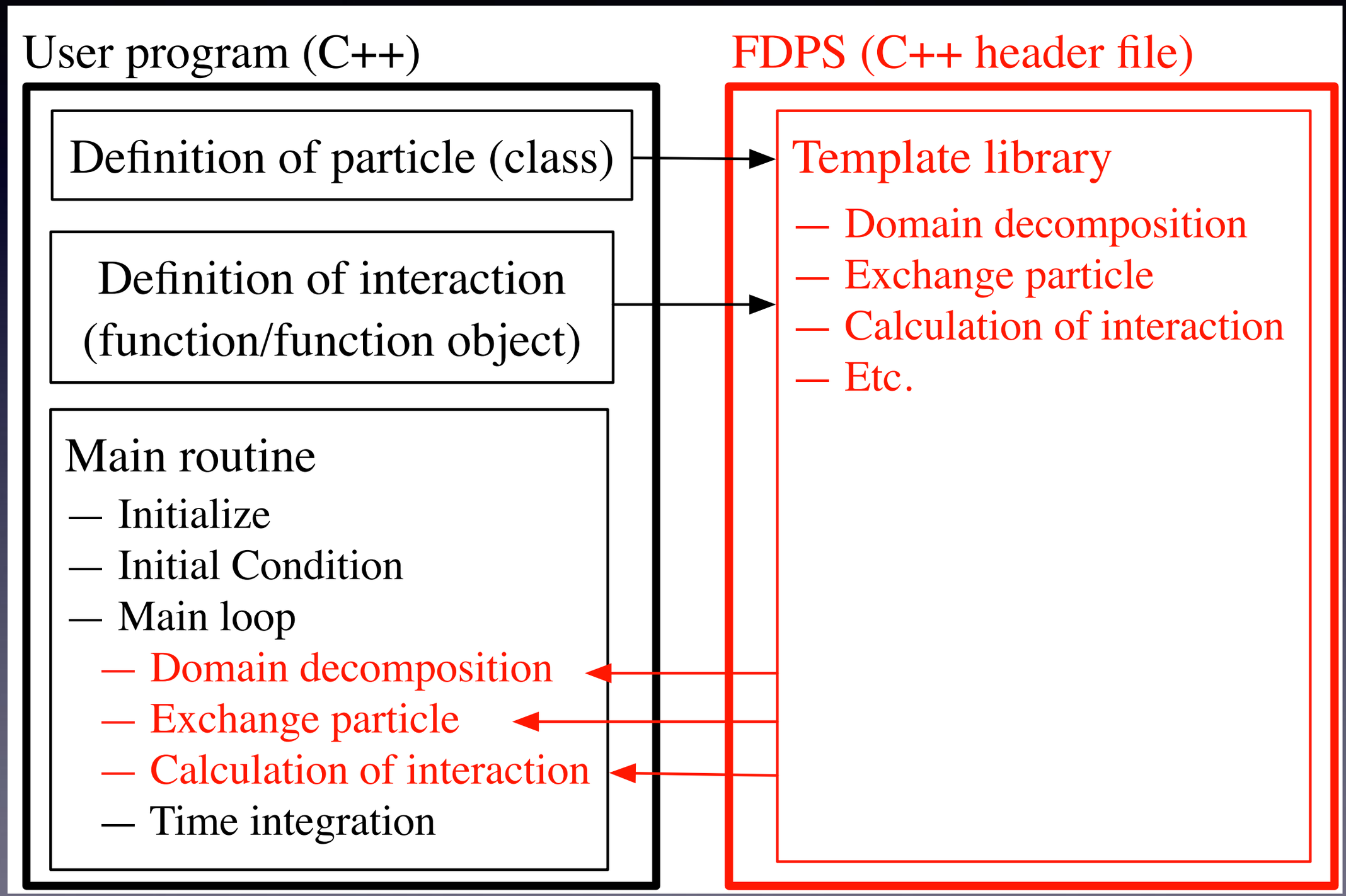
FDPSの実装方針(1)

- 内部実装の言語としてC++を選択
  - 高い自由度
    - 粒子データの定義に**クラス**を利用
    - 相互作用の定義に**関数ポインタ**・**関数オブジェクト**を利用
  - 高い性能
    - 上のクラス・関数ポインタ・関数オブジェクトを受け取るために**テンプレートクラス**を利用
      - コンパイル時に静的にコード生成するため

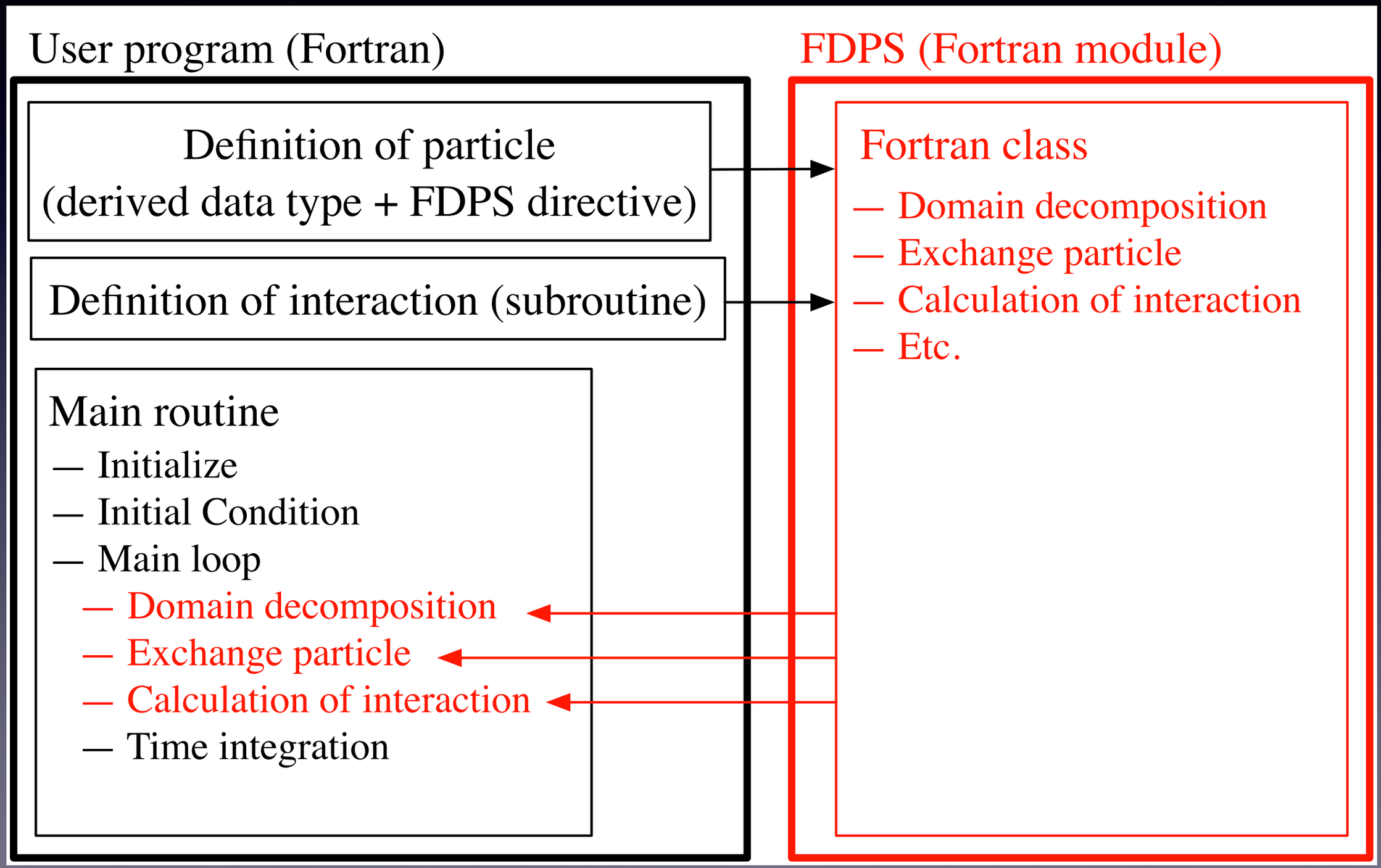
FDPSの実装方針(2)

- 並列化
  - 分散メモリ環境(ノード間)：MPI
  - 共有メモリ環境(ノード内)：OpenMP
    或いは
    GPU

FDPSの基本設計


User program (C++)
Definition of particle (class)
Definition of interaction (function/function object)
Main routine
— Initialize
— Initial Condition
— Main loop
— Domain decomposition
— Exchange particle
— Calculation of interaction
— Time integration
FDPS (C++ header file)
Template library
— Domain decomposition
— Exchange particle
— Calculation of interaction
— Etc.

基本設計 (Fortranの場合)



FDPSのリリースノート

- 2012年11月 FDPSの開発開始
- 2015年3月 FDPS Ver. 1.0
- 2016年1月 FDPS Ver. 2.0
  - アクセラレータ利用のために、Multiwalk法(Hamada et al 2009)を実装
- 2016年12月 FDPS Ver. 3.0
  - Fortran Interfaceの実装
- 2017年11月 FDPS Ver. 4.0
  - SPH法やMD計算等で計算を高速化するために、相互作用リスト再利用のアルゴリズムの実装

サンプルコード(N体)

粒子型の定義
(Fortranのクラス)

粒子間の相互作用の定義
(Fortranのサブルーチン)

FDPSモジュールをインクルード

メインルーチン(メイン関数)の実装

大規模並列N体コードが
200行程度で書ける！

```
moduleuser_defined_types
    use,intrinsic::iso_c_binding
    usefdps_vector
    usefdps_super_particle
    implicitnone

    type,public,bind(c)::full_particle!$fdpsFP,EPI,EPJ,Force
        !$fdpscopyFromForcefull_particle(pot,pot)(acc,acc)
        !$fdpscopyFromFPfull_particle(id,id)(mass,mass)(eps,eps)(pos,pos)
        !$fdpsclearid=keep,mass=keep,eps=keep,pos=keep,vel=keep
        integer(kind=c_long_long)::id
        real(kind=c_double)mass!$fdpscharge
        real(kind=c_double)::eps
        type(fdps_f64vec)::pos!$fdpsposition
        type(fdps_f64vec)::vel!$fdpsvelocity
        real(kind=c_double)::pot
        type(fdps_f64vec)::acc
    endtypefull_particle

    contains

    subroutinecalc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f)bind(c)
        integer(c_int),intent(in),value::n_ip,n_jp
        type(full_particle),dimension(n_ip),intent(in)::ep_i
        type(full_particle),dimension(n_jp),intent(in)::ep_j
        type(full_particle),dimension(n_ip),intent(inout)::f
        integer(c_int)::i,j
        real(c_double)::eps2,poti,r3_inv,r_inv
        type(fdps_f64vec)::xi,ai,rij

        doi=1,n_ip
            eps2=ep_i(i)%eps*ep_i(i)%eps
            xi=ep_i(i)%pos
            ai=0.0d0
            poti=0.0d0
            doj=1,n_jp
                rij%x=xi%x-ep_j(j)%pos%x
                rij%y=xi%y-ep_j(j)%pos%y
                rij%z=xi%z-ep_j(j)%pos%z
                r3_inv=rij%x*rij%x&
                        + r i j % y * r i j % y &
                        + r i j % z * r i j % z &
                        + e p s 2
                r_inv=1.0d0/sqrt(r3_inv)
                r3_inv=r_inv*r_inv
                r_inv=r_inv*ep_j(j)%mass
                r3_inv=r3_inv*r_inv
                ai%x=ai%x-r3_inv*rij%x
                ai%y=ai%y-r3_inv*rij%y
                ai%z=ai%z-r3_inv*rij%z
                poti=poti-r_inv
            enddo
            f(i)%pot=f(i)%pot+poti
            f(i)%acc=f(i)%acc+ai
        enddo

    endsubroutinecalc_gravity_pp

    subroutinecalc_gravity_psp(ep_i,n_ip,ep_j,n_jp,f)bind(c)
        integer(c_int),intent(in),value::n_ip,n_jp
        type(full_particle),dimension(n_ip),intent(in)::ep_i
        type(fdps_spj_monopole),dimension(n_jp),intent(in)::ep_j
        type(full_particle),dimension(n_ip),intent(inout)::f
        integer(c_int)::i,j
        real(c_double)::eps2,poti,r3_inv,r_inv
        type(fdps_f64vec)::xi,ai,rij

        doi=1,n_ip
            eps2=ep_i(i)%eps*ep_i(i)%eps
            xi=ep_i(i)%pos
            ai=0.0d0
            poti=0.0d0
            doj=1,n_jp
                rij%x=xi%x-ep_j(j)%pos%x
                rij%y=xi%y-ep_j(j)%pos%y
                rij%z=xi%z-ep_j(j)%pos%z
                r3_inv=rij%x*rij%x&
                        + r i j % y * r i j % y &
                        + r i j % z * r i j % z &
                        + e p s 2
                r_inv=1.0d0/sqrt(r3_inv)
                r3_inv=r_inv*r_inv
                r_inv=r_inv*ep_j(j)%mass
                r3_inv=r3_inv*r_inv
                ai%x=ai%x-r3_inv*rij%x
                ai%y=ai%y-r3_inv*rij%y
                ai%z=ai%z-r3_inv*rij%z
                poti=poti-r_inv
            enddo
            f(i)%pot=f(i)%pot+poti
            f(i)%acc=f(i)%acc+ai
        enddo

    endsubroutinecalc_gravity_psp
endmoduleuser_defined_types
```

```
subroutinef_main()
    usefdps_module
    useuser_defined_types
    implicitnone
    doubleprecision,parameter::time_end=10.0d0
    doubleprecision,parameter::dt=1.0d0/128.0d0
    integer::i,j,k,ierr
    integer::psys_num,dinfo_num,tree_num
    character(len=64)::tree_type
    doubleprecision::time_sys=0.0d0
    type(fdps_controller)::fdps_ctrl
    callfdps_ctrl%PS_Initialize()
    callfdps_ctrl%create_dinfo(dinfo_num)
    callfdps_ctrl%init_dinfo(dinfo_num)
    callfdps_ctrl%create_psys(psys_num,'full_particle')
    callfdps_ctrl%init_psys(psys_num)
    tree_type="Long,full_particle,full_particle,full_particle,Monopole"
    callfdps_ctrl%create_tree(tree_num,tree_type)
    callfdps_ctrl%init_tree(tree_num,0)
    callread_IC(fdps_ctrl,psys_num)
    callcalc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
    do
        callkick(fdps_ctrl,psys_num,0.5d0*dt)
        time_sys=time_sys+dt
        calldrift(fdps_ctrl,psys_num,dt)
        callcalc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
        callkick(fdps_ctrl,psys_num,0.5d0*dt)
        if(time_sys>=time_end)exit
    enddo
    callfdps_ctrl%PS_Finalize()
endsubroutinef_main

subroutinecalc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
    usefdps_module
    useuser_defined_types
    implicitnone
    type(fdps_controller),intent(IN)::fdps_ctrl
    integer,intent(IN)::psys_num,dinfo_num,tree_num
    type(c_funptr)::pfunc_ep_ep,pfunc_ep_sp
    callfdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
    callfdps_ctrl%exchange_particle(psys_num,dinfo_num)
    pfunc_ep_ep=c_funloc(calc_gravity_pp)
    pfunc_ep_sp=c_funloc(calc_gravity_psp)
    callfdps_ctrl%calc_force_all_and_write_back(tree_num,&
                                                pfunc_ep_ep,&
                                                pfunc_ep_sp,&
                                                psys_num,&
                                                dinfo_num)
endsubroutinecalc_gravity

subroutinekick(fdps_ctrl,psys_num,dt)
    usefdps_vector
    usefdps_module
    useuser_defined_types
    implicitnone
    type(fdps_controller),intent(IN)::fdps_ctrl
    integer,intent(IN)::psys_num
    doubleprecision,intent(IN)::dt
    integer::i,nptcl_loc
    type(full_particle),dimension(:),pointer::ptcl
    nptcl_loc=fdps_ctrl%get_nptcl_loc(psys_num)
    callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
    doi=1,nptcl_loc
        ptcl(i)%vel=ptcl(i)%vel+ptcl(i)%acc*dt
    enddo
    nullify(ptcl)
endsubroutinekick

subroutinedrift(fdps_ctrl,psys_num,dt)
    usefdps_vector
    usefdps_module
    useuser_defined_types
    implicitnone
    type(fdps_controller),intent(IN)::fdps_ctrl
    integer,intent(IN)::psys_num
    doubleprecision,intent(IN)::dt
    integer::i,nptcl_loc
    type(full_particle),dimension(:),pointer::ptcl
    nptcl_loc=fdps_ctrl%get_nptcl_loc(psys_num)
    callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
    doi=1,nptcl_loc
        ptcl(i)%pos=ptcl(i)%pos+ptcl(i)%vel*dt
    enddo
    nullify(ptcl)
endsubroutinedrift

subroutineread_IC(fdps_ctrl,psys_num)
    usefdps_module
    useuser_defined_types
    implicitnone
    type(fdps_controller),intent(IN)::fdps_ctrl
    integer,intent(IN)::psys_num
    character(len=16),parameter::root_dir="input_data"
    character(len=16),parameter::file_prefix="proc"
    integer::i,myrank,nptcl_loc
    character(len=64)::fname,proc_num
    type(full_particle),dimension(:),pointer::ptcl
    myrank=fdps_ctrl%get_rank()
    write(proc_num,"(i5.5)")myrank
    fname=trim(root_dir)//"/"&
          //trim(file_prefix)//proc_num//".dat"
    open(unit=9,file=trim(fname),action='read',form='unformatted',&
         access='stream',status='old')
        read(9)nptcl_loc
        callfdps_ctrl%set_nptcl_loc(psys_num,nptcl_loc)
        callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
        doi=1,nptcl_loc
            read(9)ptcl(i)%id,ptcl(i)%mass,ptcl(i)%eps,&
                   ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z,&
                   ptcl(i)%vel%x,ptcl(i)%vel%y,ptcl(i)%vel%z
        enddo
    close(unit=9)
    nullify(ptcl)
endsubroutineread_IC
```

サンプルコード(N体)

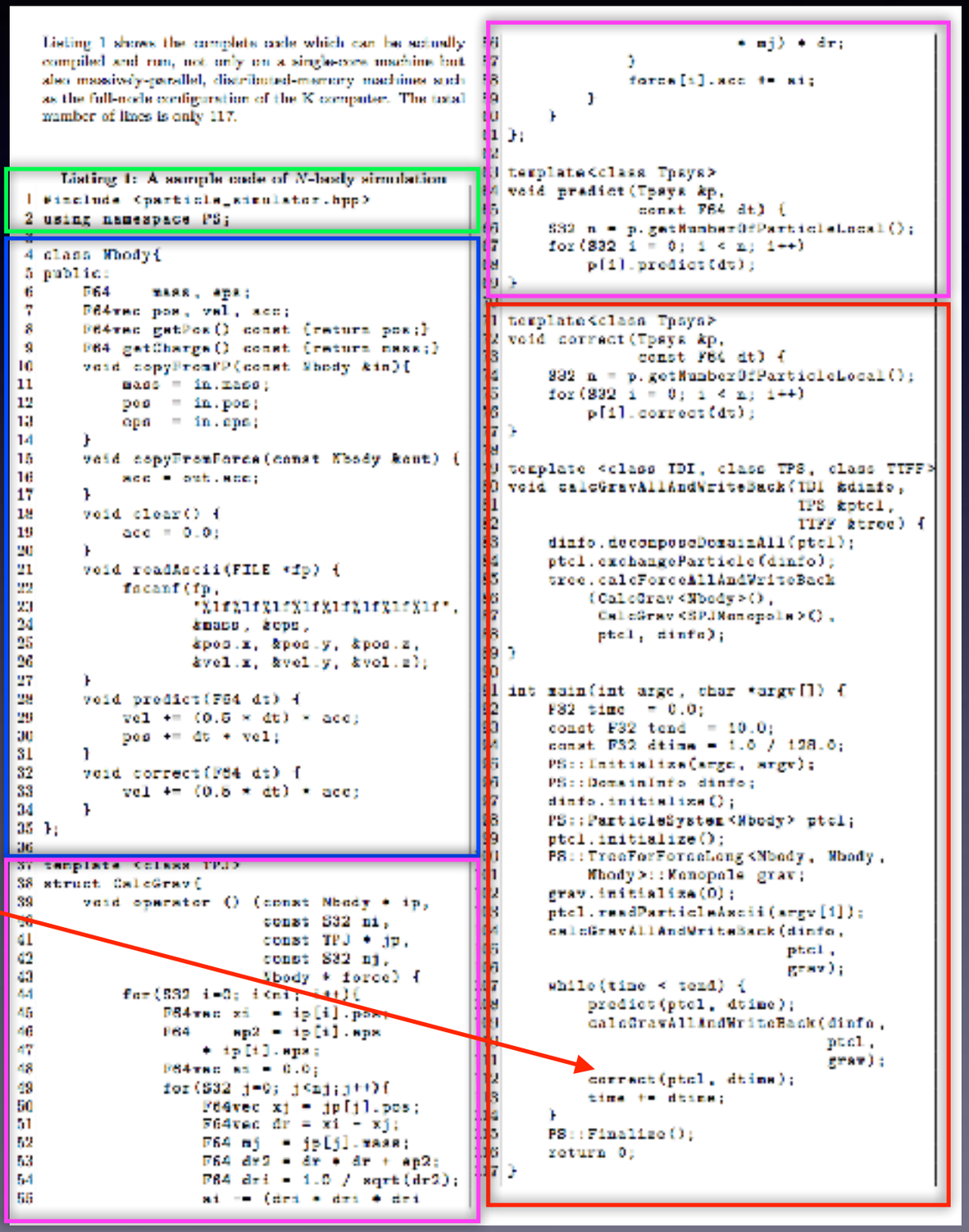
FDPSのインストール(ヘッダファイルをインクルードするだけ)

粒子データの定義
(C++のクラス)

粒子間の相互作用の定義
(C++の関数オブジェクト
または関数ポインタ)

メインルーチン(メイン
関数)の実装

大規模並列N体コードが
117行で書ける！



Listing 1 shows the complete code which can be actually compiled and run, not only on a single-core machine but also massively-parallel, distributed-memory machines such as the full-node configuration of the K computer. The total number of lines is only 117.

Listing 1: A sample code of N-body simulation

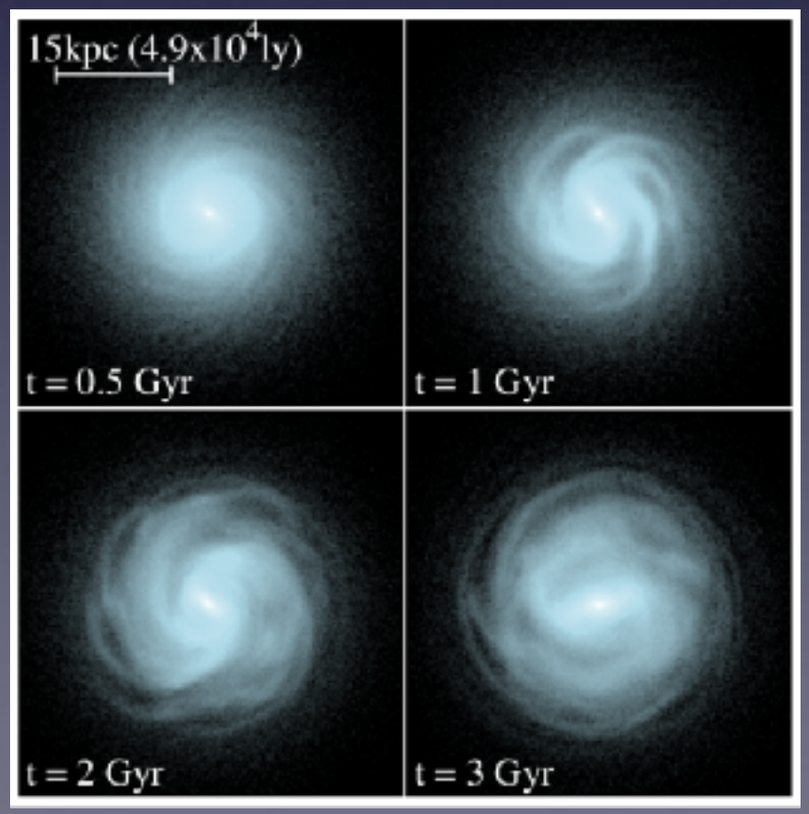
```
#include <particle_simulator.hpp>
using namespace PS;

class Nbody{
public:
    F64    mass, eps;
    F64vec pos, vel, acc;
    F64vec getPos() const {return pos;}
    F64 getCharge() const {return mass;}
    void copyFromFP(const Nbody &in){
        mass = in.mass;
        pos  = in.pos;
        eps  = in.eps;
    }
    void copyFromForce(const Nbody &out) {
        acc = out.acc;
    }
    void clear() {
        acc = 0.0;
    }
    void readAscii(FILE *fp) {
        fscanf(fp,
               "%lf%lf%lf%lf%lf%lf%lf%lf",
               &mass, &eps,
               &pos.x, &pos.y, &pos.z,
               &vel.x, &vel.y, &vel.z);
    }
    void predict(F64 dt) {
        vel += (0.5 * dt) * acc;
        pos += dt * vel;
    }
    void correct(F64 dt) {
        vel += (0.5 * dt) * acc;
    }
};

template<class TPJ>
struct CalcGrav{
    void operator () (const Nbody * ip,
                      const S32 ni,
                      const TPJ * jp,
                      const S32 nj,
                      Nbody * force) {
        for(S32 i=0; i<ni; ++i){
            F64vec xi  = ip[i].pos;
            F64    ep2 = ip[i].eps
                * ip[i].eps;
            F64vec ai = 0.0;
            for(S32 j=0; j<nj;j++){
                F64vec xj = jp[j].pos;
                F64vec dr = xi - xj;
                F64 mj  = jp[j].mass;
                F64 dr2 = dr * dr + ep2;
                F64 dri = 1.0 / sqrt(dr2);
                ai -= (dri * dri * dri
                       * mj) * dr;
            }
            force[i].acc += ai;
        }
    }
};

template<class Tpsys>
void predict(Tpsys &p,
             const F64 dt) {
    S32 n = p.getNumberOfParticleLocal();
    for(S32 i = 0; i < n; i++)
        p[i].predict(dt);
}

template<class Tpsys>
void correct(Tpsys &p,
             const F64 dt) {
    S32 n = p.getNumberOfParticleLocal();
    for(S32 i = 0; i < n; i++)
        p[i].correct(dt);
}

template <class TDI, class TPS, class TTFF>
void calcGravAllAndWriteBack(TDI &dinfo,
                             TPS &ptcl,
                             TTFF &tree) {
    dinfo.decomposeDomainAll(ptcl);
    ptcl.exchangeParticle(dinfo);
    tree.calcForceAllAndWriteBack
        (CalcGrav<Nbody>(),
         CalcGrav<SPJMonopole>(),
         ptcl, dinfo);
}

int main(int argc, char *argv[]) {
    F32 time  = 0.0;
    const F32 tend  = 10.0;
    const F32 dtime = 1.0 / 128.0;
    PS::Initialize(argc, argv);
    PS::DomainInfo dinfo;
    dinfo.initialize();
    PS::ParticleSystem<Nbody> ptcl;
    ptcl.initialize();
    PS::TreeForForceLong<Nbody, Nbody,
        Nbody>::Monopole grav;
    grav.initialize(0);
    ptcl.readParticleAscii(argv[1]);
    calcGravAllAndWriteBack(dinfo,
                            ptcl,
                            grav);
    while(time < tend) {
        predict(ptcl, dtime);
        calcGravAllAndWriteBack(dinfo,
                                ptcl,
                                grav);
        correct(ptcl, dtime);
        time += dtime;
    }
    PS::Finalize();
    return 0;
}
```

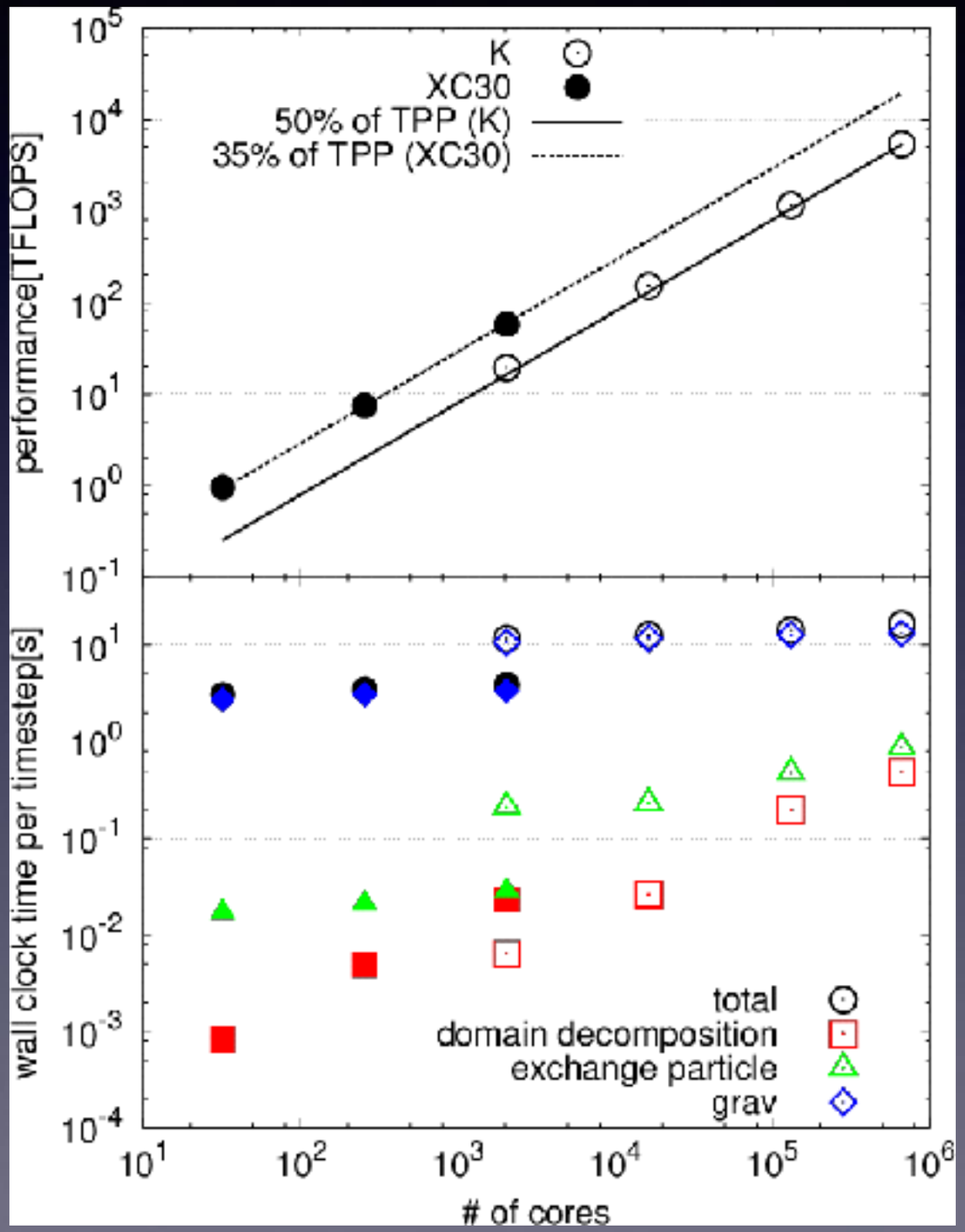
重要なポイント

- ユーザーはMPIやOpenMPを考えなくてよい
- 相互作用関数の実装について
  - ２重ループ：複数の粒子に対する複数の粒子からの作用の計算
  - チューニングが必要(FDPSチームに相談可)
    - 除算回数の最小化
    - SIMD演算器の有効利用

性能(N体)

- 円盤銀河
- 粒子数: 2.7×10^5/core
- 精度: $\Theta=0.4$ 四重極
- 京コンピュータ, XC30

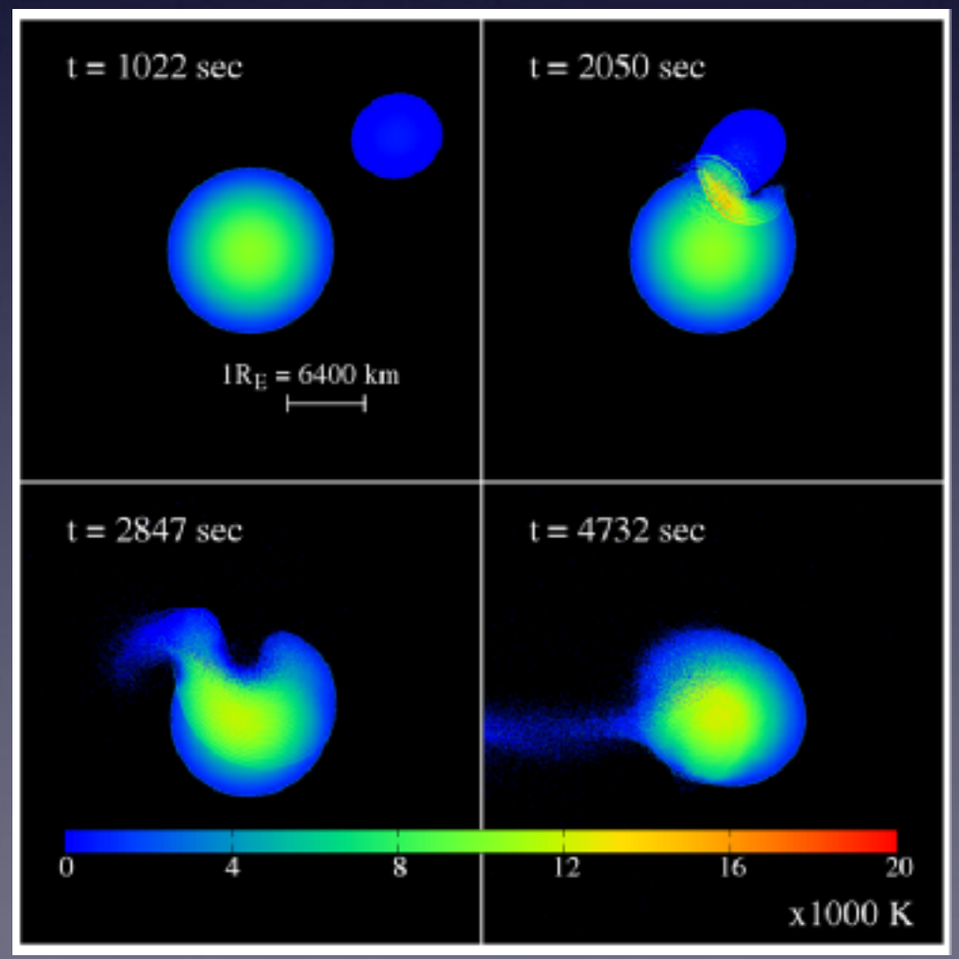


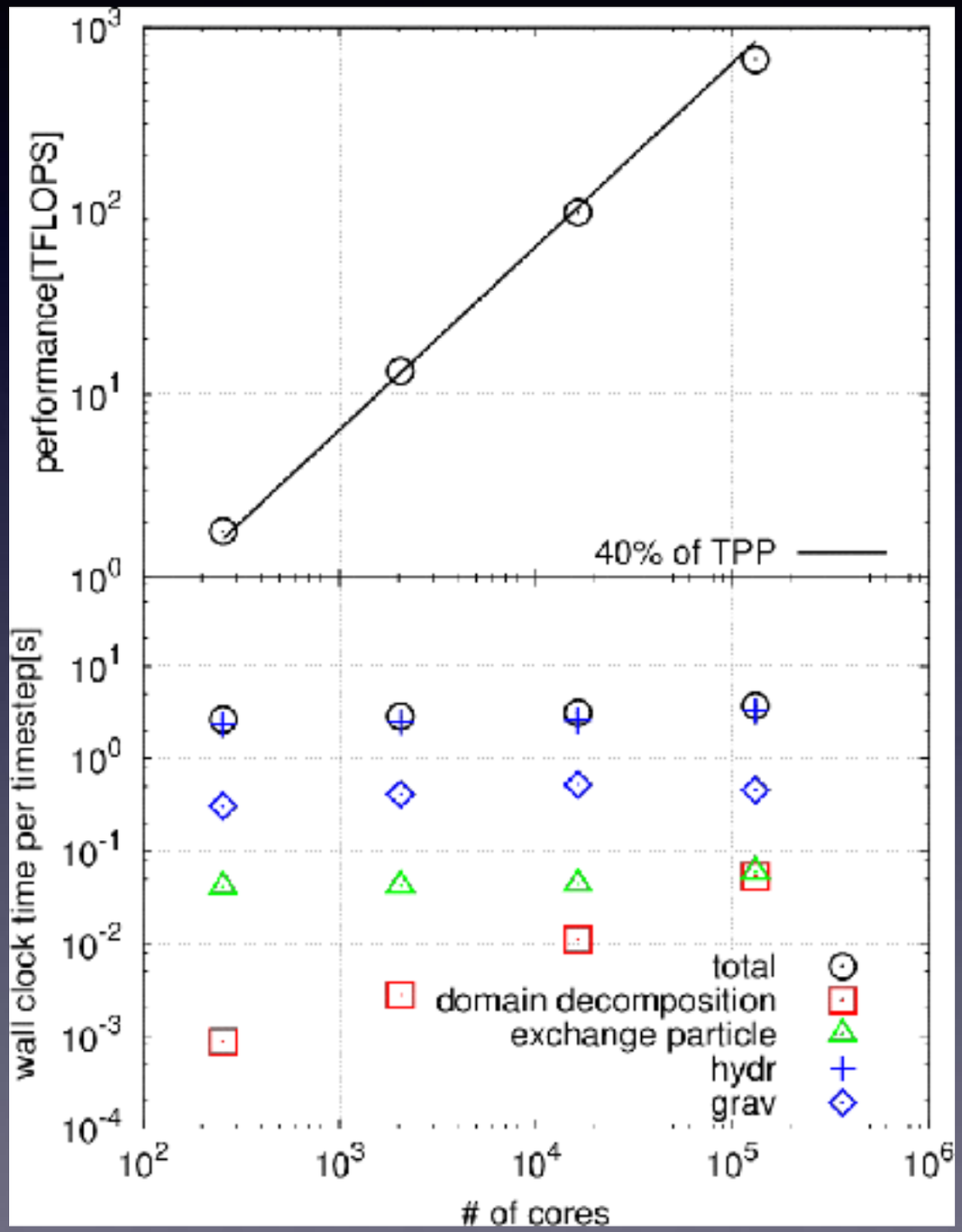


Iwasawa et al. (2016)

性能(SPH)

- 巨大衝突シミュレーション
- 粒子数: 2.0x10^4/core
- 京コンピュータ





Iwasawa et al. (2016)

まとめ

- FDPSは大規模並列粒子シミュレーションコードの開発を支援するフレームワーク
- FDPSのAPIを呼び出すだけで粒子シミュレーションを並列化
- N体コードを200行で記述
- 京コンピュータで理論ピーク性能の40、50%の性能を達成