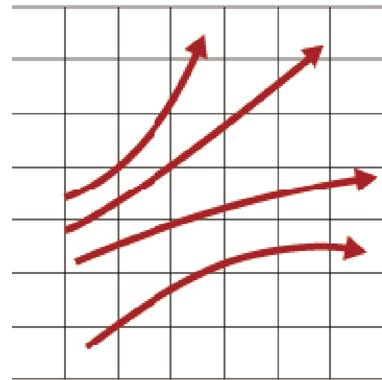
FDPS(Framework for Developing Particle Simulator)を用いた惑星形成シミュレーションへの応用

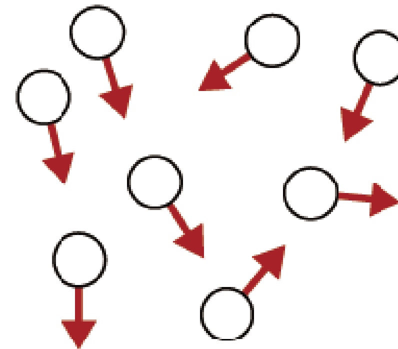
岩澤全規、谷川衝、細野七月、
似鳥啓吾、村主崇行、牧野淳一郎
理化学研究所　計算科学研究機構
粒子系シミュレータ研究チーム

2015年3月14日-17日

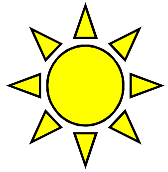
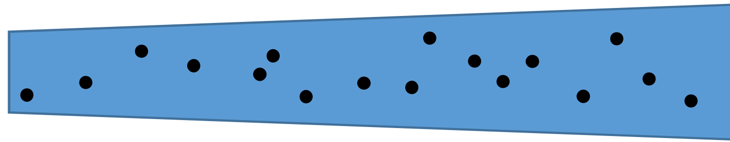
粒子シミュレーション

- 系を相互作用する多数の粒子によって表現し、個々の粒子の発展方程式を解くことで、系の進化をシミュレーションする。
  - 粒子が自動的に集まり系を表現するため、物体の衝突や破壊、形が大きく変わる系、密度コントラストの高い系等のシミュレーションに強い。
  - 科学、工学の幅広い分野で使われている。
  - 重力N体シミュレーション、MD、DEM、SPH、MPS、メッシュフリ法等。

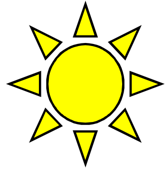
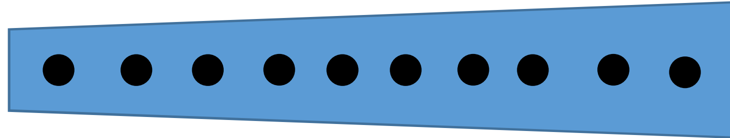

メッシュ法


粒子法

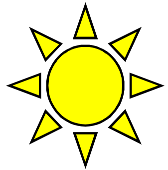
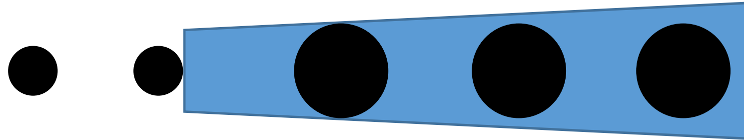
惑星形成における粒子シミュレーション

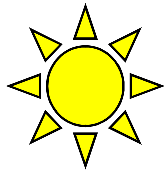
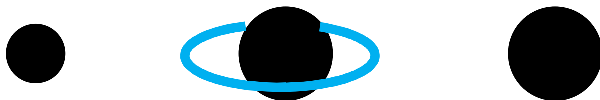
ダスト集積し微惑星へ(DEM)

微惑星集積し原始惑星へ(N体)

巨大衝突を経て惑星へ(SPH)

様々な段階で粒子法が使われている

リングシミュレーション(DEM)

大規模並列粒子シミュレーションコード開発の困難

- 分散メモリー環境での並列化の困難。
  - 計算領域の分割と粒子データの交換
  - 相互作用計算の為の粒子データの交換
  - 相互作用計算や近傍粒子探査を高速に行うために、粒子をツリー構造を使って管理
- 多くの研究者がプログラムの開発に多くの時間を取られてしまう。
- これらの困難な部分を意識せずにコード開発が出来るフレームワーク(FDPS)の開発を行った(iwasawa et al 2016, arXiv:1601.03138)。

FDPSとは

- Framework for Developing Particle Simulatorの略。
- 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。
  - ユーザは並列化やツリー構造による粒子管理など、プログラム開発を困難にしている部分を意識する必要がない。
- 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

粒子データのベクトル

  - 任意の2粒子間相互作用を扱うことが出来る。

FDPSの実装方針

- 内部実装の言語としてC++を選択
  - 高い自由度と高い性能を両立させるためにFDPSはC++のテンプレートライブラリになっている。
  - ユーザーは相互作用関数と粒子データを定義し、FDPSのAPIを使ってプログラムの開発を行う。
- 並列化
  - 分散メモリー環境(ノード間):MPI
  - 共有メモリー環境(ノード内):OpenMP
    - FDPSが提供するAPIは並列化されており、ユーザーは並列化を意識してコードを書く必要がない。

FDPSを用いた粒子シミュレーションの流れ

1. 計算領域全体を分割する。
2. 計算領域に合わせて粒子を再配置する。
3. 各プロセスが担当する粒子への相互作用を計算する。
4. 相互作用の結果を使って粒子の情報を更新する。

- FDPSは手順1,2,3を担当。
- 手順1,2,3に対応したクラスがある。
  - DomainInfoクラス: 領域のデータを持ち、領域分割を行う。
  - ParticleSystemクラス: 粒子のデータを持ち、粒子交換を行う。
  - TreeForForceクラス: 相互作用の計算を行う。
  - ユーザーはこれらのクラスのインスタンスを作り、メンバ関数を呼び出すことでそれぞれの処理を行う。

FDPSを使ったプログラム例(重力N体)

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 }
```

FDPSのインストール(ヘッダーファイルのインクルード)

粒子クラスの定義

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

FDPSを使ったプログラム例(粒子クラス)

```
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;
    }
};
```

- いくつかのメンバ関数は必須、関数名は固定。
  - getPos(): 粒子座標を返す
  - getCharge(): 質量や電荷を返す
  - copyFromFP(): 粒子クラス間でのコピー
  - copyFromForce(): 相互作用の結果をコピー
  - clear(): 相互作用の結果を消去
- これらのメンバ関数を使ってFDPSは粒子クラスにアクセスする。

FDPSを使ったプログラム例(相互作用関数)

```
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;
        }
    }
};
```

相互作用を受ける粒子のループ
(i粒子ループ)

相互作用を及ぼす粒子のループ
(j粒子ループ)

- ニュートン重力の場合
- 関数内で2重ループを書く
  - Barnesの方法を使うため。

FDPSを使ったプログラム例(メイン関数)

```
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;
}
```

FDPSの初期化
領域分割インスタンス
粒子交換インスタンス
相互作用用計算インスタンス
FDPSの終了処理

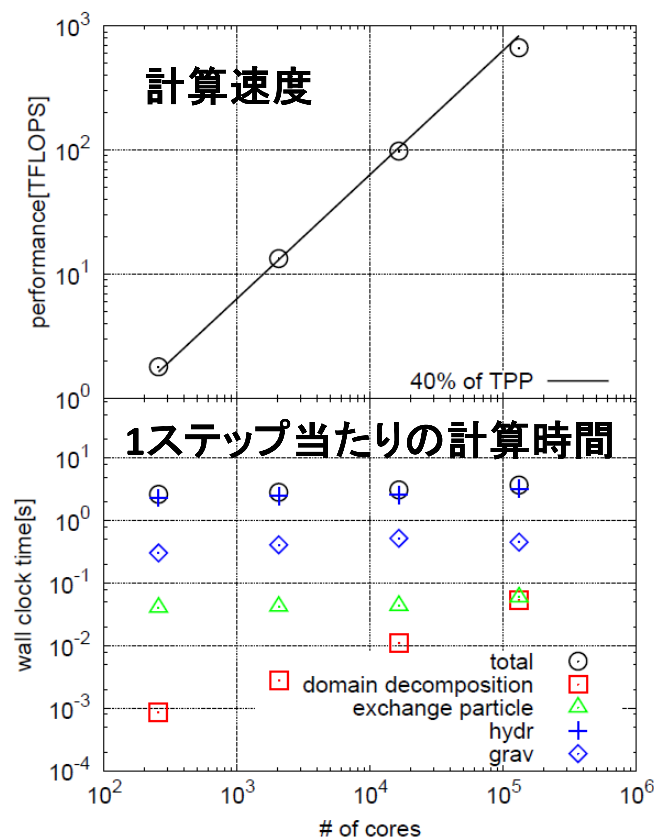
```
template <class TDI, class TPS, class TTFF>
void calcGravAllAndWriteBack(TDI &dinfo,
                             TPS &ptcl,
                             TTFF &tree) {
    dinfo.decomposeDomainAll(ptcl);
    ptcl.exchangeParticle(dinfo);
    tree.calcForceAllAndWriteBack
        (CalcGrav<Nbody>(),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

領域分割実行
粒子交換実行
相互作用用計算実行

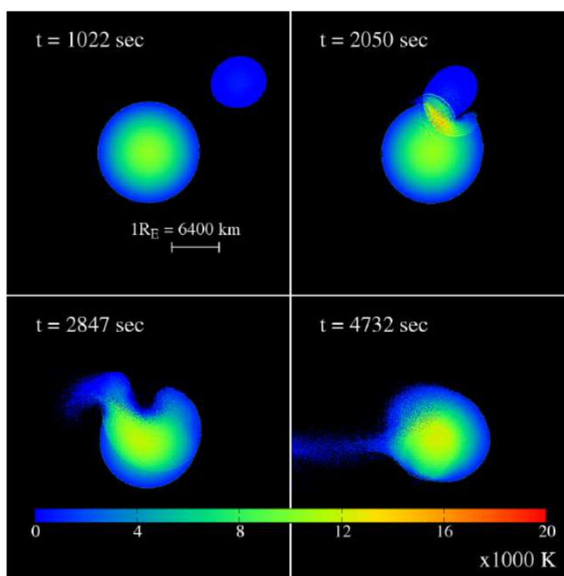
- 領域分割、粒子交換、相互作用用計算に関するクラスのインスタンスを作り、メンバ関数を呼び出す。
- 明示的にMPIを呼んでいない。

巨大衝突シミュレーション

N=40K/core (weak scaling)



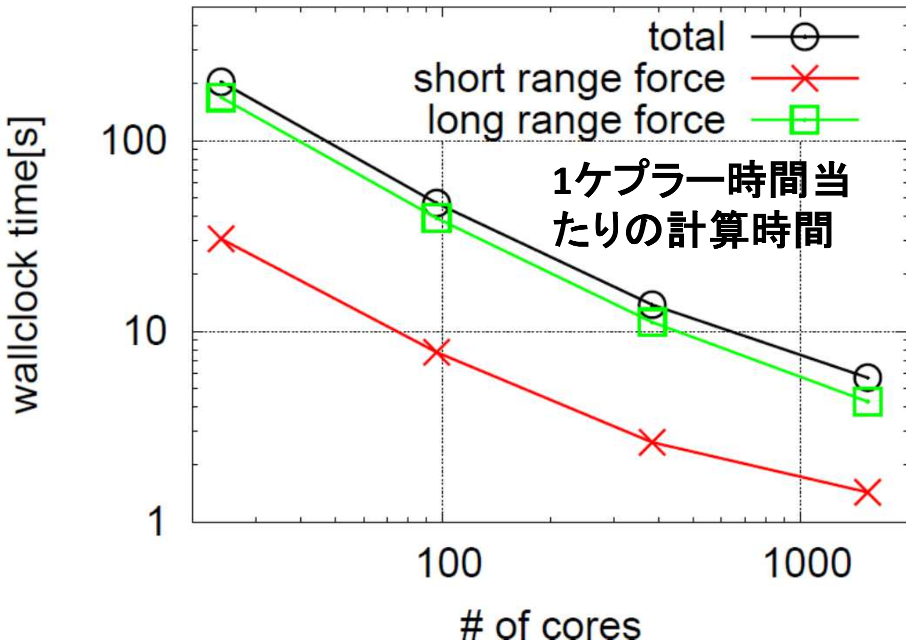
- FDPSを用いて、SPH法により巨大衝突シミュレーションを「京」コンピュータで実行。
- 実行効率: 約40%



詳細はZ311b!

微惑星集積シミュレーション

N=2M (Strong scaling)



- FDPSを用いて、Particle-Particle Particle-Tree法(Oshino et. al 2011)により微惑星集積シミュレーションを「アテルイ」で実行。
- 初期条件：a=0.5-5AU

$$\rho(a) = 10\left[g/cm^2\right]\left(\frac{a}{1[AU]}\right)^{-1.5}$$

- 1ケプラー時間を約6秒@N=2M

詳細はZ303b!

まとめ

- 大規模並列粒子シミュレーションプログラムの開発を容易にするフレームワーク:FDPSの開発を行った。
- Tree法による重力N体シミュレーションなら数百行で書ける。
- 惑星形成用アプリケーションとして巨大衝突シミュレーションや微惑星集積シミュレーションを実行中。
  - 微惑星形成シミュレーションや惑星リングのシミュレーションも可能であり、今後開発を行っていきたい。
- https://github.com/FDPS/FDPS でver.2.0をMITライセンスで公開中。