

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

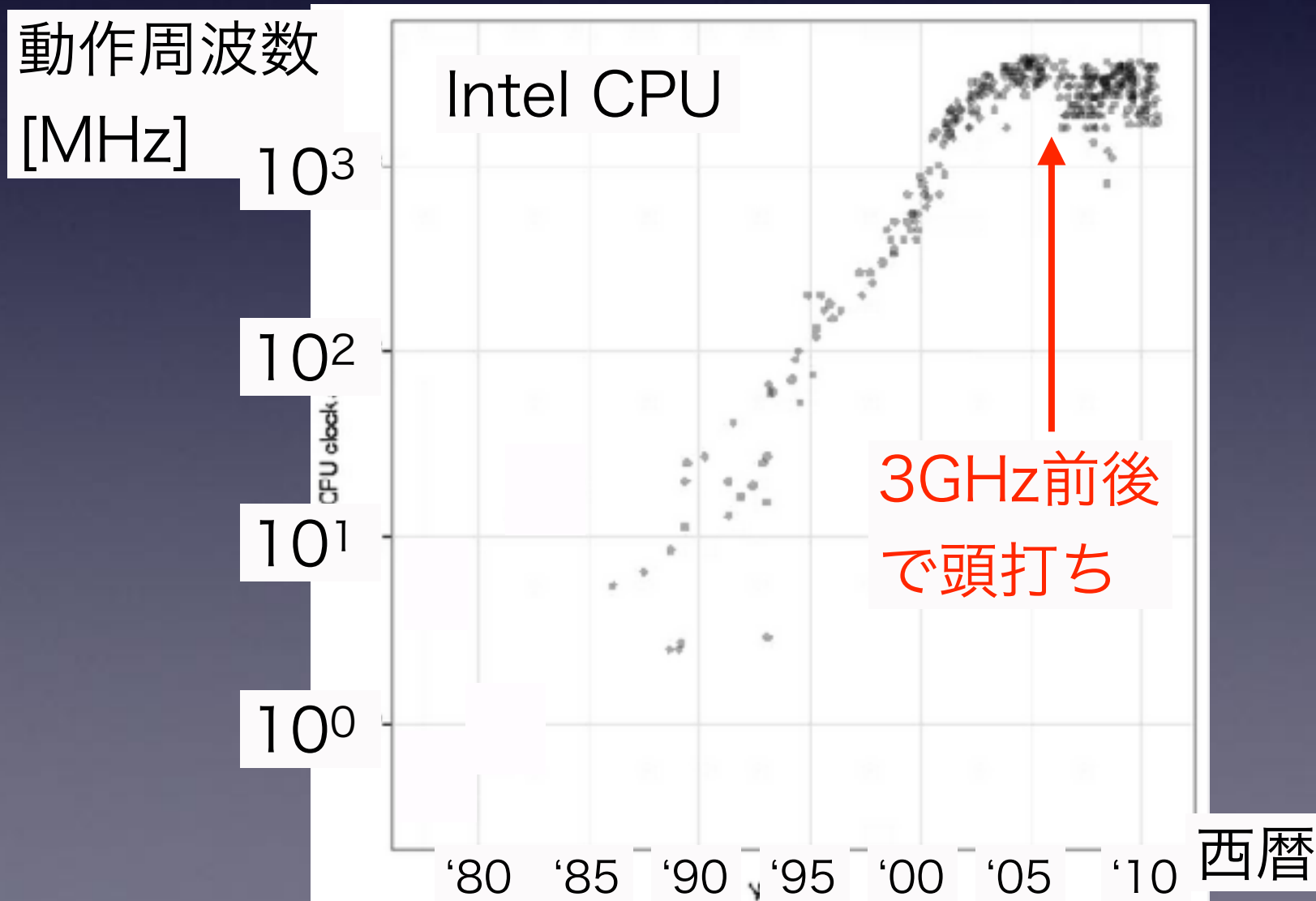
粒子の持つ物理量をその導関数に変換する関数

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

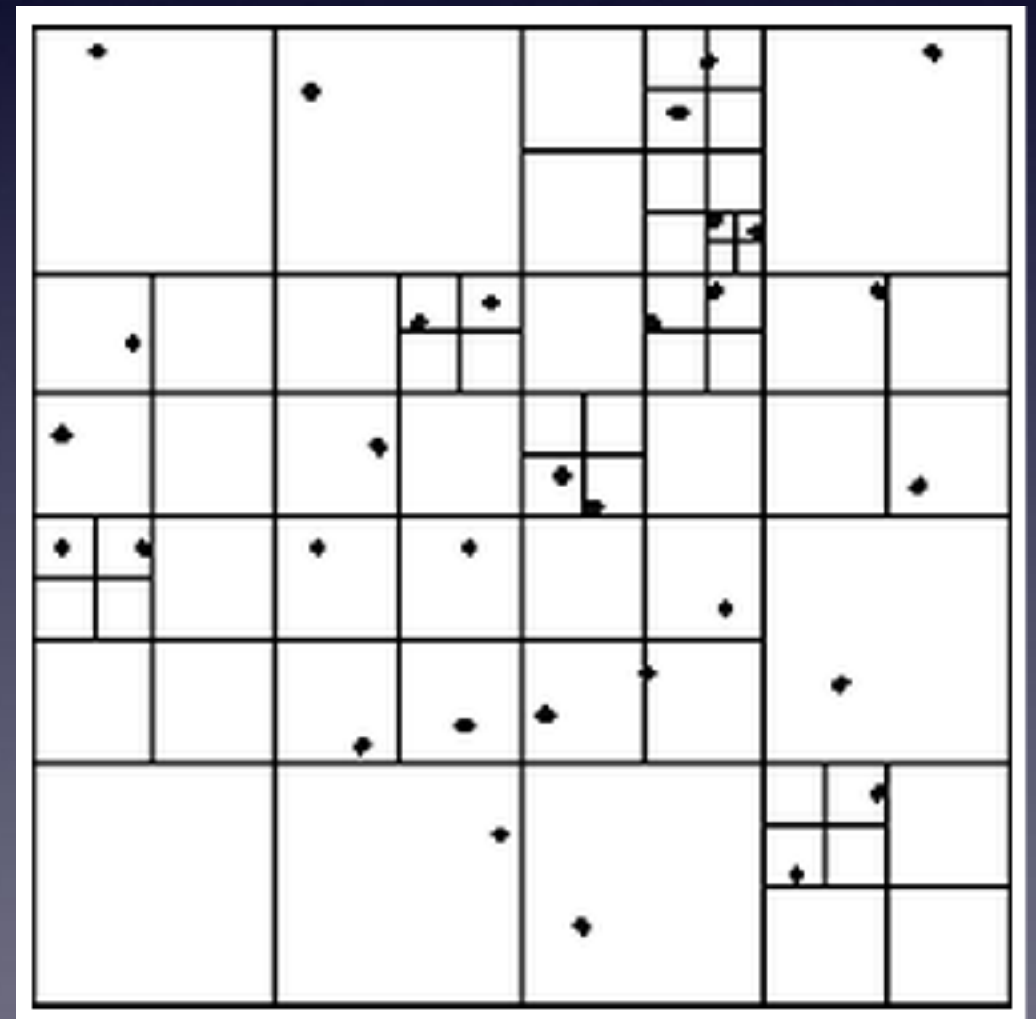
- ・ 分散メモリ環境での並列化
 - ・ 計算領域の分割と粒子データの交換
 - ・ 相互作用計算のための粒子データの交換
- ・ 共有メモリ環境での並列化
 - ・ ツリー構造のマルチウォーク
 - ・ 相互作用計算の負荷分散
- ・ 1 コア内での並列化
 - ・ SIMD演算器の有効利用

実は並列でなくとも、、、

- ・ 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

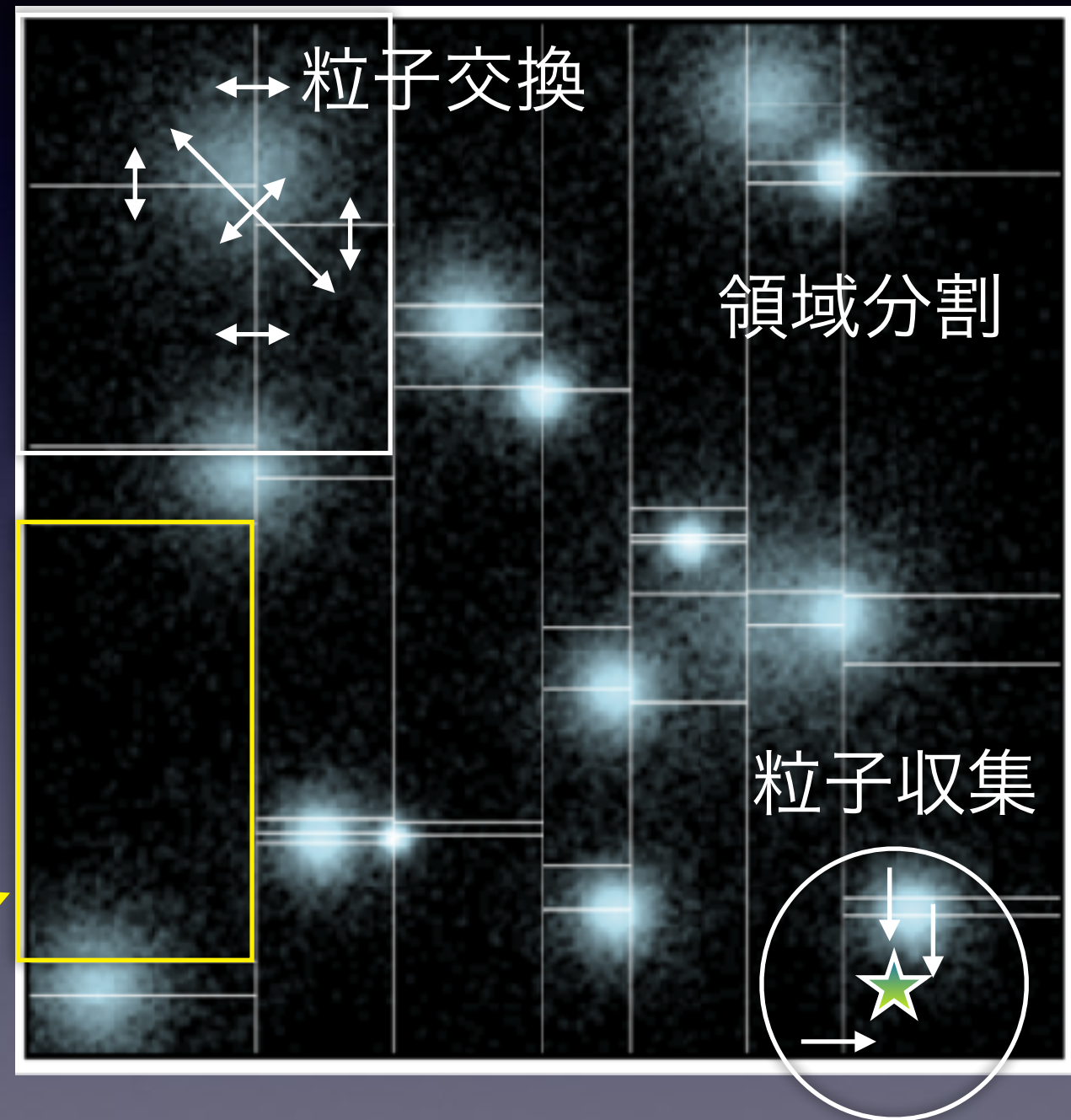


粒子シミュレーションの手順

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)

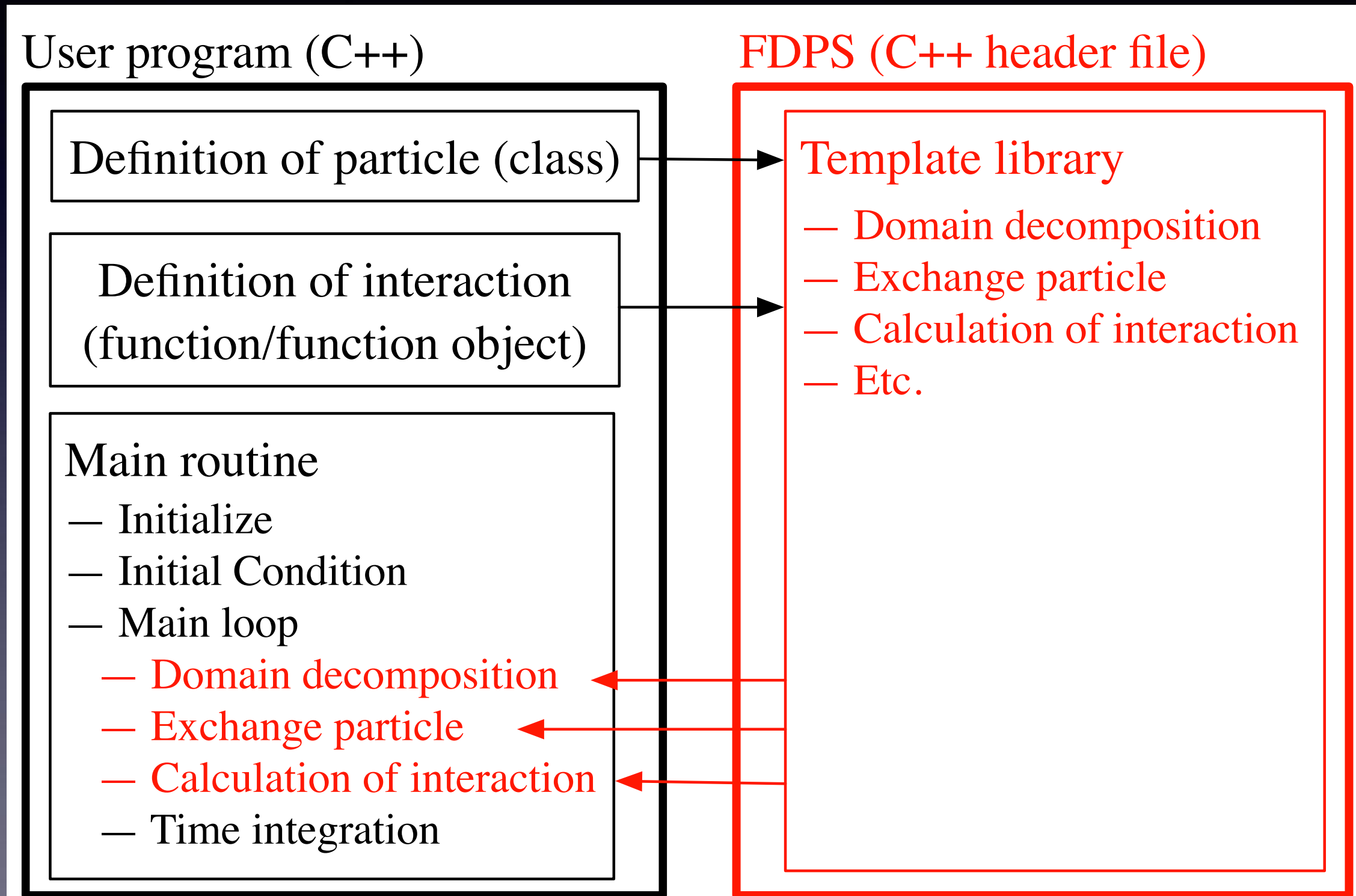
Main routine

- 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)

User program (Fortran)

Definition of particle
(derived data type + FDPS directive)

Definition of interaction (subroutine)

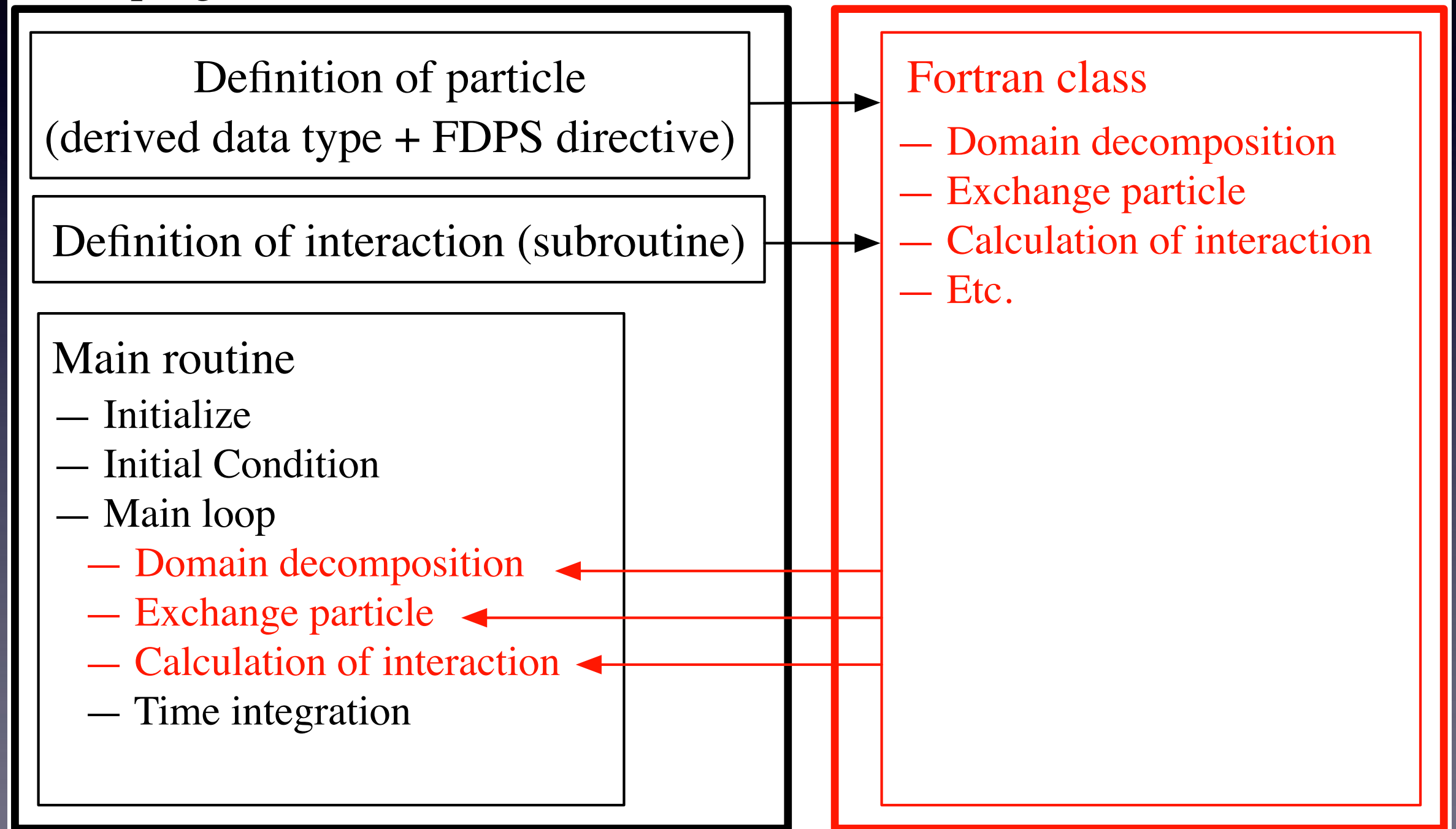
Main routine

- Initialize
- Initial Condition
- Main loop
 - Domain decomposition
 - Exchange particle
 - Calculation of interaction
 - Time integration

FDPS (Fortran module)

Fortran class

- Domain decomposition
- Exchange particle
- Calculation of interaction
- Etc.



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)

FDPDS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！

```
1 module user_defined_types
2   use, intrinsic :: iso_c_binding
3   usefdps_vector
4   usefdps_super_particle
5   implicit none
6
7   type, public, bind(c) :: full_particle!$fdpsFP,EPJ,EPJ,Force
8     !$fdpscopyFromForcefull_particle(pot,pot)(acc,acc)
9     !$fdpscopyFromFPfull_particle(id,id)(mass,mass)(eps,eps)(pos,pos)
10    !$fdpsclearid=keep,mass=keep,eps=keep,pos=keep,vel=keep
11    integer(kind=c_long_long)::i
12    real(kind=c_double)::mass,$fdpscharge
13    real(kind=c_double)::eps
14    type(fdps_f64vec)::pot,$fdpsposition
15    type(fdps_f64vec)::vel,$fdpsvelocity
16    real(kind=c_double)::pot
17    type(fdps_f64vec)::acc
18  endtype full_particle
19
20 contains
21
22  subroutine calc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f)bind(c)
23    integer(c_int),intent(in),value::n_ip,n_jp
24    type(full_particle),dimension(n_ip),intent(in)::ep_i
25    type(full_particle),dimension(n_jp),intent(in)::ep_j
26    type(full_particle),dimension(n_ip),intent(inout)::f
27    integer(c_int)::i,j
28    real(c_double)::eps2,poti,r3_inv,r_inv
29    type(fdps_f64vec)::xi,ai,rij
30
31    do i=1,n_ip
32      eps2=ep_i(i)%eps*ep_i(i)%eps
33      xi=ep_i(i)%pos
34      ai=0.0d0
35      poti=0.0d0
36      do j=1,n_jp
37        rij%x=xi%x-ep_j(j)%pos%x
38        rij%y=xi%y-ep_j(j)%pos%y
39        rij%z=xi%z-ep_j(j)%pos%z
40        r3_inv=rij%x*rij%x&
41          +rij%y*rij%y&
42          +rij%z*rij%z&
43          +eps2
44        r_inv=1.0d0/sqrt(r3_inv)
45        r3_inv=r_inv*r_inv
46        r_inv=r_inv*ep_j(j)%mass
47        r3_inv=r3_inv*r_inv
48        ai%x=ai%x-r3_inv*rij%x
49        ai%y=ai%y-r3_inv*rij%y
50        ai%z=ai%z-r3_inv*rij%z
51        poti=poti-r_inv
52      enddo
53      f(i)%pot=f(i)%pot+poti
54      f(i)%acc=f(i)%acc+ai
55    enddo
56  endsubroutine calc_gravity_pp
57
58  subroutine calc_gravity_psp(ep_i,n_ip,ep_j,n_jp,f)bind(c)
59    integer(c_int),intent(in),value::n_ip,n_jp
60    type(full_particle),dimension(n_ip),intent(in)::ep_i
61    type(fdps_sp_monopole),dimension(n_jp),intent(in)::ep_j
62    type(full_particle),dimension(n_ip),intent(inout)::f
63    integer(c_int)::i,j
64    real(c_double)::eps2,poti,r3_inv,r_inv
65    type(fdps_f64vec)::xi,ai,rij
66
67    do i=1,n_ip
68      eps2=ep_i(i)%eps*ep_i(i)%eps
69      xi=ep_i(i)%pos
70      ai=0.0d0
71      poti=0.0d0
72      do j=1,n_jp
73        rij%x=xi%x-ep_j(j)%pos%x
74        rij%y=xi%y-ep_j(j)%pos%y
75        rij%z=xi%z-ep_j(j)%pos%z
76        r3_inv=rij%x*rij%x&
77          +rij%y*rij%y&
78          +rij%z*rij%z&
79          +eps2
80        r_inv=1.0d0/sqrt(r3_inv)
81        r3_inv=r_inv*r_inv
82        r_inv=r_inv*ep_j(j)%mass
83        r3_inv=r3_inv*r_inv
84        ai%x=ai%x-r3_inv*rij%x
85        ai%y=ai%y-r3_inv*rij%y
86        ai%z=ai%z-r3_inv*rij%z
87        poti=poti-r_inv
88      enddo
89      f(i)%pot=f(i)%pot+poti
90      f(i)%acc=f(i)%acc+ai
91    enddo
92  endsubroutine calc_gravity_psp
93
94 endmodule user_defined_types
```

```
1 subroutine main()
2   usefdps_module
3   useuser_defined_types
4   implicit none
5   doubleprecision,parameter::time_end=10.0d0
6   doubleprecision,parameter::dt=1.0d0/128.0d0
7   integer::i,j,k,ierr
8   integer::psys_num,dinfo_num,tree_num
9   character(len=64)::tree_type
10  doubleprecision::time_sys=0.0d0
11  type(fdps_controller)::fdps_ctrl
12  callfdps_ctrl%PS_initialize()
13  callfdps_ctrl%create_dinfo(dinfo_num)
14  callfdps_ctrl%init_dinfo(dinfo_num)
15  callfdps_ctrl%create_psys(psys_num,'full_particle')
16  callfdps_ctrl%init_psys(psys_num)
17  tree_type='Long,full_particle,full_particle,full_particle,Monopole'
18  callfdps_ctrl%create_tree(tree_num,tree_type)
19  callfdps_ctrl%init_tree(tree_num,0)
20  callread_IC(fdps_ctrl,psys_num)
21  callcalc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
22  do
23    callkick(fdps_ctrl,psys_num,0.5d0*dt)
24    time_sys=time_sys+dt
25    calldrift(fdps_ctrl,psys_num,dt)
26    callcalc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
27    callkick(fdps_ctrl,psys_num,0.5d0*dt)
28    if(time_sys>=time_end)exit
29  enddo
30  callfdps_ctrl%PS_finalize()
31 endsubroutine main
32
33 subroutine calc_gravity(fdps_ctrl,psys_num,dinfo_num,tree_num)
34   usefdps_module
35   useuser_defined_types
36   implicit none
37   type(fdps_controller),intent(IN)::fdps_ctrl
38   integer,intent(IN)::psys_num,dinfo_num,tree_num
39   type(c_funptr)::pfunc_ep_ep,pfunc_ep_sp
40   callfdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
41   callfdps_ctrl%exchange_particle(psys_num,dinfo_num)
42   pfunc_ep_ep=c_funloc(calc_gravity_pp)
43   pfunc_ep_sp=c_funloc(calc_gravity_psp)
44   callfdps_ctrl%calc_force_all_and_write_back(tree_num,&
45     pfunc_ep_ep,&
46     pfunc_ep_sp,&
47     psys_num,&
48     dinfo_num)
49 endsubroutine calc_gravity
50
51 subroutine kick(fdps_ctrl,psys_num,dt)
52   usefdps_vector
53   usefdps_module
54   useuser_defined_types
55   implicit none
56   type(fdps_controller),intent(IN)::fdps_ctrl
57   integer,intent(IN)::psys_num
58   doubleprecision,intent(IN)::dt
59   integer::i,nptcl_loc
60   type(full_particle),dimension(:),pointer::ptcl
61   nptcl_loc=fdps_ctrl%get_nptcl_loc(psys_num)
62   callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
63   do i=1,nptcl_loc
64     ptcl(i)%vel=ptcl(i)%vel+ptcl(i)%acc*dt
65   enddo
66   nullify(ptcl)
67 endsubroutine kick
68
69 subroutine drift(fdps_ctrl,psys_num,dt)
70   usefdps_vector
71   usefdps_module
72   useuser_defined_types
73   implicit none
74   type(fdps_controller),intent(IN)::fdps_ctrl
75   integer,intent(IN)::psys_num
76   doubleprecision,intent(IN)::dt
77   integer::i,nptcl_loc
78   type(full_particle),dimension(:),pointer::ptcl
79   nptcl_loc=fdps_ctrl%get_nptcl_loc(psys_num)
80   callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
81   do i=1,nptcl_loc
82     ptcl(i)%pos=ptcl(i)%pos+ptcl(i)%vel*dt
83   enddo
84   nullify(ptcl)
85 endsubroutine drift
86
87 subroutine read_IC(fdps_ctrl,psys_num)
88   usefdps_module
89   useuser_defined_types
90   implicit none
91   type(fdps_controller),intent(IN)::fdps_ctrl
92   integer,intent(IN)::psys_num
93   character(len=16),parameter::root_dir='input_data'
94   character(len=16),parameter::file_prefix='proc'
95   integer::i,myrank,nptcl_loc
96   character(len=64)::fname,proc_num
97   type(full_particle),dimension(:),pointer::ptcl
98   myrank=fdps_ctrl%get_rank()
99   write(proc_num,"(i5.5)")myrank
100  fname=trim(root_dir)//"/"&
101    //trim(file_prefix)//proc_num//".dat"
102  open(unit=9,file=trim(fname),action='read',form='unformatted',&
103    access='stream',status='old')
104  read(9)nptcl_loc
105  callfdps_ctrl%set_nptcl_loc(psys_num,nptcl_loc)
106  callfdps_ctrl%get_psys_fptr(psys_num,ptcl)
107  do i=1,nptcl_loc
108    read(9)ptcl(i)%id,ptcl(i)%mass,ptcl(i)%eps,&
109      ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z,&
110      ptcl(i)%vel%x,ptcl(i)%vel%y,ptcl(i)%vel%z
111  enddo
112  close(unit=9)
113  nullify(ptcl)
114 endsubroutine read_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 template code which can be actually compiled and run, not only on a single-processor 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

```
1 #include <particle_simulator.hpp>
2 using namespace PS;
3
4 class Nbody{
5 public:
6     F64 mass, xps;
7     F64vec pos, vel, acc;
8     F64vec getPos() const {return pos;}
9     F64 getCharge() const {return mass;}
10    void copyFromFP(const Nbody &in){
11        mass = in.mass;
12        pos = in.pos;
13        xps = in.xps;
14    }
15    void copyFromForce(const Nbody &out) {
16        acc = out.acc;
17    }
18    void clear() {
19        acc = 0.0;
20    }
21    void readAscii(FILE *fp) {
22        fscanf(fp,
23            "%lf%lf%lf%lf%lf%lf%lf%lf",
24            &mass, &xps,
25            &pos.x, &pos.y, &pos.z,
26            &vel.x, &vel.y, &vel.z);
27    }
28    void predict(F64 dt) {
29        vel += (0.5 * dt) * acc;
30        pos += dt * vel;
31    }
32    void correct(F64 dt) {
33        vel += (0.5 * dt) * acc;
34    }
35 };
36
```

```
37 template <class TPos>
38 struct CalcGrav{
39     void operator () (const Nbody * ip,
40         const S32 ni,
41         const TPos * jp,
42         const S32 nj,
43         Nbody * force) {
44         for(S32 i=0; i<ni; i++){
45             F64vec xi = ip[i].pos;
46             F64 wp2 = ip[i].xps
47                 * ip[i].xps;
48             F64vec ax = 0.0;
49             for(S32 j=0; j<nj; j++){
50                 F64vec xj = jp[j].pos;
51                 F64vec dr = xi - xj;
52                 F64 r2 = dr.x*dr.x + dr.y*dr.y + dr.z*dr.z;
53                 F64 dr2 = dr * dr + wp2;
54                 F64 dxi = 1.0 / sqrt(dr2);
55                 xi -= (dr2 * dxi * dr);
56             }
57         }
58     }
59 };
60
```

```
61     * mj) * dr;
62     }
63     force[i].acc += xi;
64 }
65 }
66
67 template<class Tpos>
68 void predict(Tpos &p,
69     const F64 dt) {
70     S32 n = p.getNumberOfParticleLocal();
71     for(S32 i = 0; i < n; i++)
72         p[i].predict(dt);
73 }
74
```

```
75 template<class Tpos>
76 void correct(Tpos &p,
77     const F64 dt) {
78     S32 n = p.getNumberOfParticleLocal();
79     for(S32 i = 0; i < n; i++)
80         p[i].correct(dt);
81 }
82
83 template <class TDI, class TFS, class TTF>
84 void calcGravAllAndWriteBack(TDI &dinfo,
85     TFS &ptcl,
86     TTF &tree) {
87     dinfo.decomposeDomainAll(ptcl);
88     ptcl.exchangeParticle(dinfo);
89     tree.calcForceAllAndWriteBack
90         (CalcGrav<Nbody>(),
91         CalcGrav<SP, Monopole>(),
92         ptcl, dinfo);
93 }
94
```

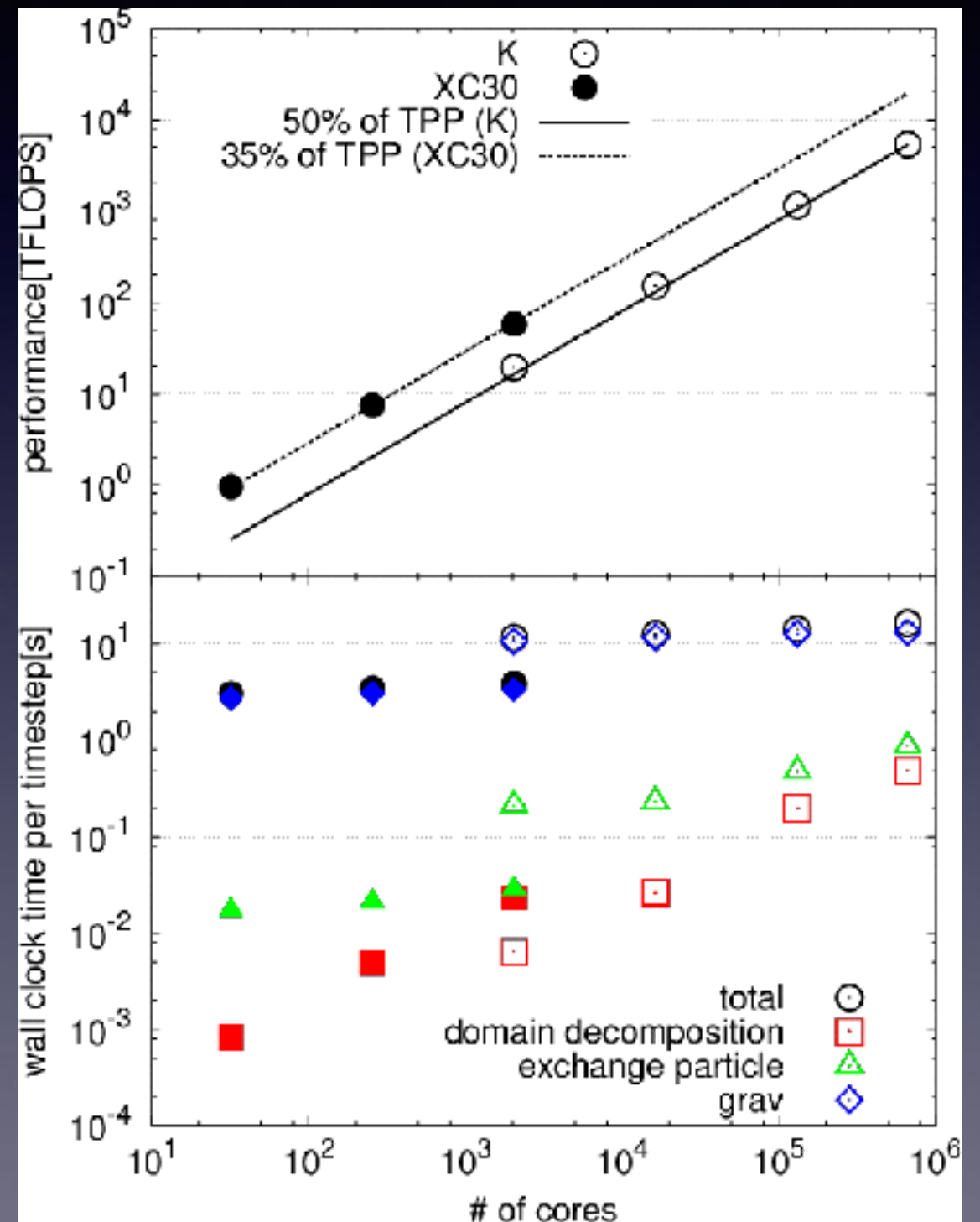
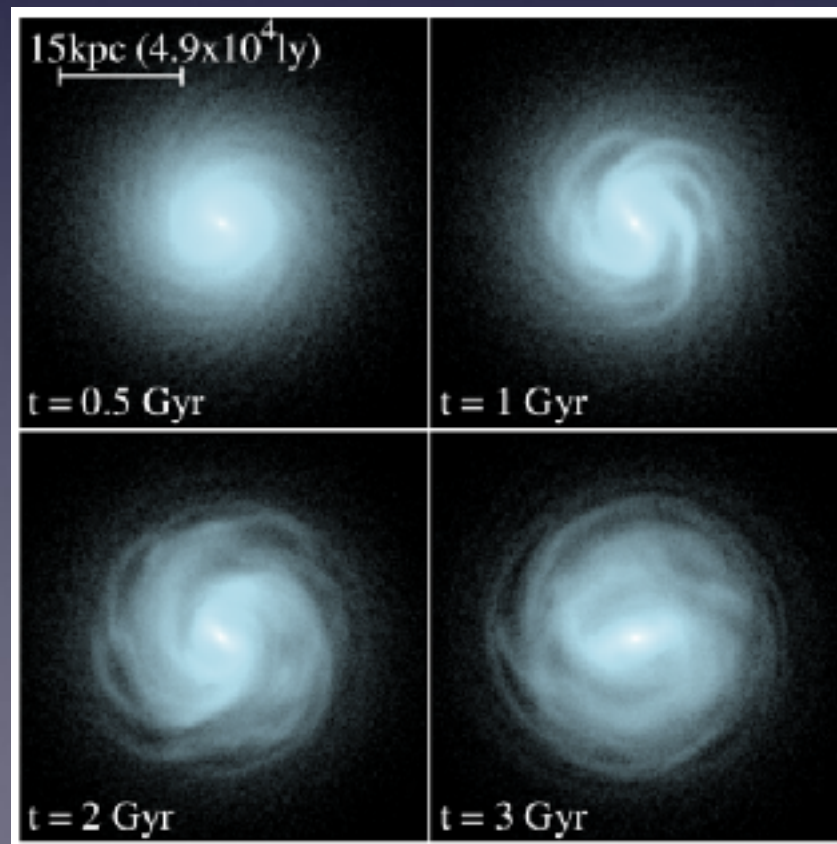
```
95 int main(int argc, char *argv[]) {
96     F64 time = 0.0;
97     const F64 tend = 10.0;
98     const F64 dtime = 1.0 / 128.0;
99     PS::Initialize(argc, argv);
100     PS::DomainInfo dinfo;
101     dinfo.initialize();
102     PS::ParticleSystem<Nbody> ptcl;
103     ptcl.initialize();
104     PS::TreeForForceLong<Nbody, Nbody,
105         Nbody>::Monopole grav;
106     grav.initialize(0);
107     ptcl.readParticleAscii(argv[1]);
108     calcGravAllAndWriteBack(dinfo,
109         ptcl,
110         grav);
111     while(time < tend) {
112         predict(ptcl, dtime);
113         calcGravAllAndWriteBack(dinfo,
114             ptcl,
115             grav);
116         correct(ptcl, dtime);
117         time += dtime;
118     }
119     PS::Finalize();
120     return 0;
121 }
```

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)

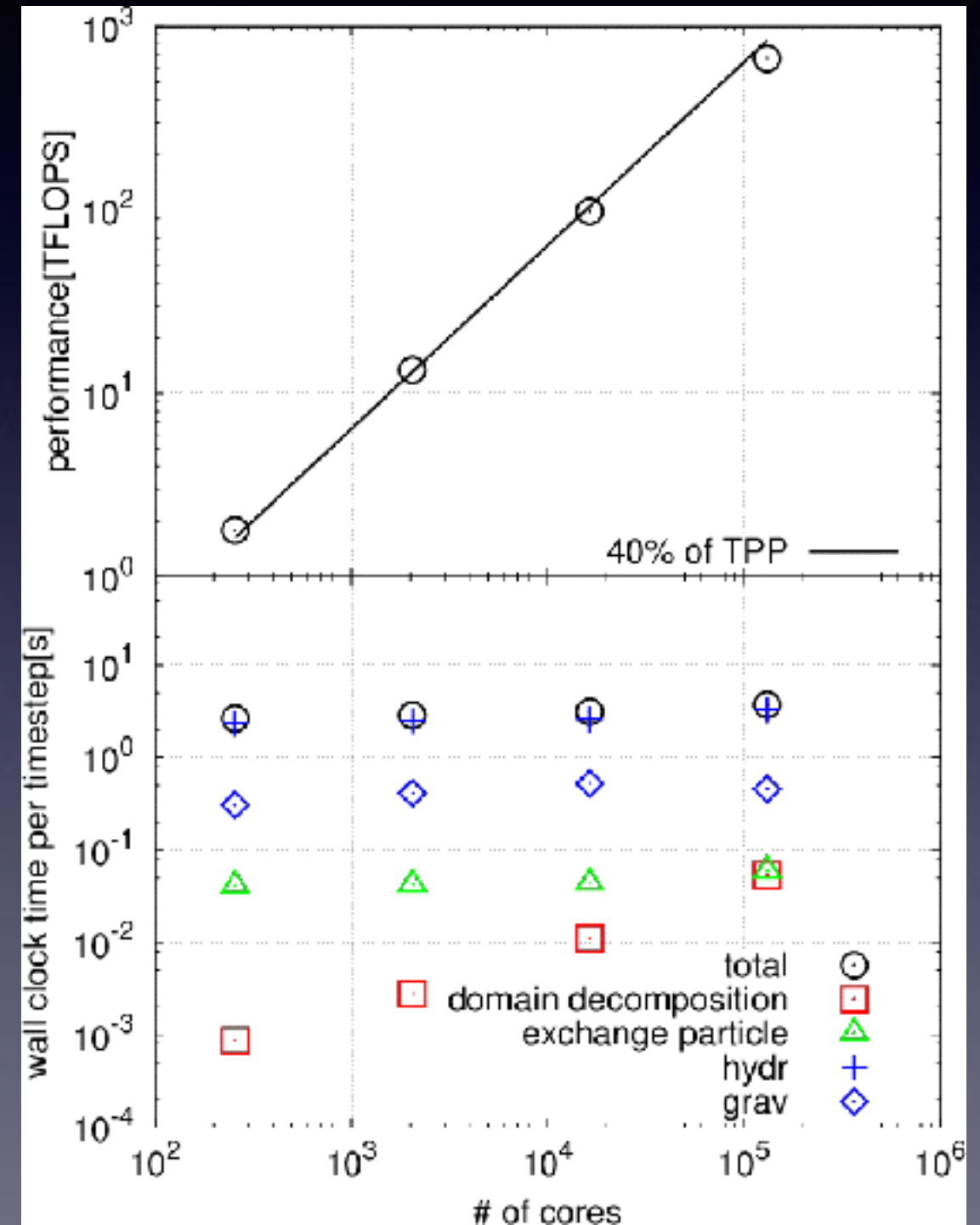
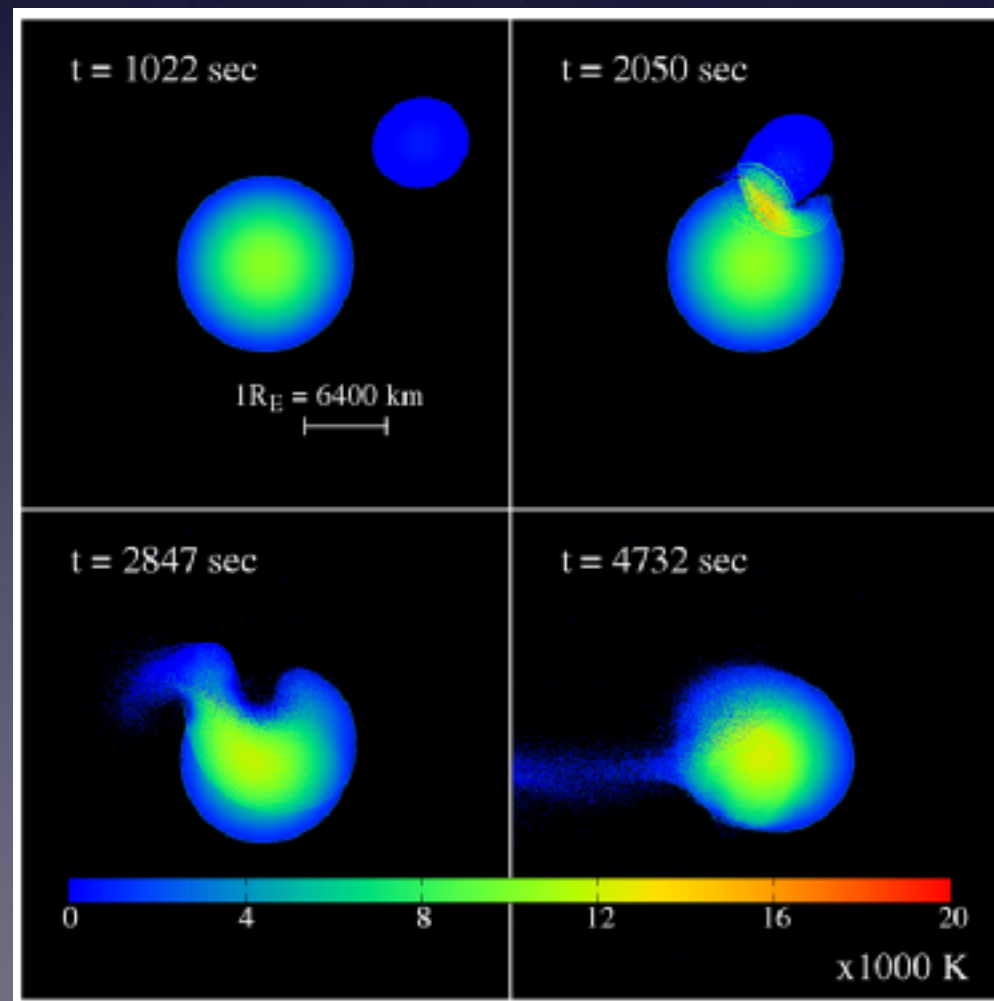
- ・ 円盤銀河
- ・ 粒子数: $2.7 \times 10^5 / \text{core}$
- ・ 精度: $\Theta = 0.4$ 四重極
- ・ 京コンピュータ, XC30



Iwasawa et al. (2016)

性能(SPH)

- ・ 巨大衝突シミュレーション
- ・ 粒子数: $2.0 \times 10^4/\text{core}$
- ・ 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