FDPS:
Framework for Developing Particle Simulator

Ataru Tanikawa
(University of Tokyo / RIKEN Advanced Institute for Computational Science)

Collaborators
Masaki Iwasawa, Natsuki Hosono, Keigo Nitadori, Takayuki Muranushi, Junichiro Makino
(RIKEN Advanced Institute for Computational Science)

Oct. 30 2015 天体形成研究会 @ 筑波大学CCS

What is FDPS ?

- FDPS is Short for “Framework for Developing Particle Simulator”.
- Using FDPS makes it easier to develop massively parallel particle simulation codes.
- Particle simulations include gravitational N-body, SPH, molecular dynamics, granular dynamics, etc.
- Governing equation:

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

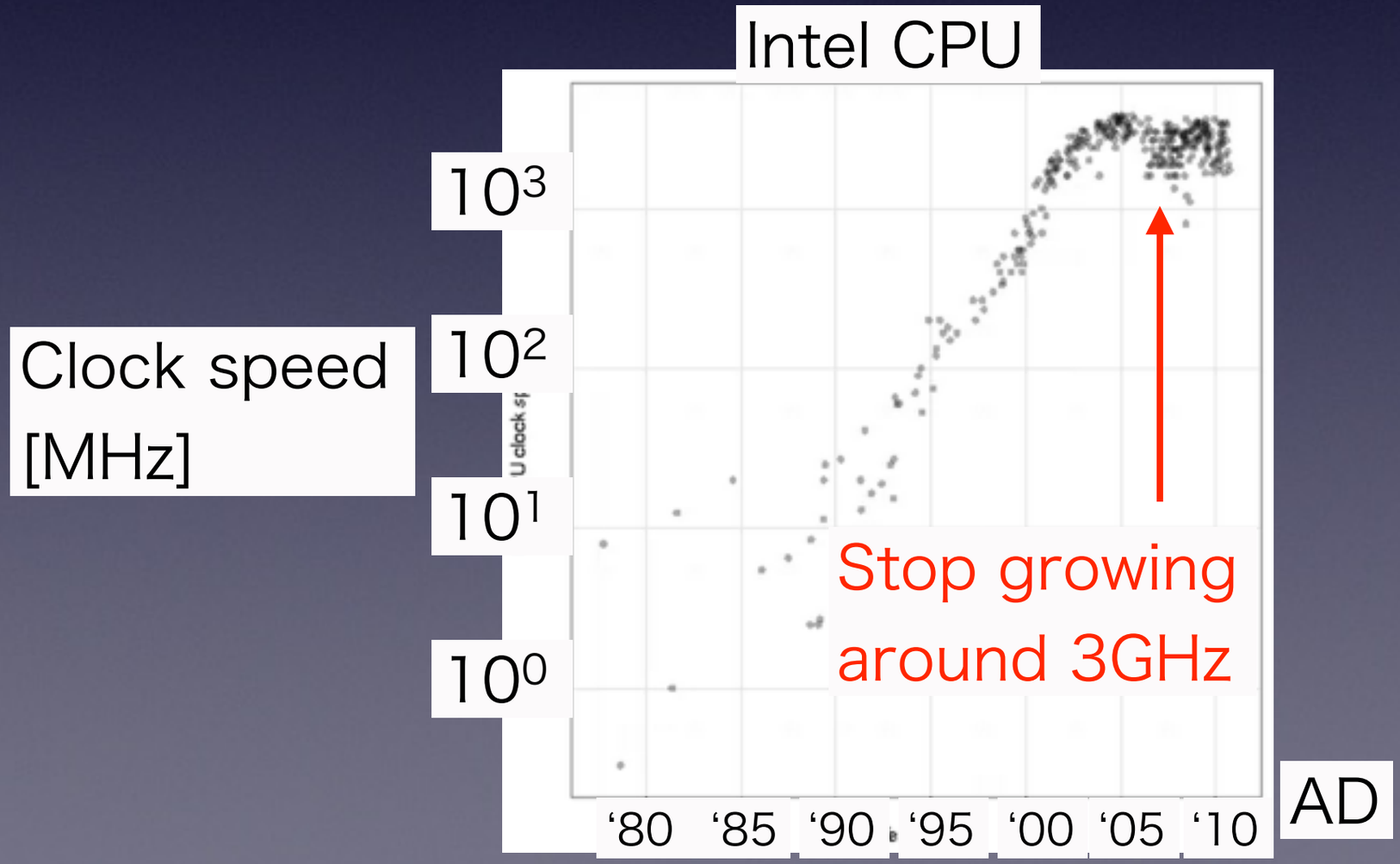
Quantity vector of a particle

Function transforming particle’s quantity to its time derivatives

Pairwise interaction function

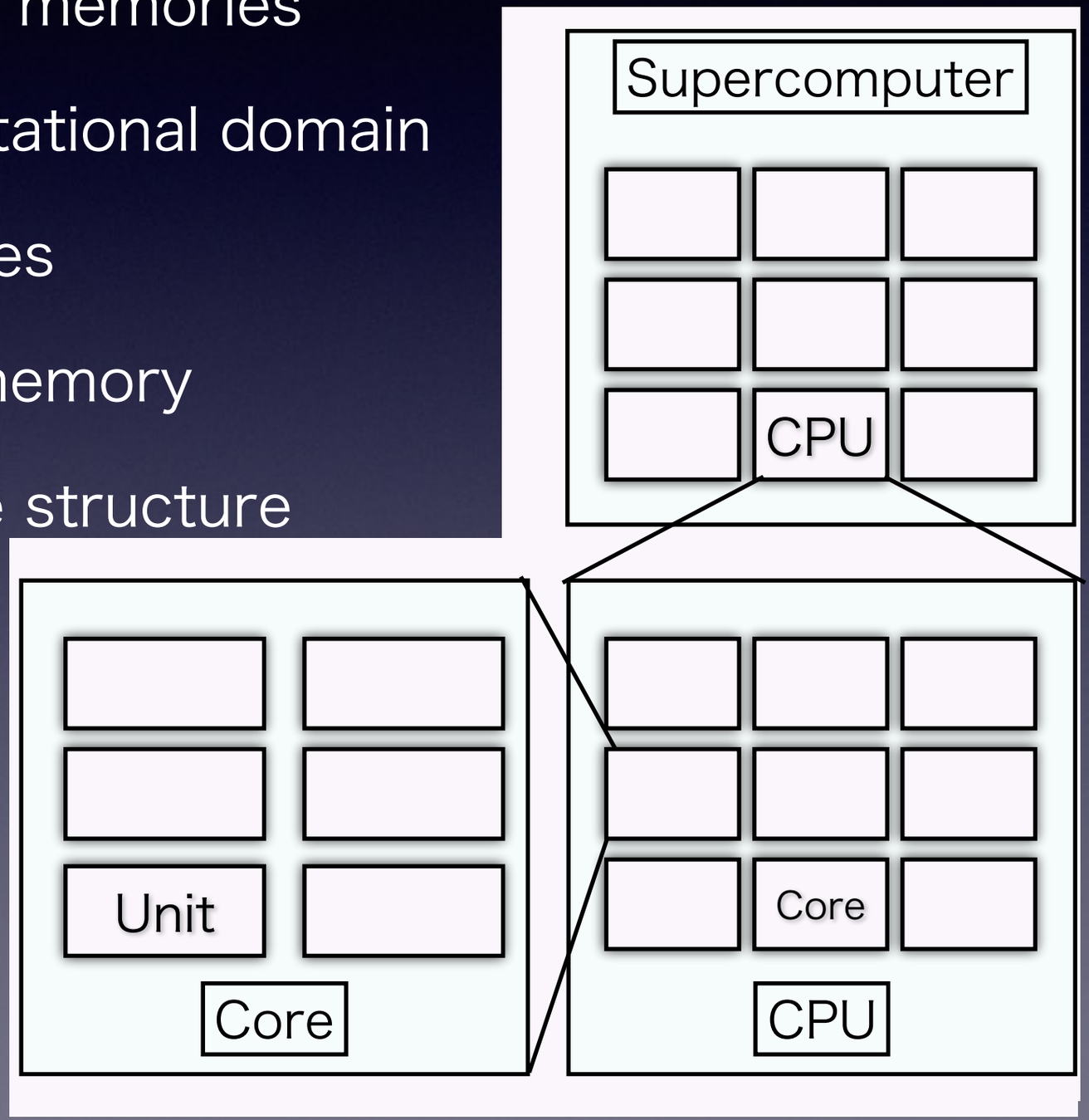
Why is massively parallel particle simulation needed ?

- Simulations with many particles and long time are desired.
- CPU clock speed will not grow any more.



Difficulties of massively parallel particle simulations

- Parallelization on distributed memories
  - Decomposition of computational domain
  - Communication of particles
- Parallelization on a shared memory
  - Multi-traverse across tree structure
  - Load balancing
- Parallelization in a core
  - Efficient use of SIMD unit

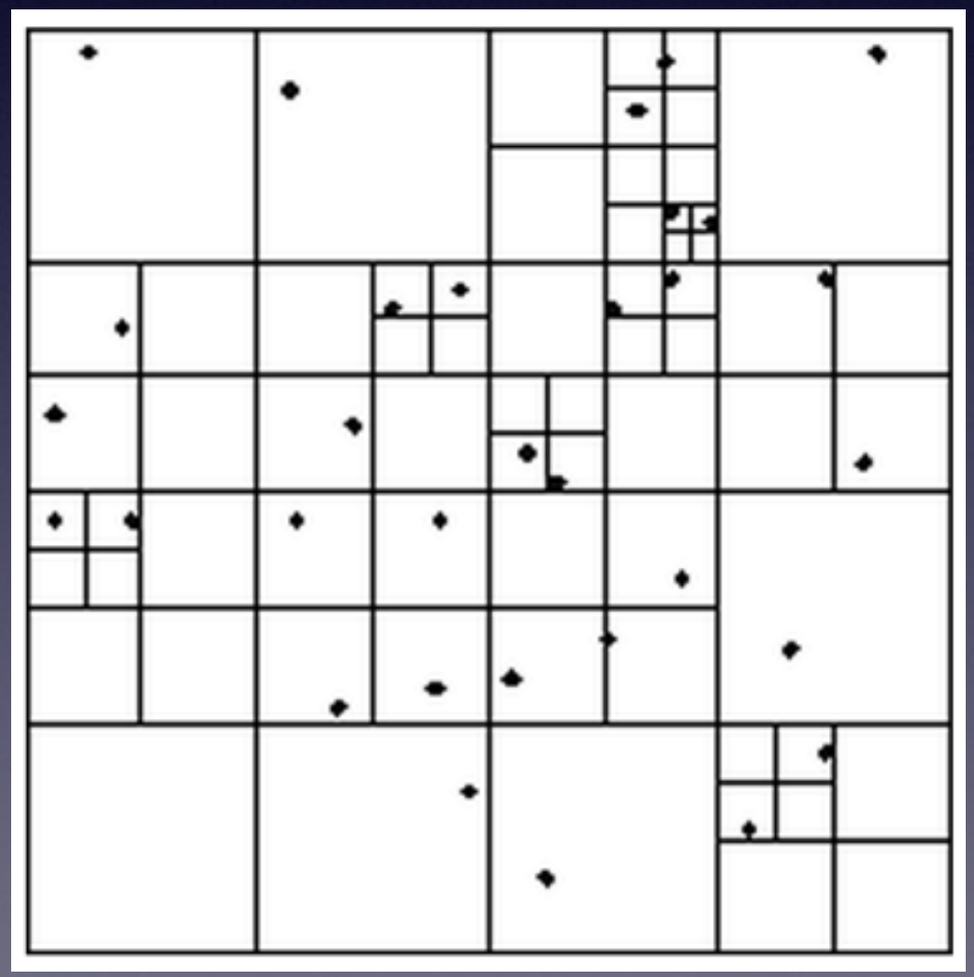


Development of tree structure is not easy even for serial processing

- Efficient use of cache memories
- Construction of true structure

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

Tree algorithm reduces N to log N in the case of N-body simulation.



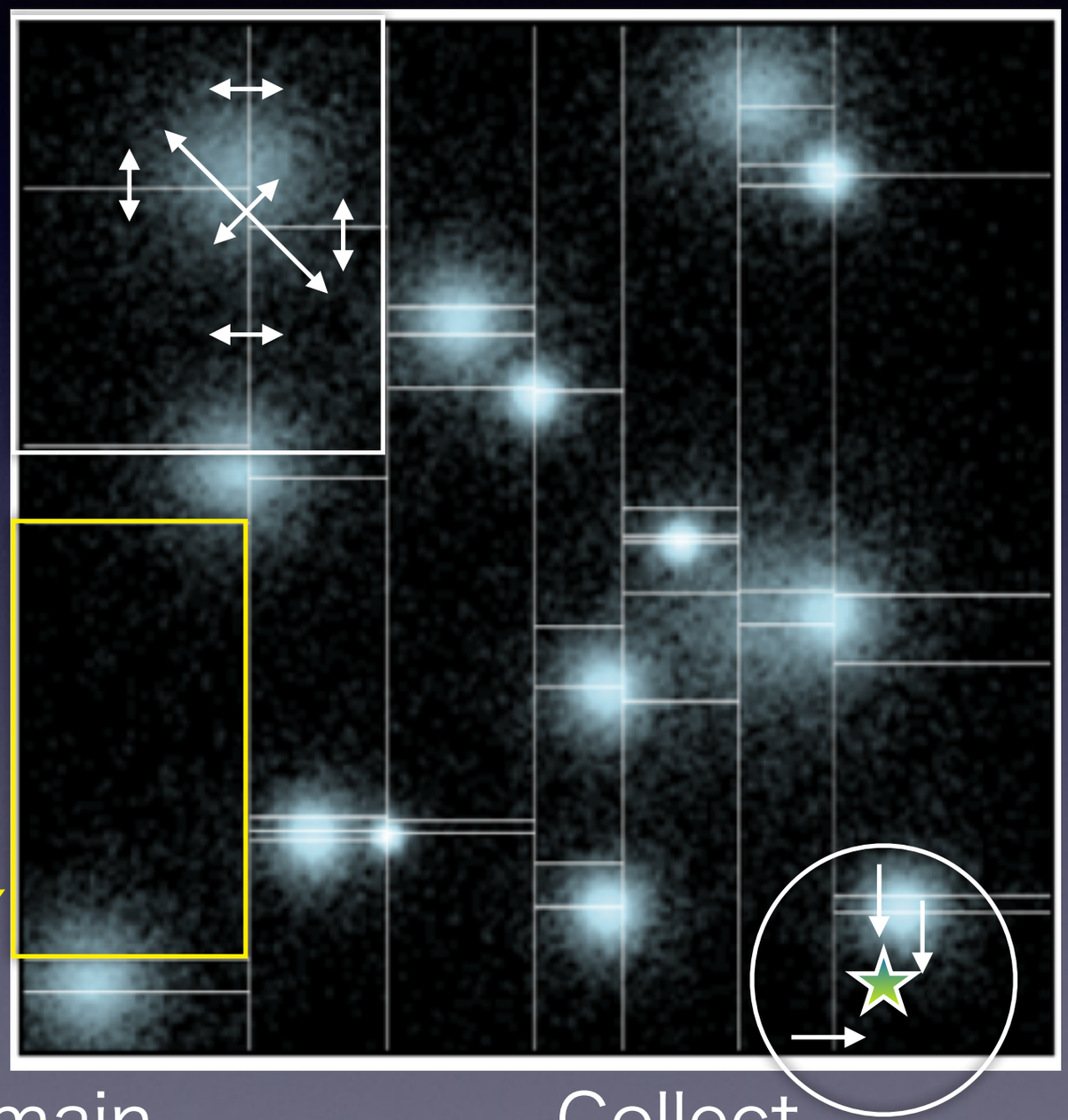
Particle simulation on FDPS

FDPS

- Domain decomposition
- Exchange of particle data
- Collection of particle data for interaction

- Actual calculation of interaction
- Integration of particle orbits

Users



1 MPI process

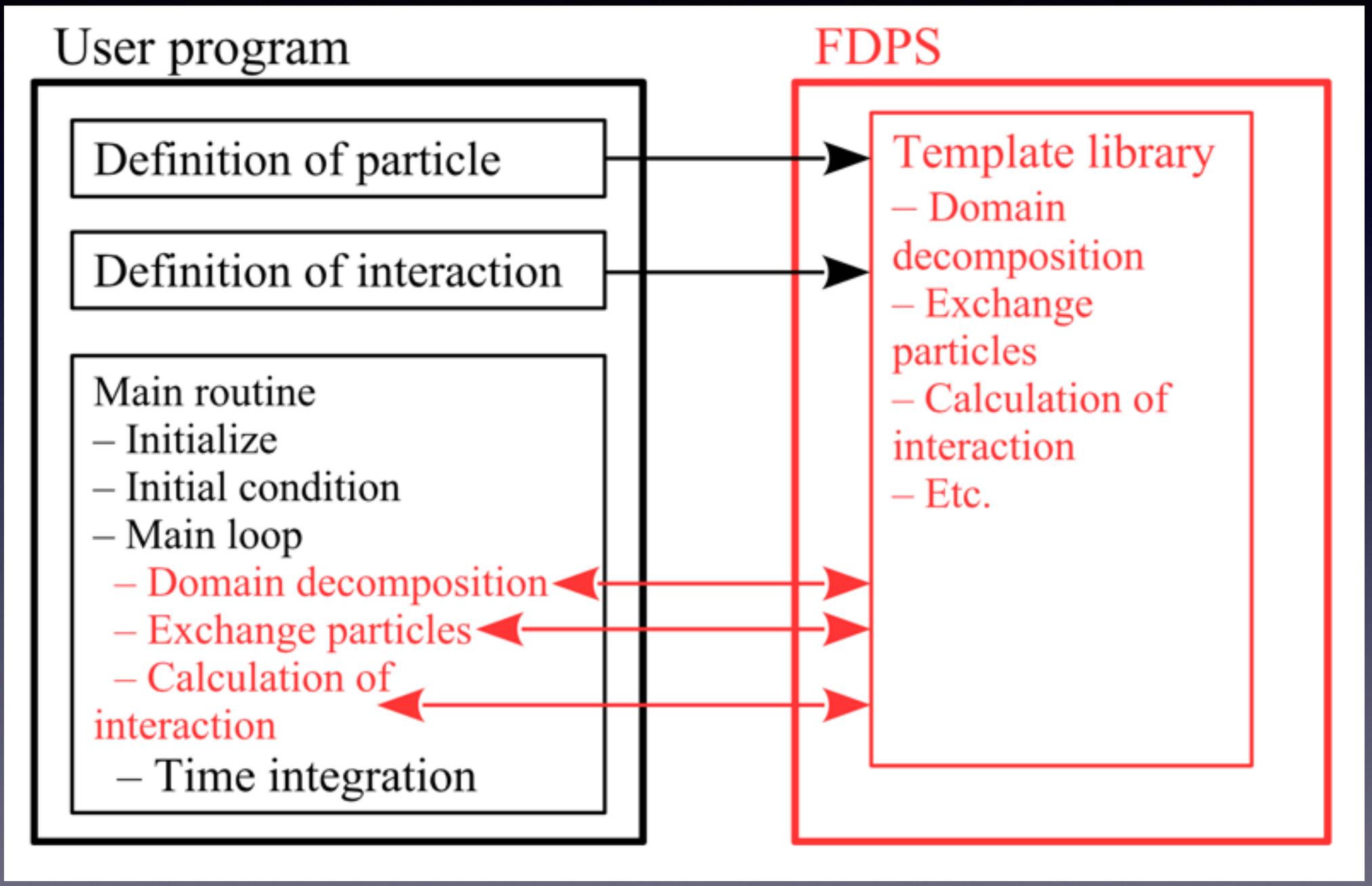
Domain decomposition

Collect particle data

Overview of FDPS code

- C++ language
  - Class (or structure) for particle data defined by users
  - Function pointer (or function object) for interaction defined by users
  - Template class to receive the above class and function pointer

Overview of FDPS code



N-body code

Only 117 lines for massively parallel N-body code

Users do not need to write MPI and OpenMP.

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

Install FDPS

Definition of particle data

Definition of interaction

Main routine

```
#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;
}
```

Definition of particle data

Particle data:
— mass
— position
— velocity
— acceleration
— gravitational softening

Communication with FDPS:
— Which is position?
— Which is mass?
— Which are required for interaction?
— Which are results?

```
 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 };
```

Definition of interaction

— Array of i-particle data
— The number of i-particles
— Array of j-particle data
— The number of j-particles
— Array to store the results

i-particle: particle receiving force
j-particle: particle exerting force

```
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 };
```

Calculate gravitational force on i-particle

Main routine

Template class for exchange of particle data

Template class for interaction calculation

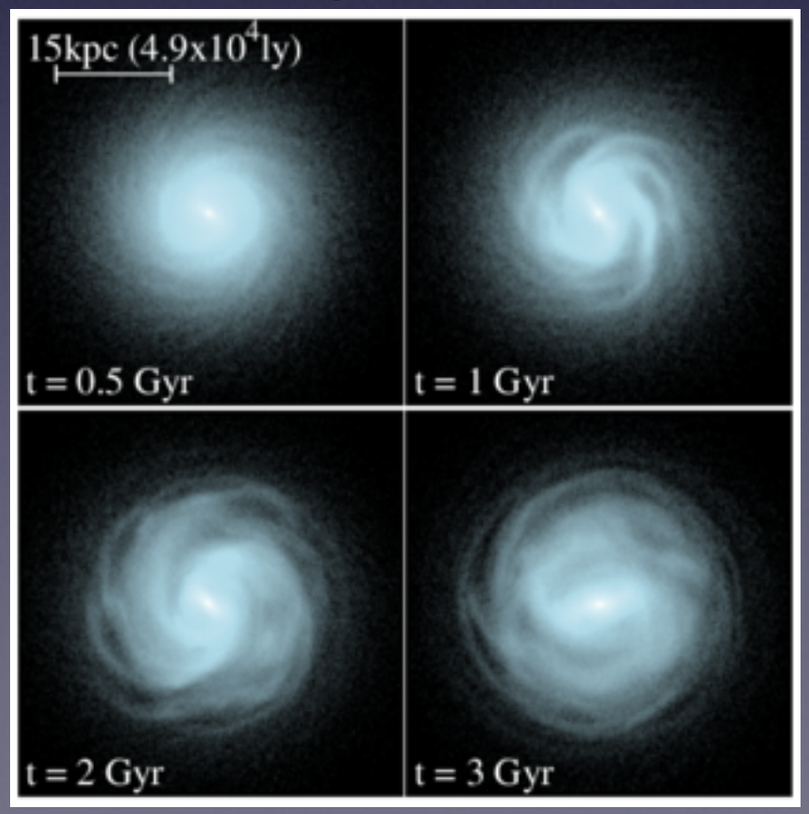
```
template <class TDI, class TPS, class TTFF>
void calcGravAllAndWriteBack(TDI &dinfo,
                             TPS &ptcl,
                             TTFF &tree) {
    dinfo.decomposeDomainAll(ptcl);
    ptcl.exchangeParticle(dinfo);
    tree.calcForceAllAndWriteBack
        (CalcGrav<Nbody>(),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

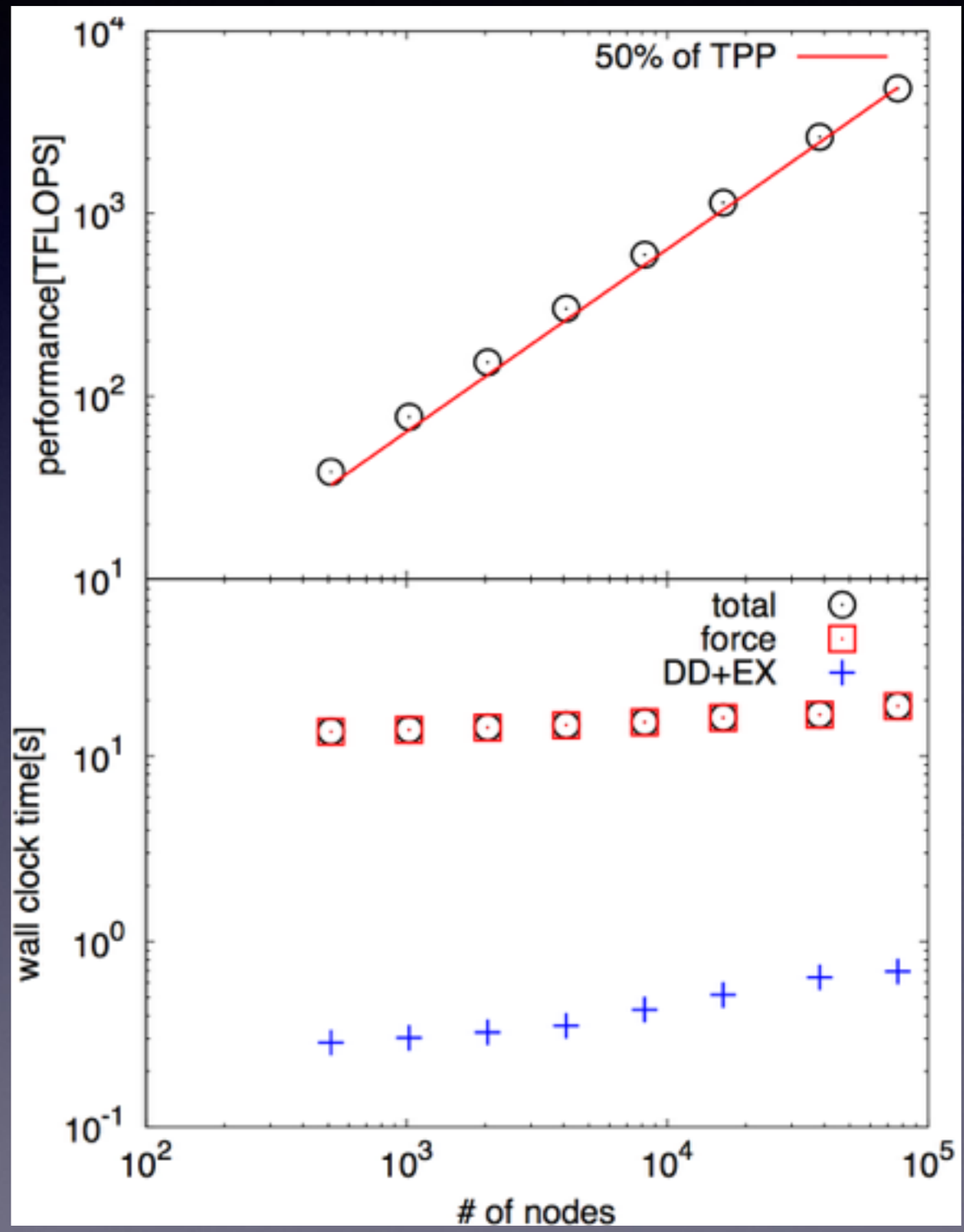
```
91  int main(int argc, char *argv[]) {
92      F32 time  = 0.0;
93      const F32 tend  = 10.0;
94      const F32 dtime = 1.0 / 128.0;
95      PS::Initialize(argc, argv);
96      PS::DomainInfo dinfo;
97      dinfo.initialize();
98      PS::ParticleSystem<Nbody> ptcl;
99      ptcl.initialize();
100     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 }
```

N-body on K computer

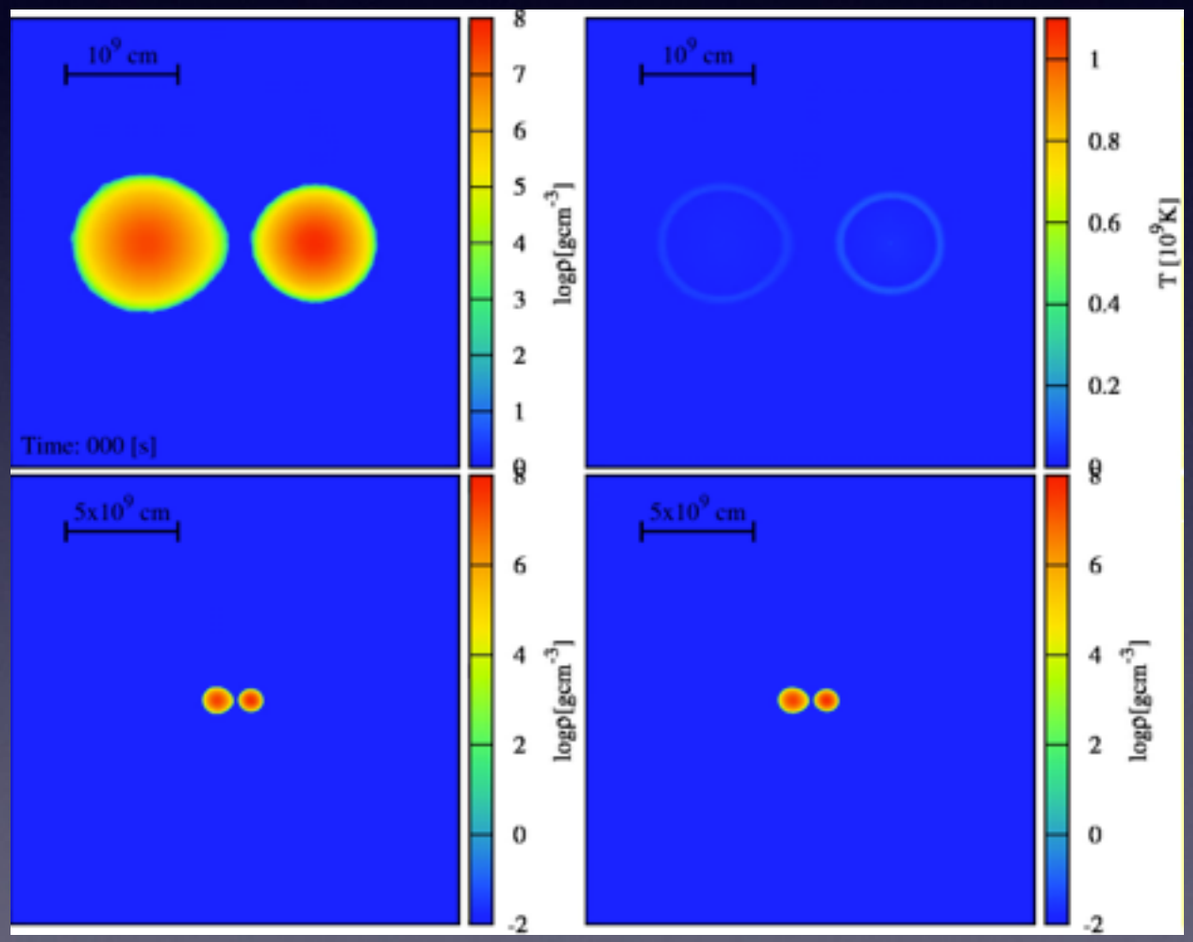
- Plummer model
- N: 2.1x10^6/node
- Accuracy: Θ=0.4, up to quadrupole

Example of N-body

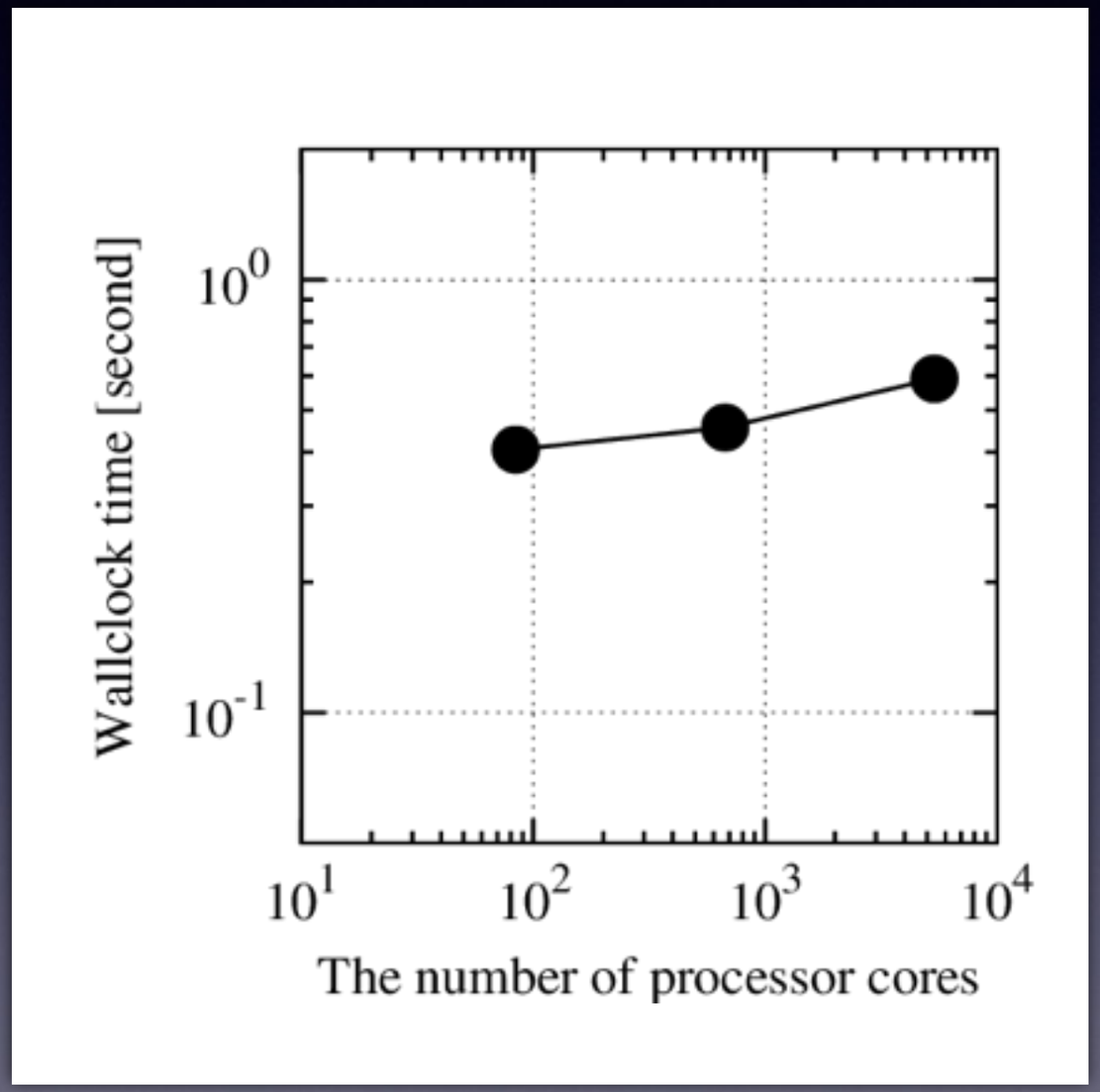




SPH performance @ CfCA XC30 (Merging binary white dwarfs)



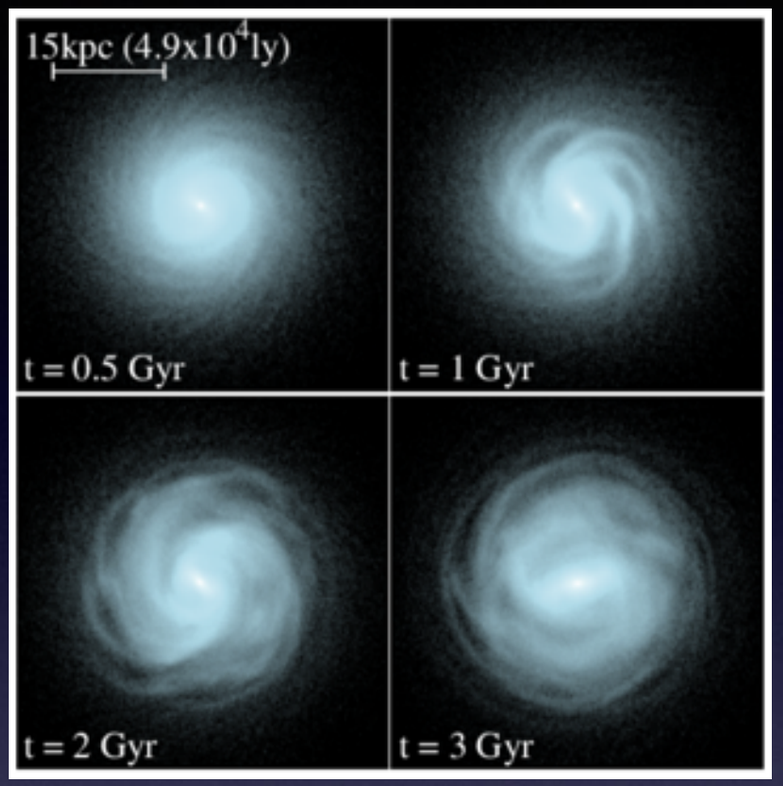
Tanikawa et al. (2015)



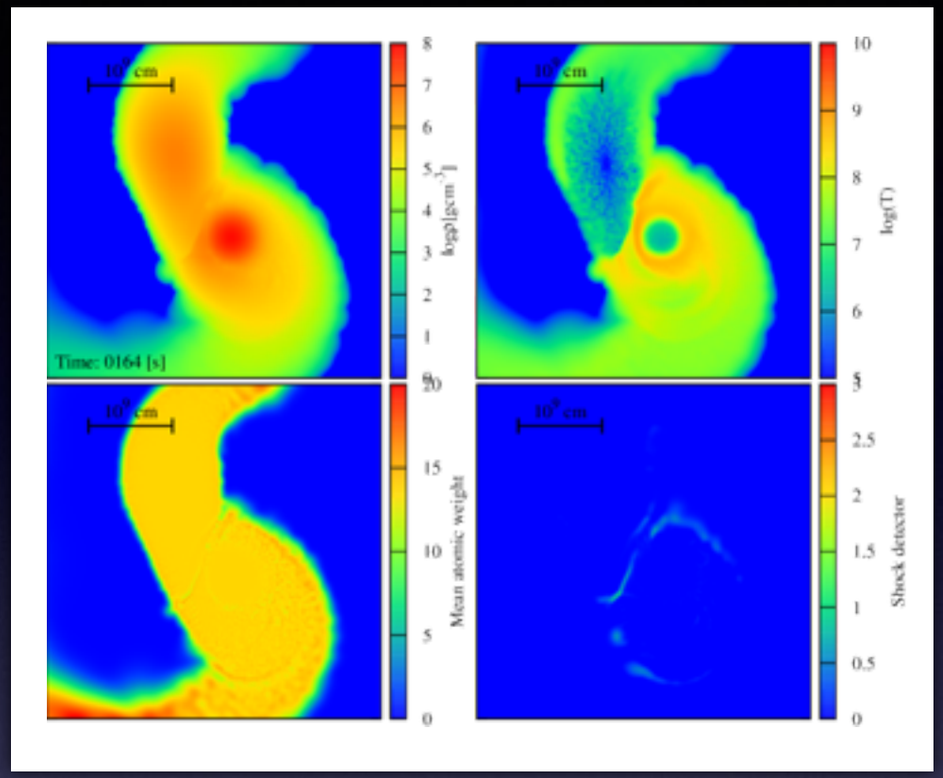
N: 8000/core

Applications

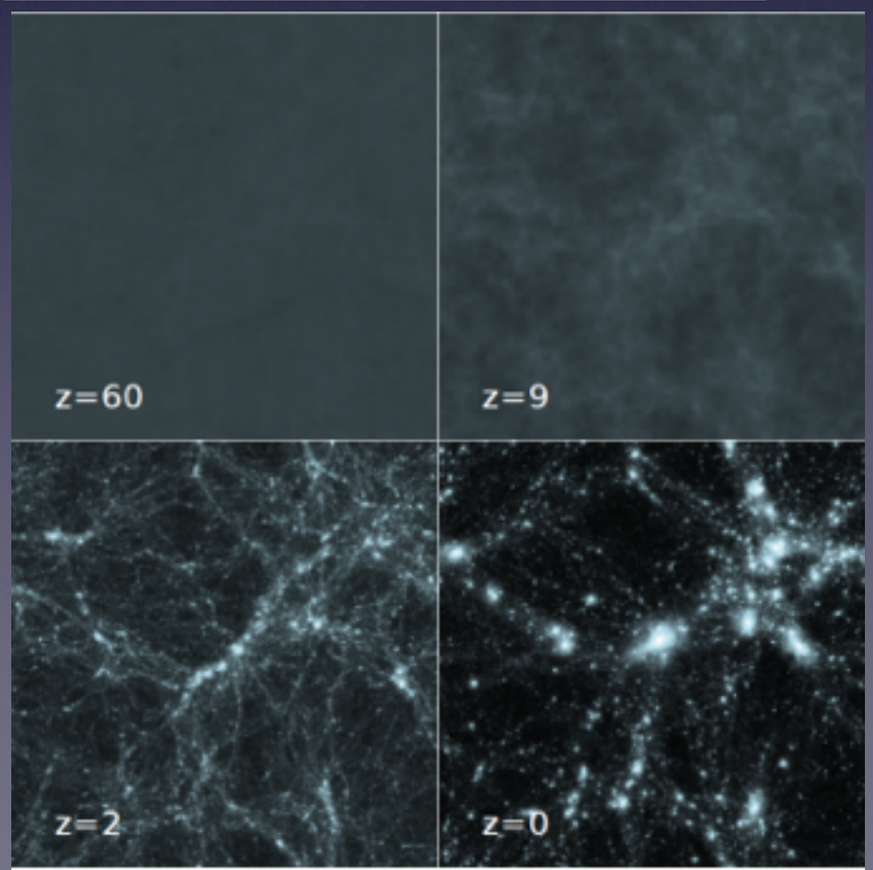
Spiral galaxy (N-body)



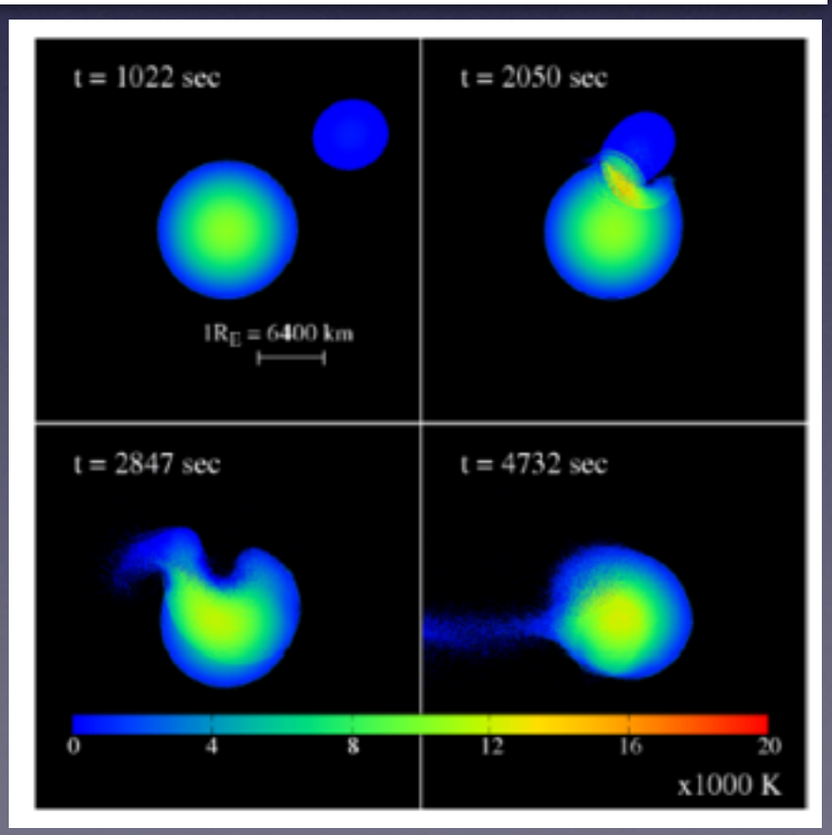
Merging binary white dwarfs (SPH)



Cosmology (N-body with periodic boundary)



Giant impact (SPH)



Sample code

- Gravitational N-body, open boundary
- SPH, ideal EoS
- SPH, ideal EoS, gravity
- Van der Waals
- Gravitational N-body, periodic boundary (Yoshikawa)

N-body on Xeon Phi

- Benchmark
  - N: 1 million
  - Tree accuracy: θ =0.5, up to monopole
  - Interaction accuracy: single precision
  - inverse square root accuracy: 23-24bit
- Computers
  - Xeon Phi 5110P (Performance 1TFlops, TDP 225W)
  - XC30 1 node (Performance 1TFlops, TDP 270W)
- Wallclock time per step
  - Xeon Phi: 0.35 second
  - XC30: 0.21 second
- Caution
  - FDPS has been not yet tuned for many-core processors.

Publication

- https://github.com/FDPS/FDPS
  - サポート: fdps-support@mail.jmlab.jp
- Iwasawa, Tanikawa, Hosono, Nitadori, Muranushi, Makino, WOLFHPC 2015: proceedings of Fifth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing
- FDPS講習会 2015/07/22 @ 神戸

Future work

- FDPS 2.0
  - Support APIs to calculate interactions on many-core processors, such as GPU and PEZY-SC.
- FDPS ?.0
  - All on many-core processors.

Summary

- https://github.com/FDPS/FDPS
- We have developed FDPS.
- You can develop particle simulation codes easily.
  - N-body code can be developed only in 117 lines.
  - N-body performance is 50% of theoretical peak performance of K computer.
  - Weak scaling of SPH is good on CfCA XC30.