

# FDPSの概要説明

岩澤全規

松江工業高等専門学校

神戸大学理学研究科

理化学研究所計算科学研究センター

2020/08/19 FDPS講習会

# FDPSとは

- Framework for Developing Particle Simulator
- 大規模並列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

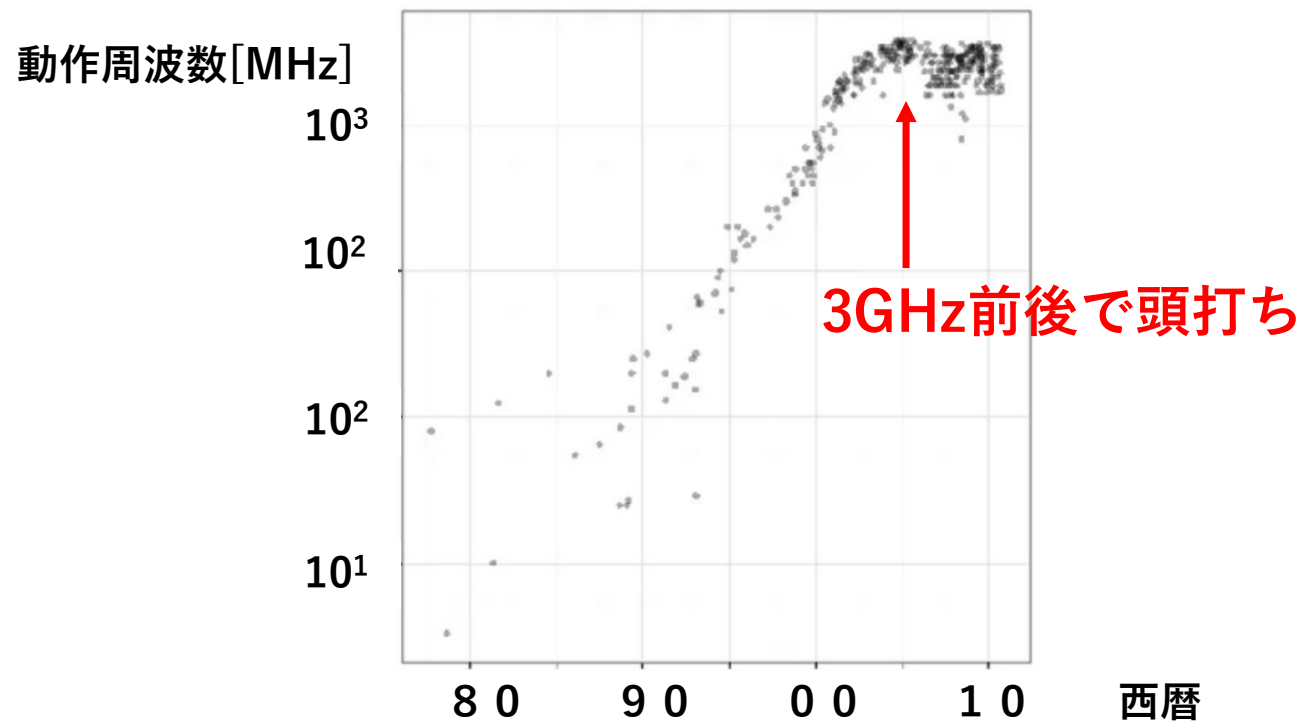
粒子間相互作用を表す関数

粒子の持つ物理量をその導関数に変換する関数

The diagram shows the governing equation for particle simulation:  $\frac{d\vec{u}_i}{dt} = \vec{g} \left( \sum_j^N \vec{f}(\vec{u}_i, \vec{u}_j), \vec{u}_i \right)$ . Red circles highlight the terms  $\vec{g}$ ,  $\vec{f}$ , and  $\vec{u}_i$  within the summation. Red arrows point from Japanese text labels to these circled terms: '粒子データのベクトル' (Vector of particle data) points to  $\vec{u}_i$ ; '粒子間相互作用を表す関数' (Function representing particle-particle interaction) points to  $\vec{f}$ ; and '粒子の持つ物理量をその導関数に変換する関数' (Function that converts the physical quantity of the particle into its derivative) points to  $\vec{g}$ .

# 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

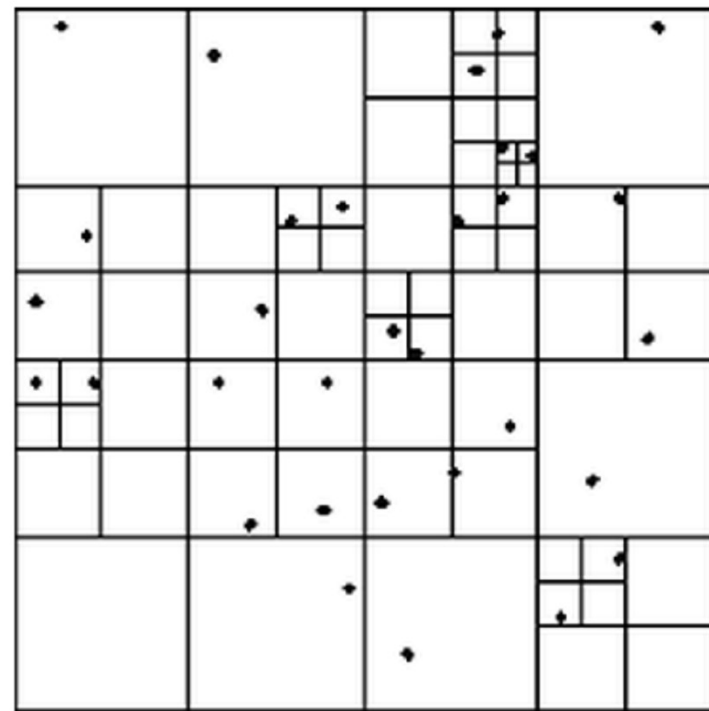
- 分散メモリ環境での並列化
  - 計算領域の分割と粒子データの交換
  - 相互作用計算のための粒子データの交換
- 共有メモリ環境での並列化
  - ツリー構造のマルチウォーク
  - 相互作用計算の負荷分散
- 1コア内での並列化
  - SIMD演算器の有効利用

実は並列でなくとも、、、

- キャッシュメモリーの有効利用
- ツリー構造の構築

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

**Nを小さい数に減らす方法**



# FDPSで困難を解決

- 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
- キャッシュメモリの有効利用
- Tree構造による粒子分布の管理
- 1コア内での並列化
  - SIMD演算器の有効利用

**FDPS**

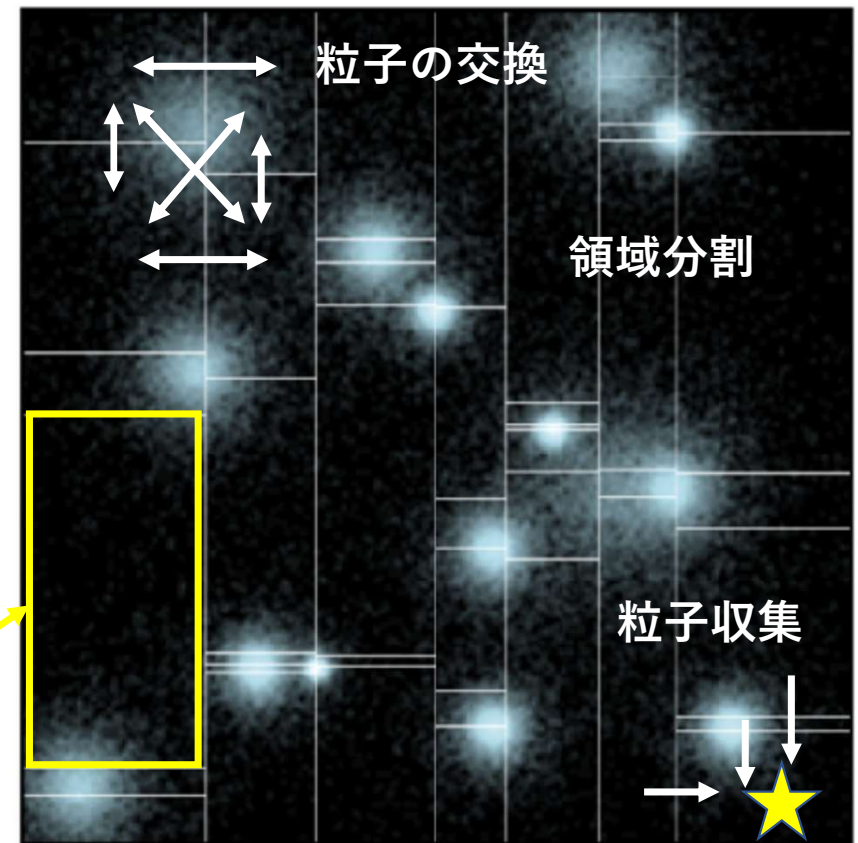
**PIKG (野村さん講義)**

# 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
    - 上のクラスや相互作用関数を受け取るためにテンプレートを利用
    - コンパイル時に静的にコード生成するため

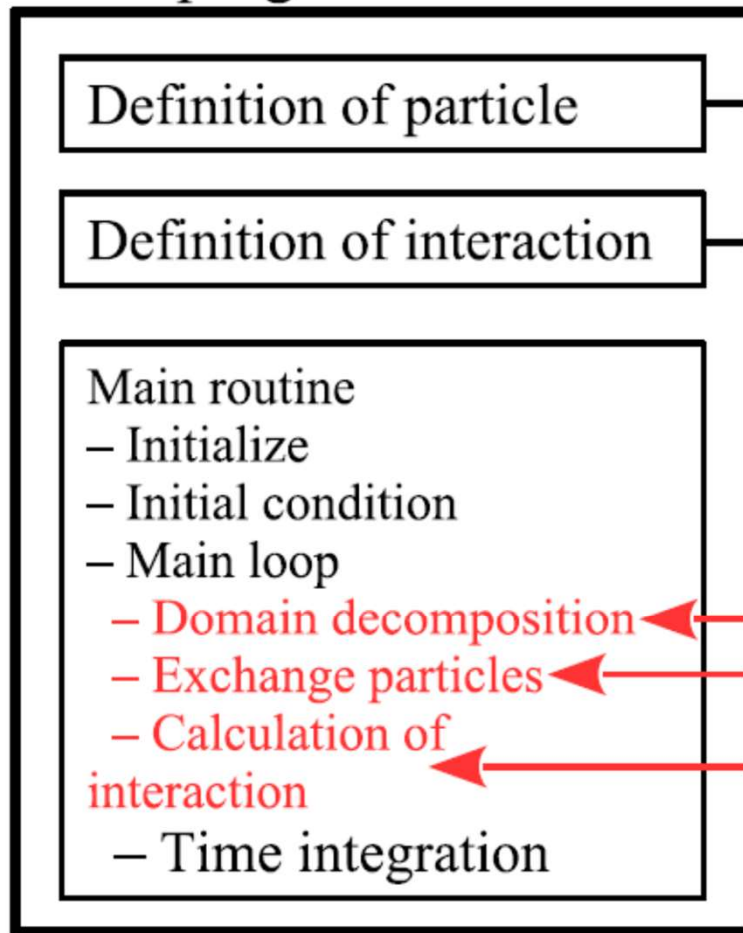


## FDPSの実装方針(2)

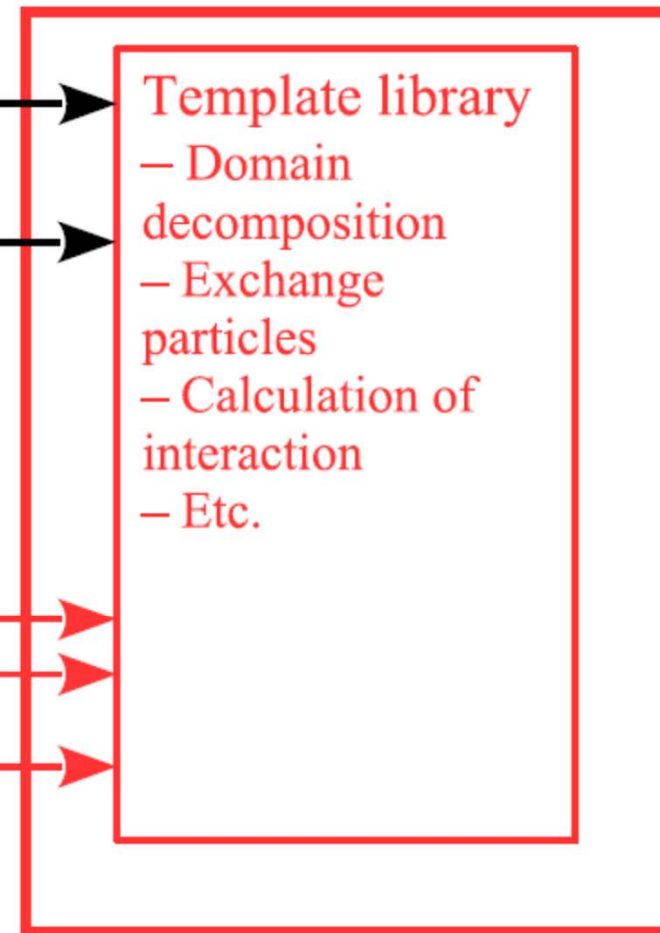
- 並列化
  - 分散メモリ環境(ノード間) : MPI
  - 共有メモリ環境(ノード内) : OpenMP

# FDPSの基本設計

User program

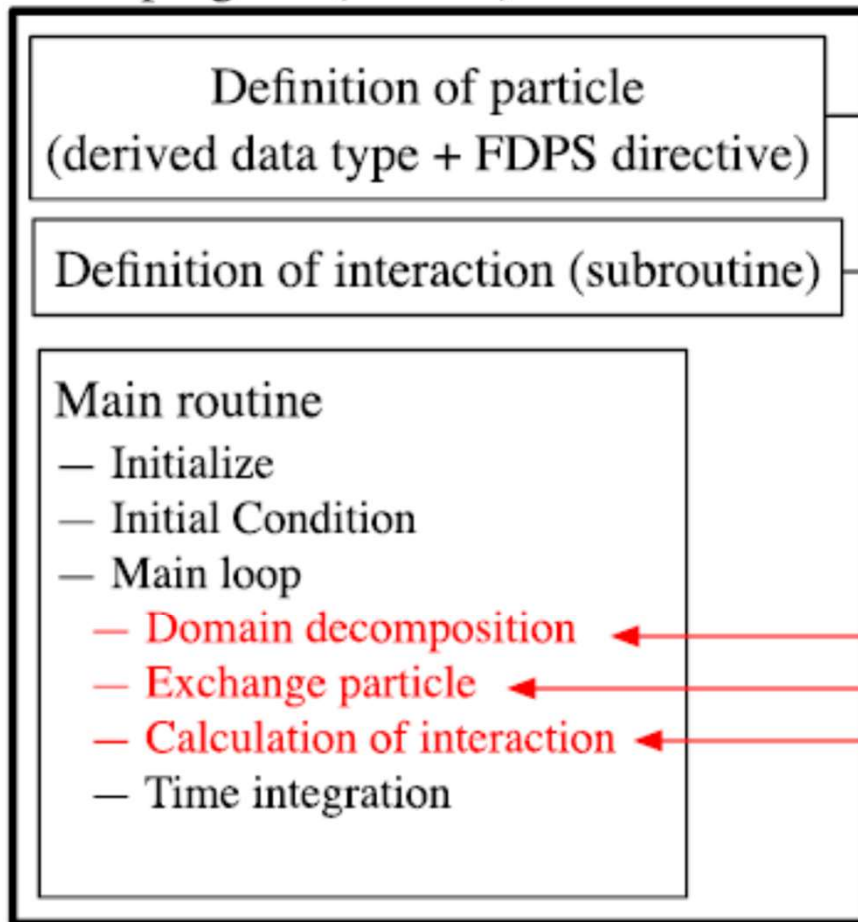


FDPS

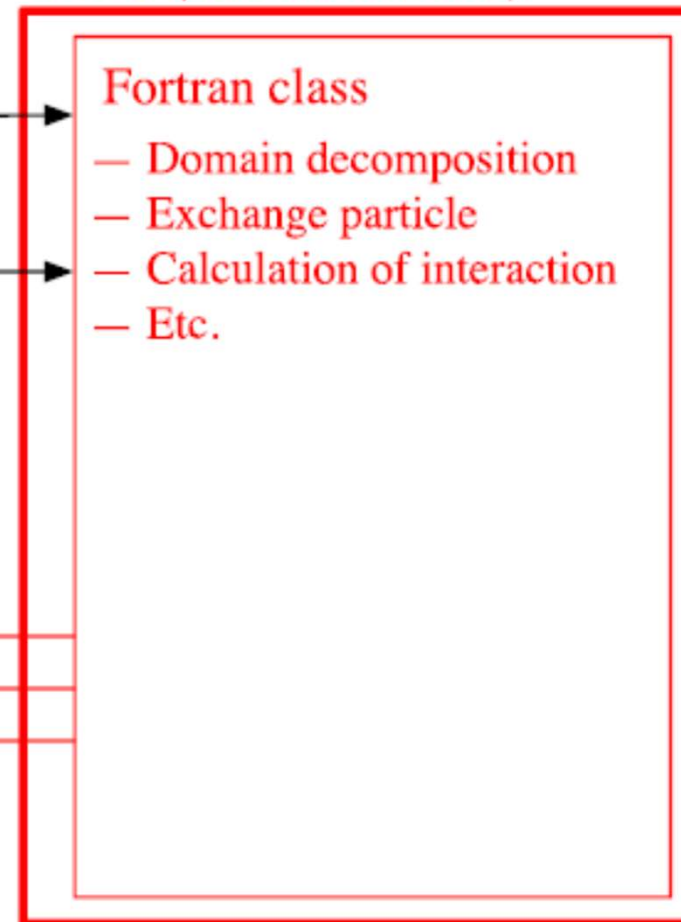


# 基本設計(Fortranの場合)

## User program (Fortran)



## FDPS (Fortran module)



# 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法の実装
- 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
- 2018年11月 FDPS Ver.5.0
  - C Interfaceの実装
- 2020年8月 FDPS Ver.6.0
  - PIKGの実装

# FDPSを使ったプログラム例

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.

```
1 #include <particle_simulator.hpp>
2 using namespace PS;
3
4 class Nbody{
5 public:
6     F64 mass, eps;
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        eps = in.eps;
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, &eps,
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 TPJ>
38 struct CalcGrav{
39     void operator () (const Nbody * ip,
40         const S32 ni,
41         const TPJ * jp,
42         const S32 nj,
43         Nbody * force) {
44         for(S32 i=0; i<ni; i++){
45             F64vec x1 = ip[i].pos;
46             F64 ep2 = ip[i].eps
47                 * ip[i].eps;
48             F64vec a1 = 0.0;
49             for(S32 j=0; j<nj; j++){
50                 F64vec xj = jp[j].pos;
51                 F64vec dr = x1 - xj;
52                 F64 mj = jp[j].mass;
53                 F64 dr2 = dr * dr + ep2;
54                 F64 dri = 1.0 / sqrt(dr2);
55                 a1 += (dri * dri * dri
```

```
56                 * mj) * dr;
57             force[i].acc += a1;
58         }
59     }
60 };
61
62 template<class Tpsys>
63 void predict(Tpsys &p,
64     const F64 dt) {
65     S32 n = p.getNumberOfParticleLocal();
66     for(S32 i = 0; i < n; i++){
67         p[i].predict(dt);
68     }
69 }
70
71 template<class Tpsys>
72 void correct(Tpsys &p,
73     const F64 dt) {
74     S32 n = p.getNumberOfParticleLocal();
75     for(S32 i = 0; i < n; i++){
76         p[i].correct(dt);
77     }
78 }
79
80 template <class TDI, class TPS, class TTF>
81 void calcGravAllAndWriteBack(TDI &dinfo,
82     TPS &ptcl,
83     TTF &tree) {
84     dinfo.decomposeDomainAll(ptcl);
85     ptcl.exchangeParticle(dinfo);
86     tree.calcForceAllAndWriteBack
87         (CalcGrav<Nbody>(),
88         CalcGrav<SPJMonopole>(),
89         ptcl, dinfo);
90 }
91
92 int main(int argc, char *argv[]) {
93     F32 time = 0.0;
94     const F32 tend = 100.0;
95     const F32 dtime = 1.0 / 128.0;
96     PS::Initialize(argc, argv);
97     PS::DomainInfo dinfo;
98     dinfo.initialize();
99     PS::ParticleSystem<Nbody> ptcl;
100    ptcl.initialize();
101    PS::TreeForForceLong<Nbody, Nbody,
102        Nbody>::Monopole grav;
103    grav.initialize(0);
104    ptcl.readParticleAscii(argv[1]);
105    calcGravAllAndWriteBack(dinfo,
106        ptcl,
107        grav);
108    while(time < tend) {
109        predict(ptcl, dtime);
110        calcGravAllAndWriteBack(dinfo,
111            ptcl,
112            grav);
113        correct(ptcl, dtime);
114        time += dtime;
115    }
116    PS::Finalize();
117    return 0;
118 }
```

FDPSのインストール(ヘッダー  
ファイルのインクルード)

粒子クラスの実装

相互作用関数の定義

メインルーチン

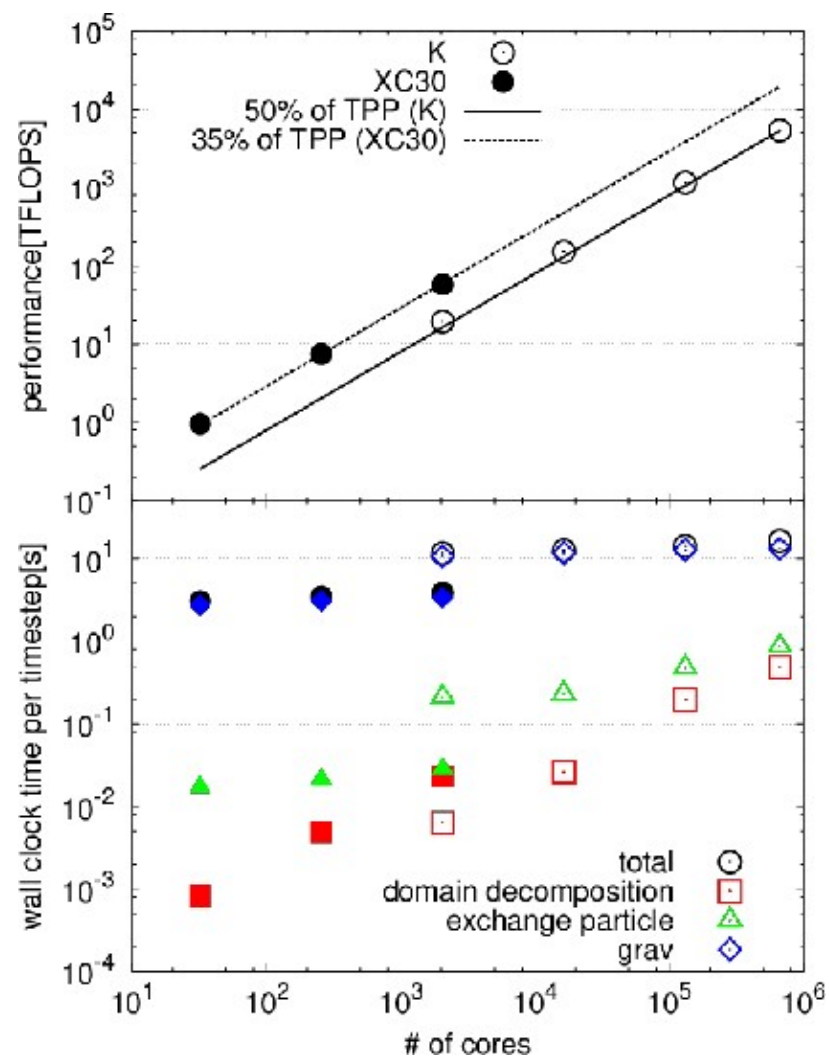
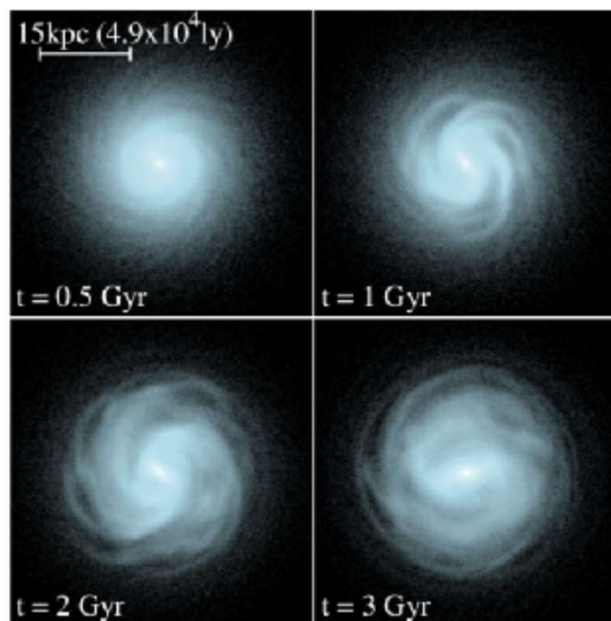
大規模並列N体コードが117行で書ける

# 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を計算
    - チューニングが必要
- ※PIKGを使えば最適化された相互作用関数を生成してくれる。

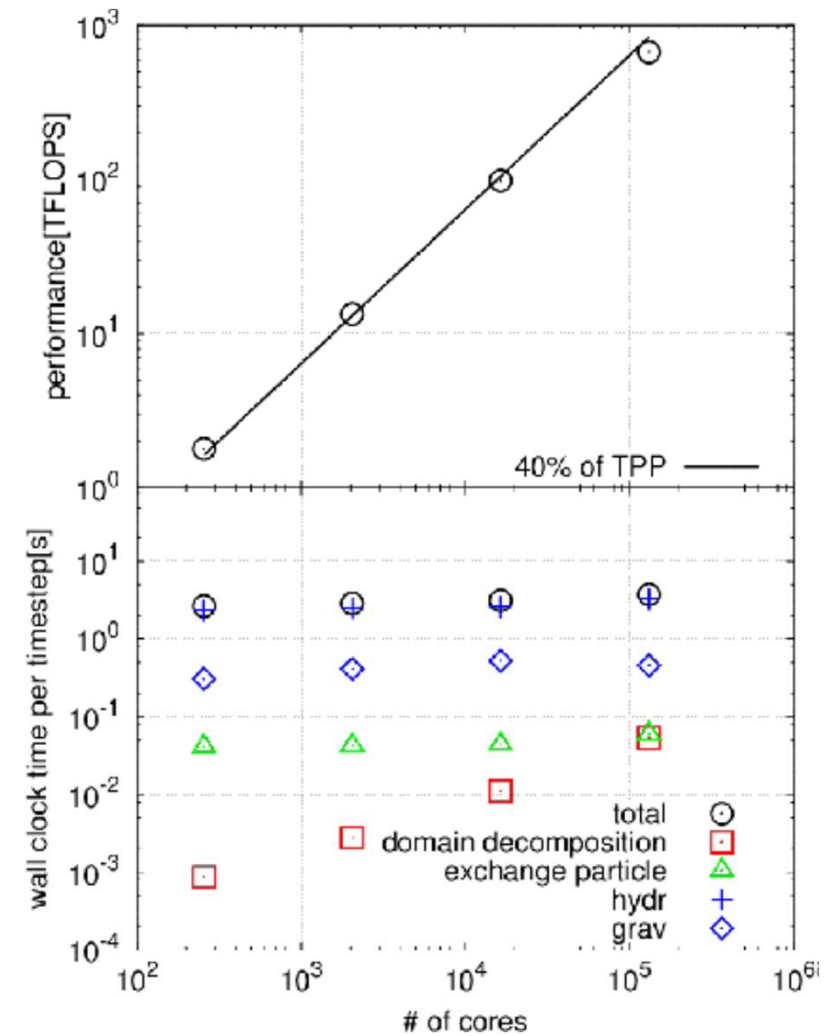
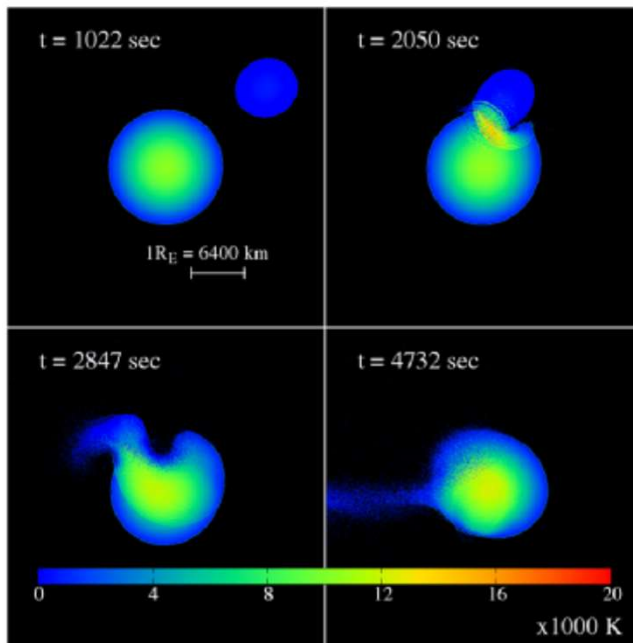
# 性能(N体)

- 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



# 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





## 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100行程度で記述
- 京コンピュータで理論ピーク性能の40、50%の性能を達成