FDPSの概要説明

谷川衝
理化学研究所 計算科学研究機構
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
粒子系シミュレータ開発チーム

2015/07/22 AICS/FOCUS 共催 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

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

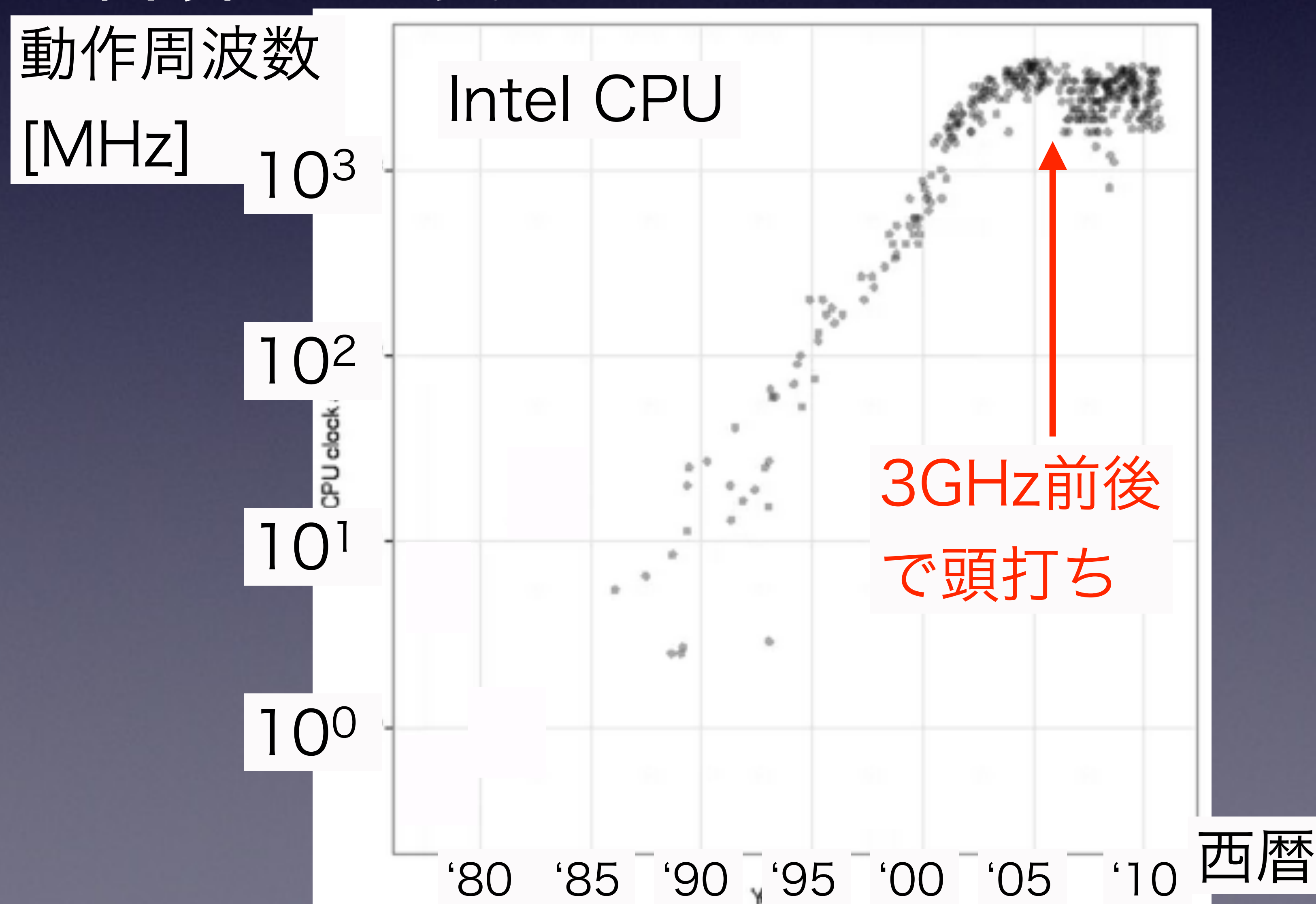
粒子データのベクトル

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

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

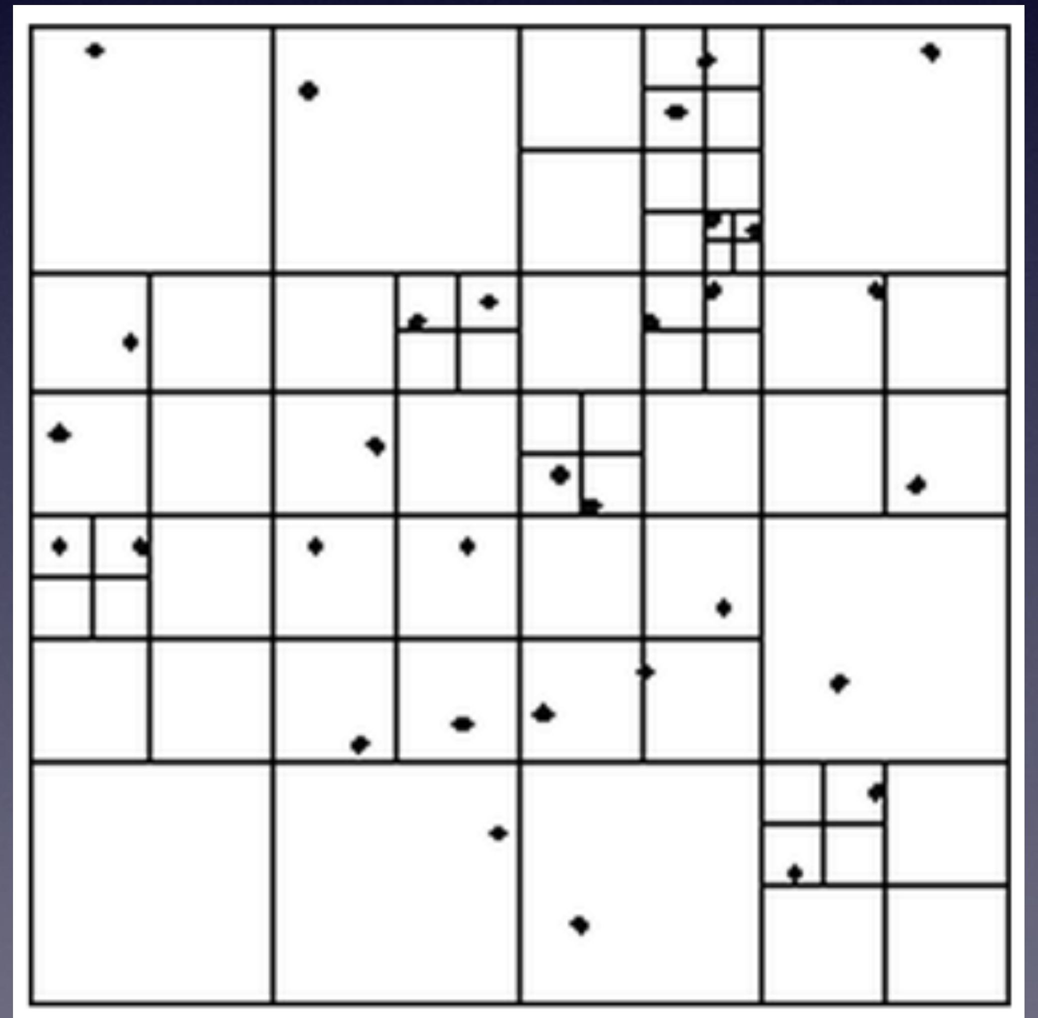
- 分散メモリ環境での並列化
  - 計算領域の分割と粒子データの交換
  - 相互作用計算のための粒子データの交換
- 共有メモリ環境での並列化
  - ツリー構造のマルチウォーク
  - 相互作用計算の負荷分散
- １コア内での並列化
  - SIMD演算器の有効利用

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

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

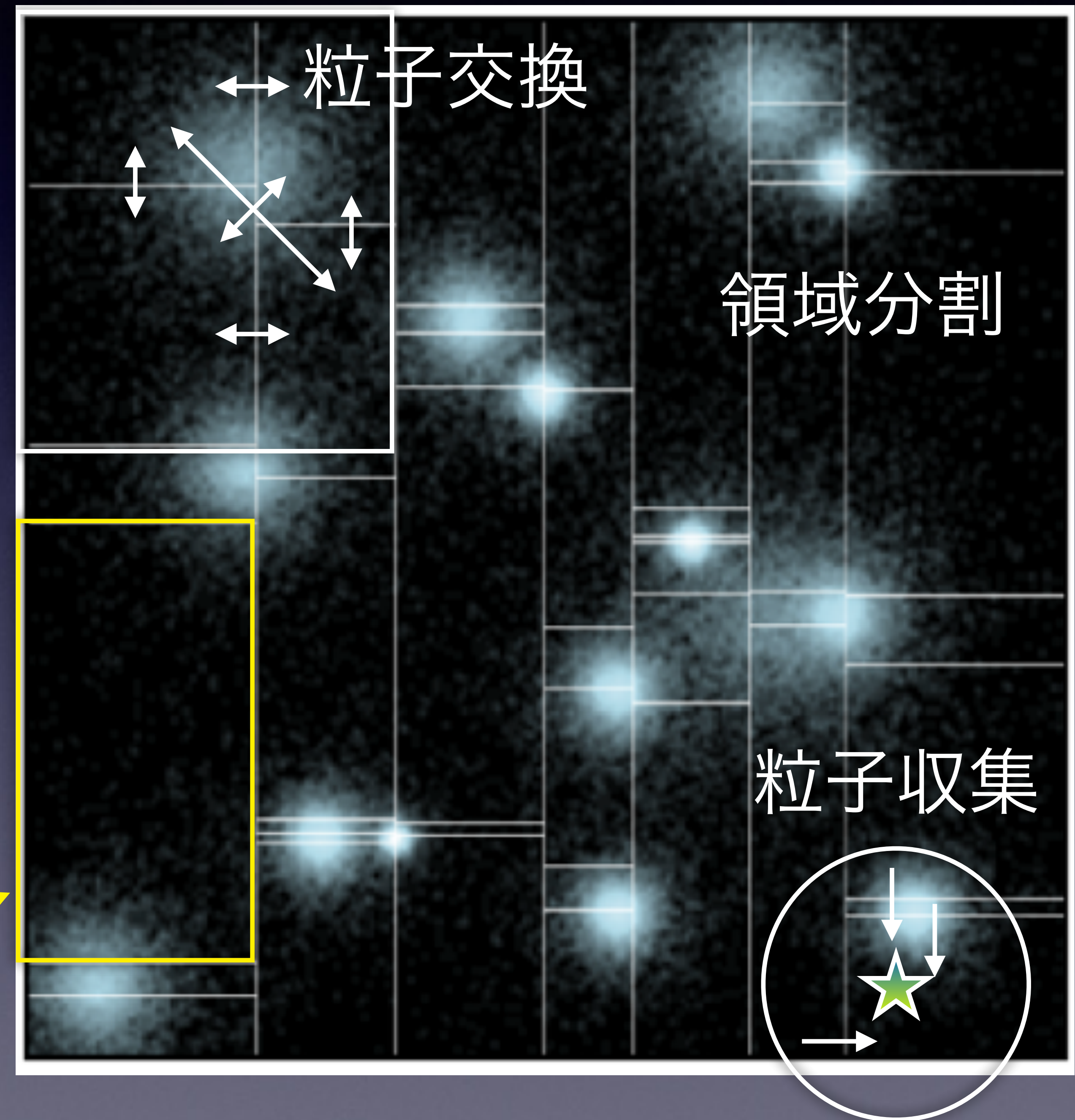
高速にNを小さい数に
減らす方法



粒子シミュレーションの手順

- 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

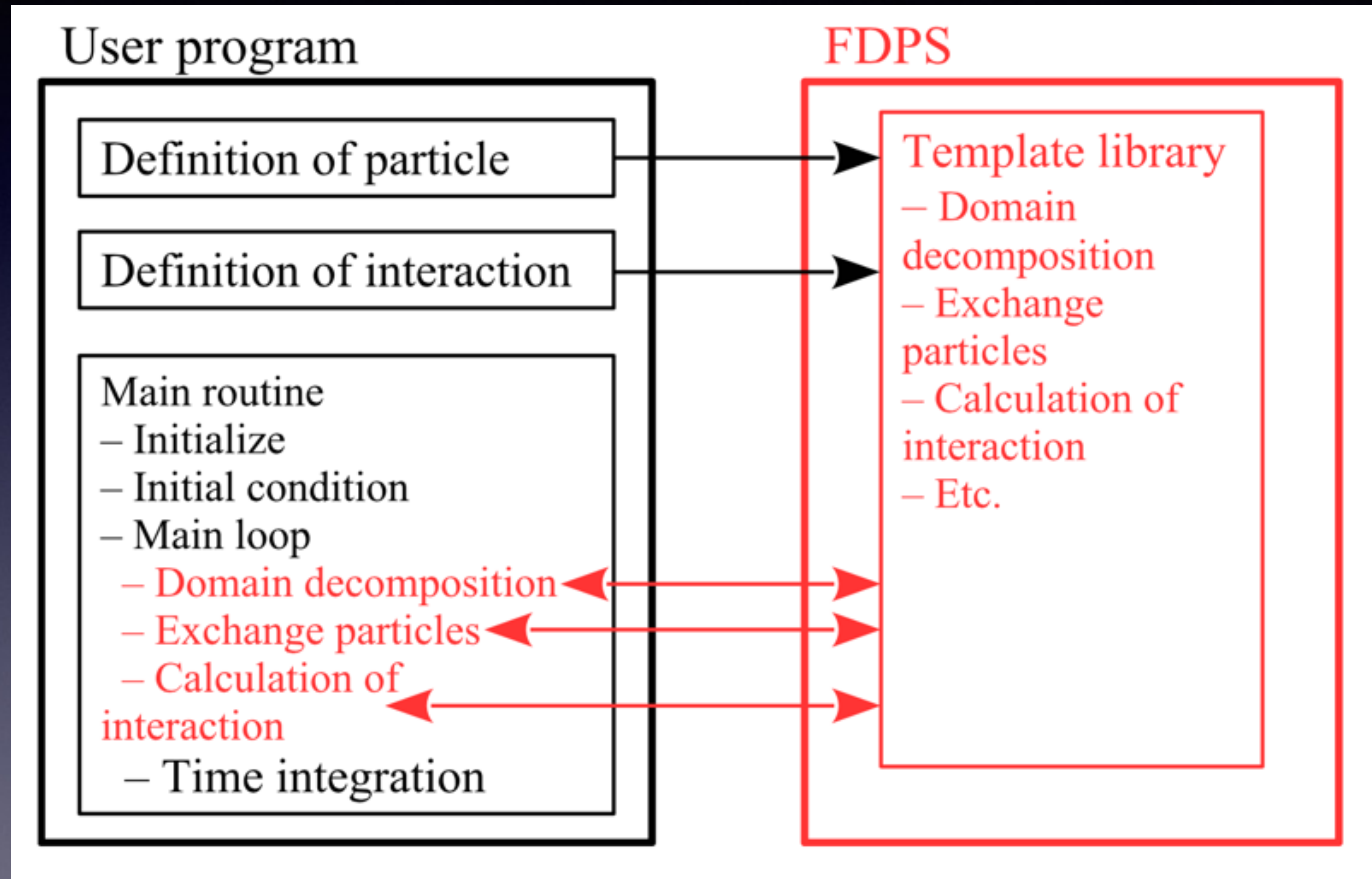
FDPSの実装方針(1)

- 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

FDPSの基本設計



ユーザーはC++の一部の知識を必要とする

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

```
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 xi  = ip[i].pos;
46              F64    ep2 = ip[i].eps
47                  * ip[i].eps;
48              F64vec ai = 0.0;
49              for(S32 j=0; j<nj;j++){
50                  F64vec xj = jp[j].pos;
51                  F64vec dr = xi - xj;
52                  F64 mj  = jp[j].mass;
53                  F64 dr2 = dr * dr + ep2;
54                  F64 dri = 1.0 / sqrt(dr2);
55                  ai -= (dri * dri * dri
56                         * mj) * dr;
57              }
58              force[i].acc += ai;
59          }
60      }
61  };
62
63  template<class Tpsys>
64  void predict(Tpsys &p,
65               const F64 dt) {
66      S32 n = p.getNumberOfParticleLocal();
67      for(S32 i = 0; i < n; i++)
68          p[i].predict(dt);
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  template <class TDI, class TPS, class TTFF>
80  void calcGravAllAndWriteBack(TDI &dinfo,
81                               TPS &ptcl,
82                               TTFF &tree) {
83      dinfo.decomposeDomainAll(ptcl);
84      ptcl.exchangeParticle(dinfo);
85      tree.calcForceAllAndWriteBack
86          (CalcGrav<Nbody>(),
87           CalcGrav<SPJMonopole>(),
88           ptcl, dinfo);
89  }
90
91  int main(int argc, char *argv[]) {
92      F32 time  = 0.0;
93      const F32 tend  = 10.0;
94      const F32 dtime = 1.0 / 128.0;
95      PS::Initialize(argc, argv);
96      PS::DomainInfo dinfo;
97      dinfo.initialize();
98      PS::ParticleSystem<Nbody> ptcl;
99      ptcl.initialize();
100     PS::TreeForForceLong<Nbody, Nbody,
101         Nbody>::Monopole grav;
102     grav.initialize(0);
103     ptcl.readParticleAscii(argv[1]);
104     calcGravAllAndWriteBack(dinfo,
105                             ptcl,
106                             grav);
107     while(time < tend) {
108         predict(ptcl, dtime);
109         calcGravAllAndWriteBack(dinfo,
110                                 ptcl,
111                                 grav);
112         correct(ptcl, dtime);
113         time += dtime;
114     }
115     PS::Finalize();
116     return 0;
117 }
```

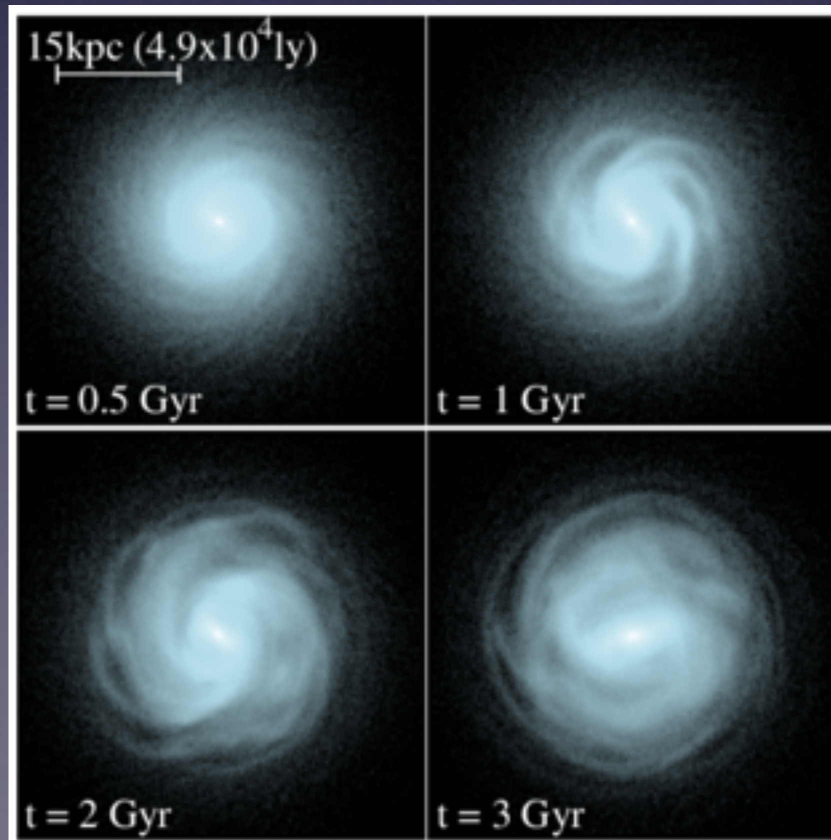
重要なポイント

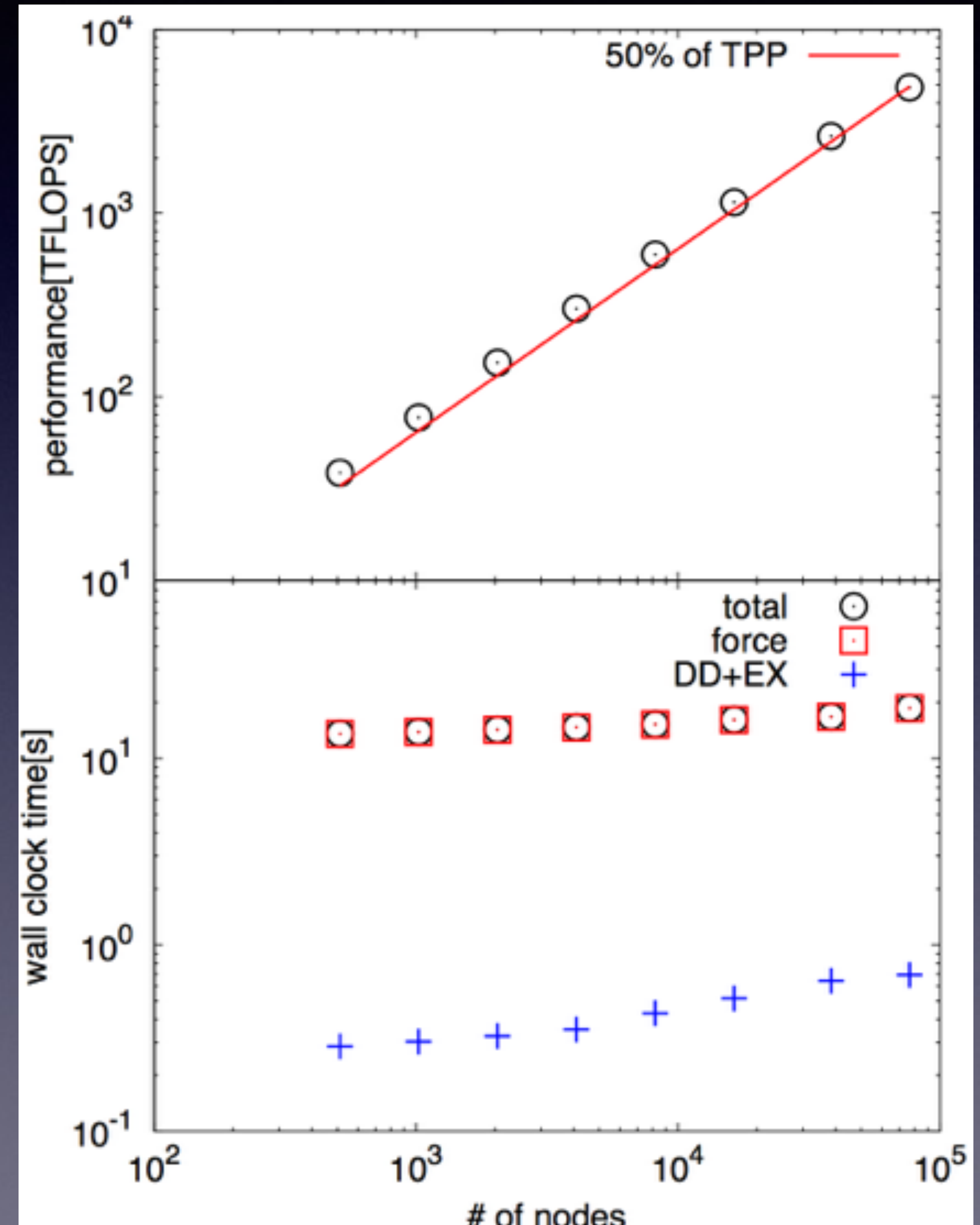
- ユーザーはMPIやOpenMPを考えなくてよい
- 相互作用関数の実装について
  - ２重ループ：複数の粒子に対する複数の粒子からの作用の計算
  - チューニングが必要(FDPSチームに相談可)
    - 除算回数の最小化
    - SIMD演算器の有効利用

性能(N体)

- プラマーモデル
- 粒子数: 2.1x10^6/node
- 精度: Θ=0.4 四重極
- 京コンピュータ

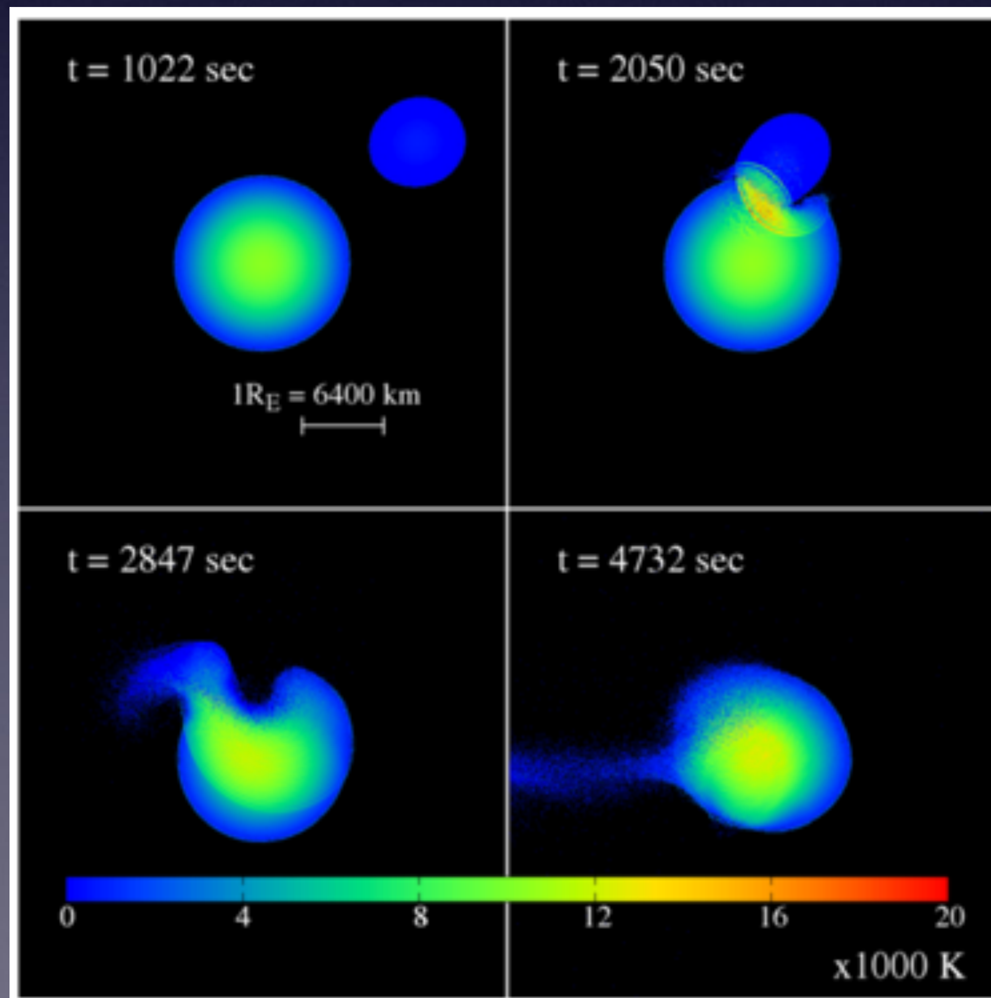
参考画像

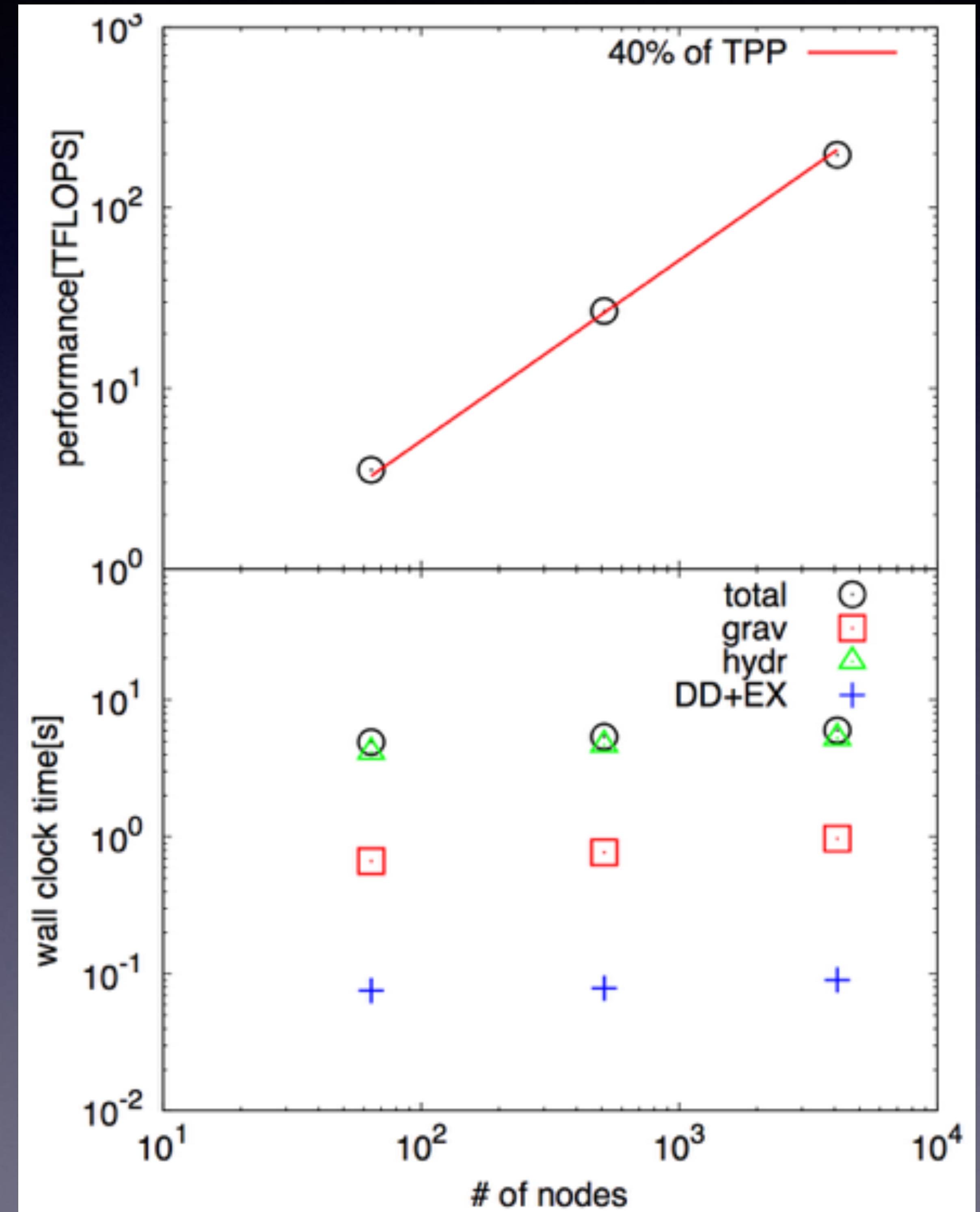




性能(SPH)

- 巨大衝突シミュレーション
- 粒子数: 3.1x10^5/node
- 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117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