FDPS講習会

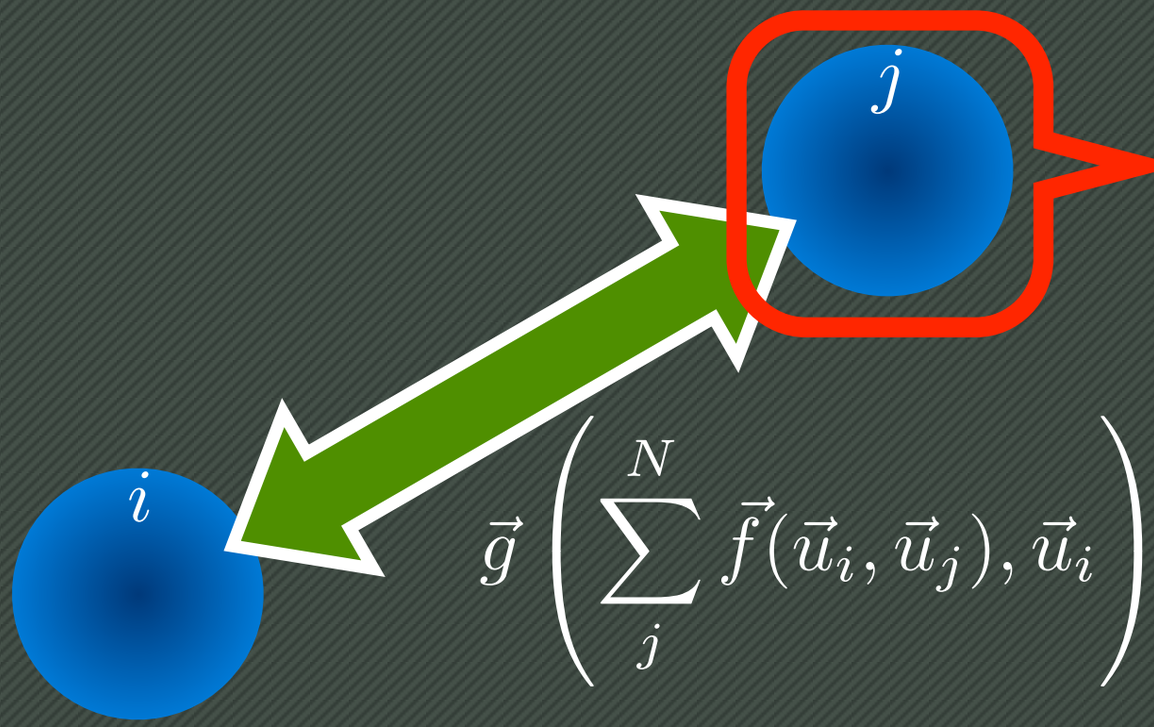
実践編

細野 七月

理研 AICS 粒子系シミュレータ開発チーム

FDPSが扱うもの

●個々の要素に作用する力が、
粒子間相互作用の重ねあわせで記述できるもの



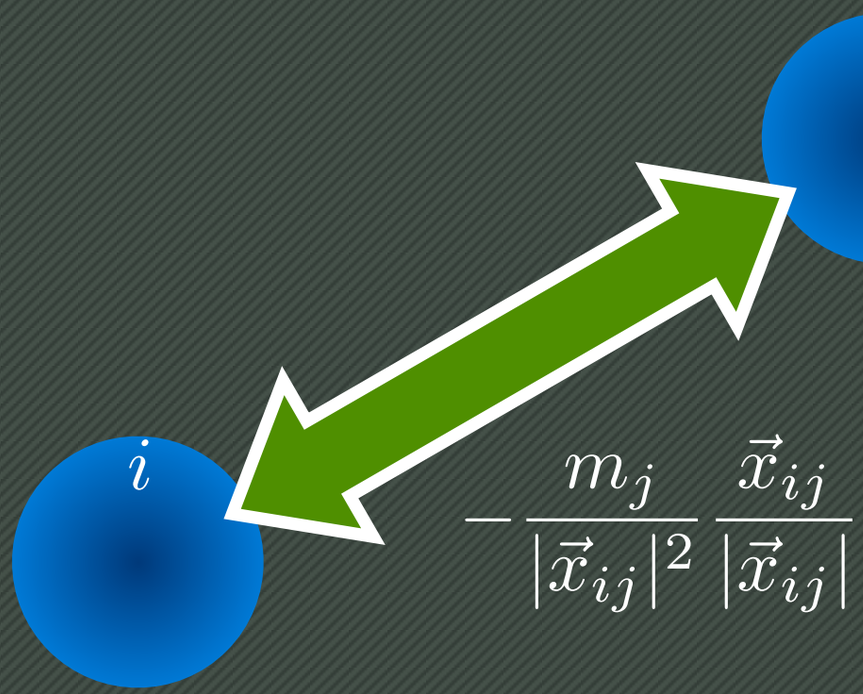
$$\vec{g}\left(\sum_j^N \vec{f}(\vec{u}_i, \vec{u}_j), \vec{u}_i\right)$$

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

FDPSが扱うもの

- 個々の要素に作用する力が、
  粒子間相互作用の重ねあわせで記述できるもの

  (例)ニュートン重力



$$\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$$
$$\frac{d\vec{u}_i}{dt} = -\sum_j \frac{m_j}{|\vec{x}_{ij}|^2}\frac{\vec{x}_{ij}}{|\vec{x}_{ij}|}$$

実際に書く前に…

●FDPSを使うのに必要な知識

Cと(ほんのすこしの)C++

●``ほんのすこし''のC++ってなに？

(1)クラス

(2)メンバ関数

(3)テンプレート

●以下では、それぞれについて、

◆それは何か？

◆FDPS内でどう使われているか？

の順に解説

(1)クラス

Class

2015/07/22 FDPS講習会

クラスの概略

●**複数のデータをまとめたもの**

Cで言うところの構造体

●クラスを使うと…

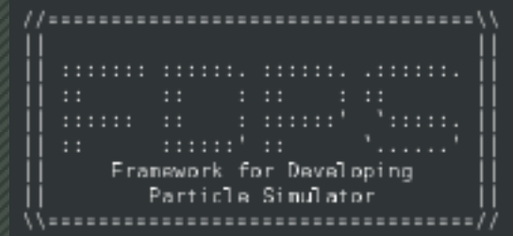
◆**演算子を定義**出来る。

◆**サブルーチンに値を送る**ときに楽。

◆**後からデータ付け足す**のも楽。

●FDPS中では、三次元ベクトルをやりとりするのに便利な**ベクトルクラス**が用意されている。

また、ユーザーは粒子データをやりとりするために必要な**粒子クラス**を記述する必要がある。



ベクトルクラス

●数学演算子を定義している例

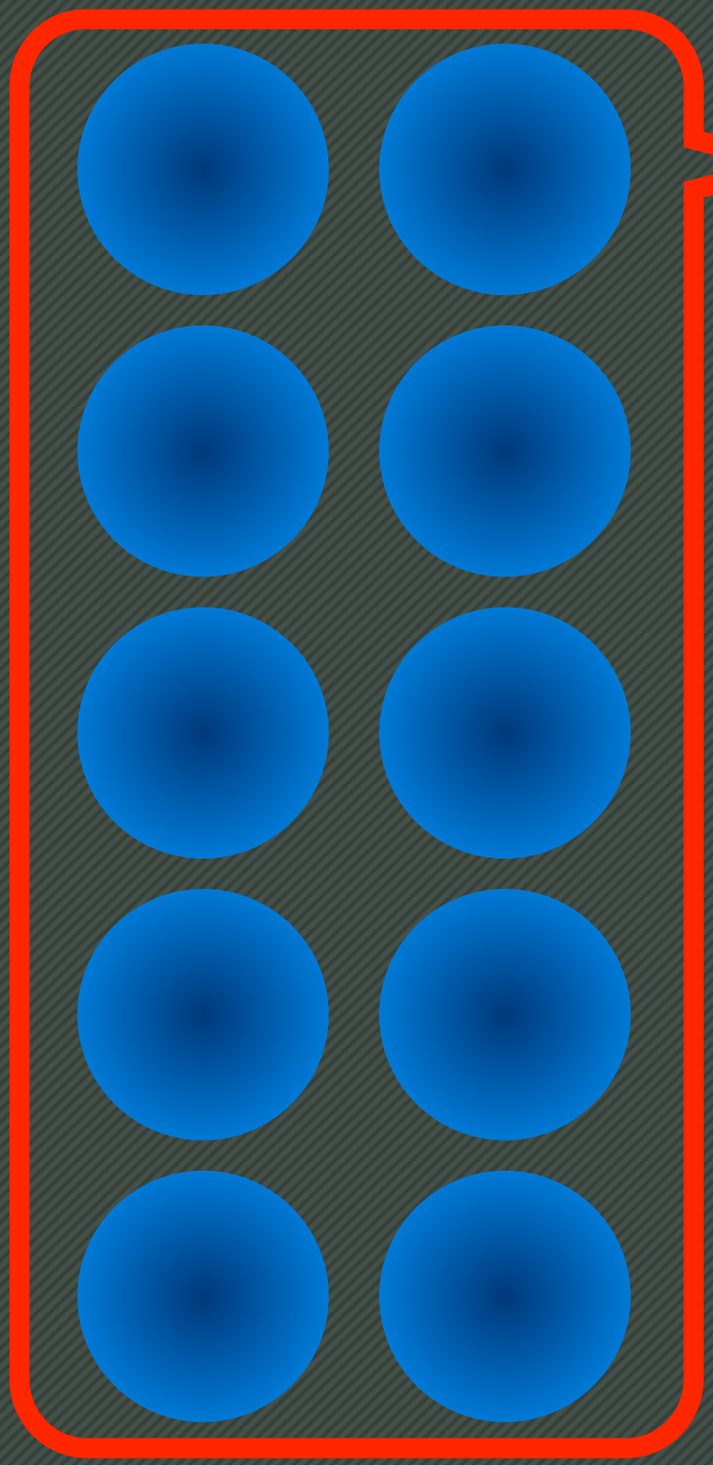
●三次元ベクトル演算を実装してあるFDPSの組み込みクラス

vec.cpp

```
1   PS::F64vec v1 = PS::F64vec( 1.0,  2.5, -3.0);
2   PS::F64vec v2 = PS::F64vec(-2.0, -1.5,  2.0);
3
4   PS::F64vec v_add = v1 + v2; // add
5   PS::F64vec v_sub = v1 - v2; // sub
6   PS::F64    inner = v1 * v2; // inner product
7   PS::F64vec outer = v1 ^ v2; // outer product
8
9   std::cout << v_add.x << std::endl;
10  // -1.0
11  std::cout << v_dub   << std::endl;
12  //    3    4    -5
```

コンパイル時にマクロを設定する事で、
二次元ベクトルにする事も可能。

粒子クラス



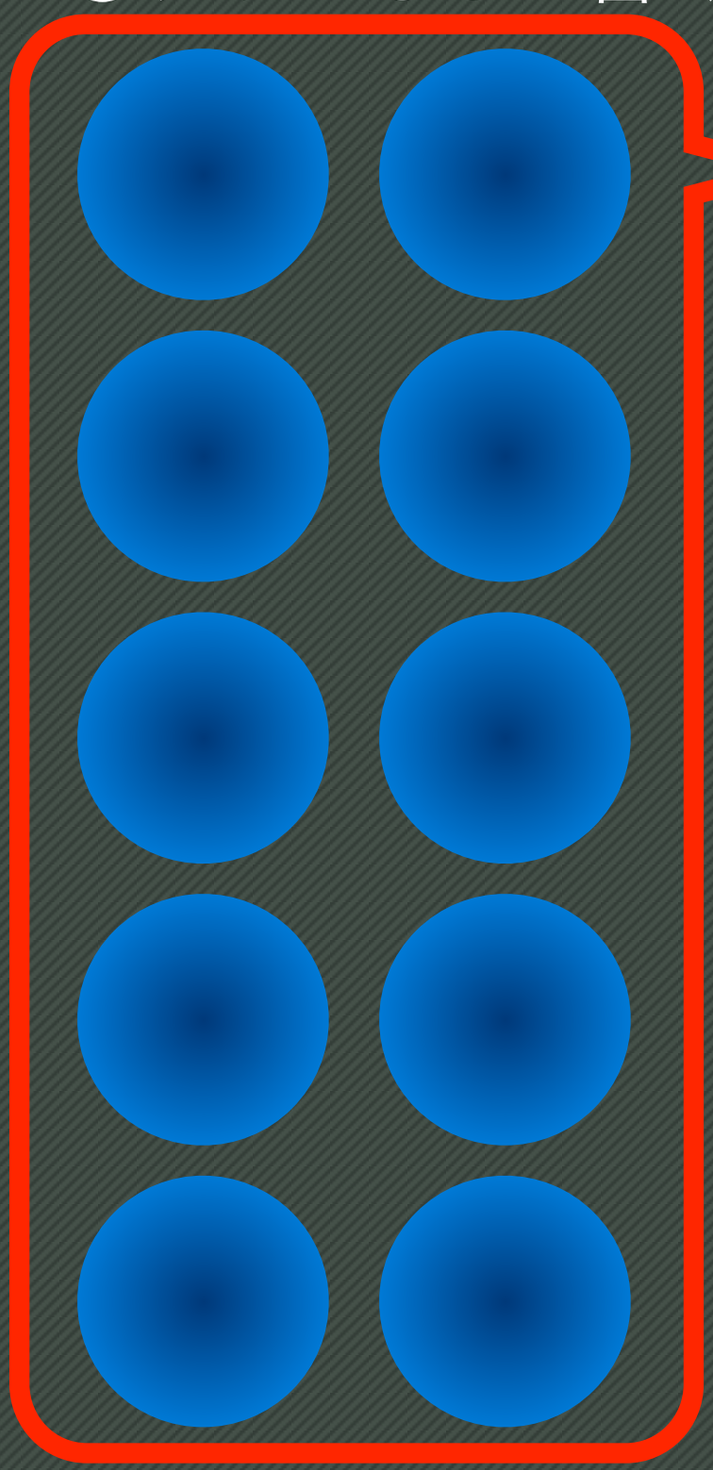
Nbody個の粒子

必要なデータは…

- mass
- position(x, y, z)
- velocity(x, y, z)
- acceleration(x, y, z)

粒子クラス

●クラスなしで書くと…

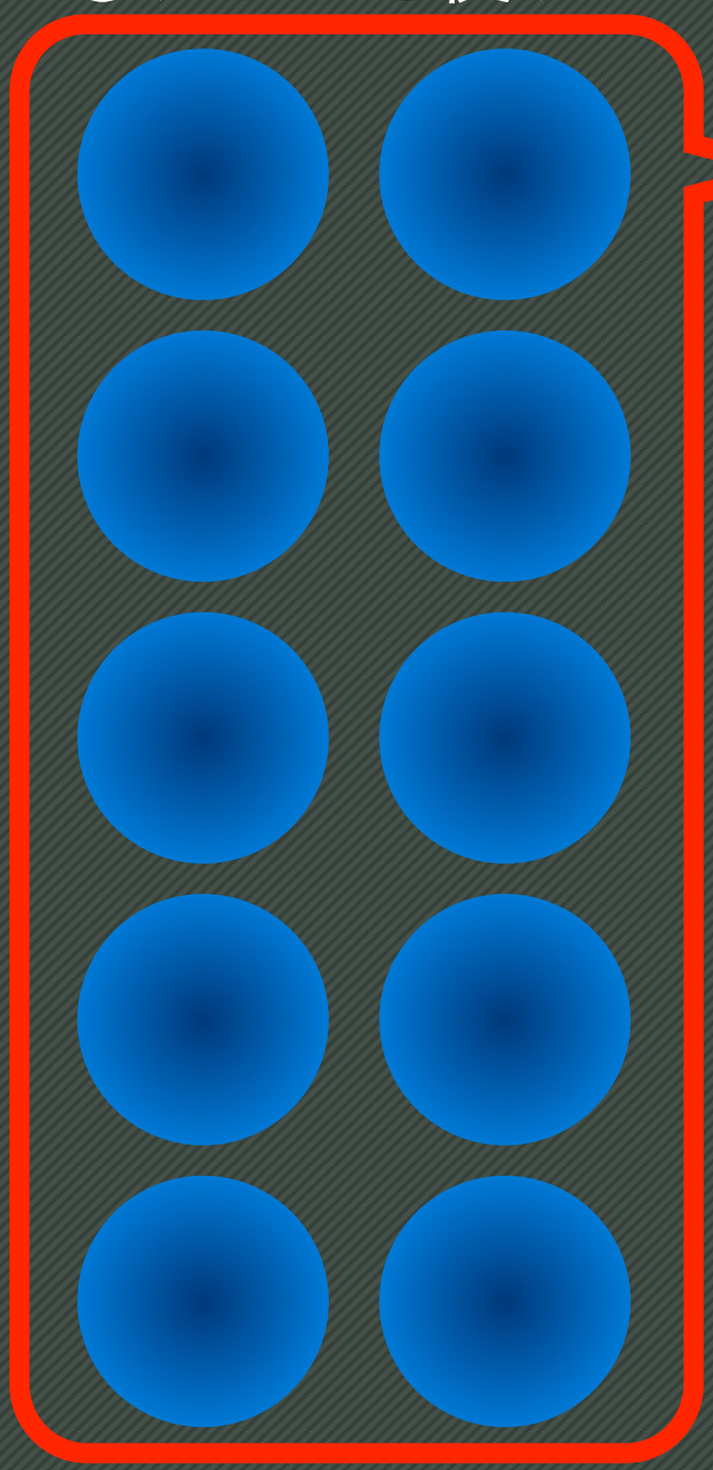


Nbody個の粒子

```
1  double mass[Nbody];
2  double position[Nbody][3];
3  double velocity[Nbody][3];
4  double acceleration[Nbody][3];
```

粒子クラス

●クラスを使うと…



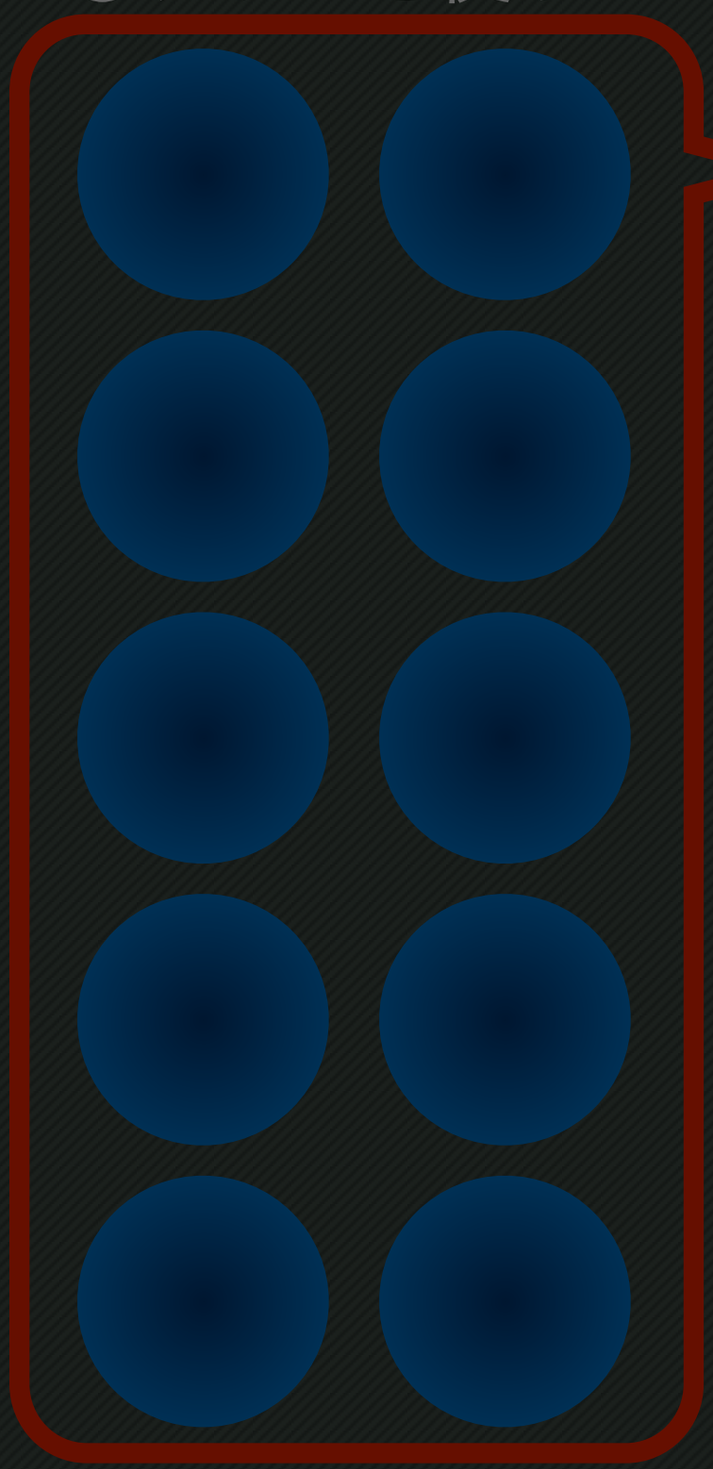
Nbody個の粒子

```
1  class FPGrav{
2    double mass;
3    double position[3];
4    double velocity[3];
5    double acceleration[3];
6  }body[Nbody];
```

```
||  ::::::: ::::::. ::::::. .:::::::.  ||
||  ::      ::    : ::    : ::         ||
||  ::::::  ::    : ::::::'  '::::::.  ||
||  ::      ::::::' ::      '......'    ||
||     Framework for Developing         ||
||       Particle Simulator             ||
\\================================//
```

粒子クラス

●クラスを使うと…



Nbody個の粒子

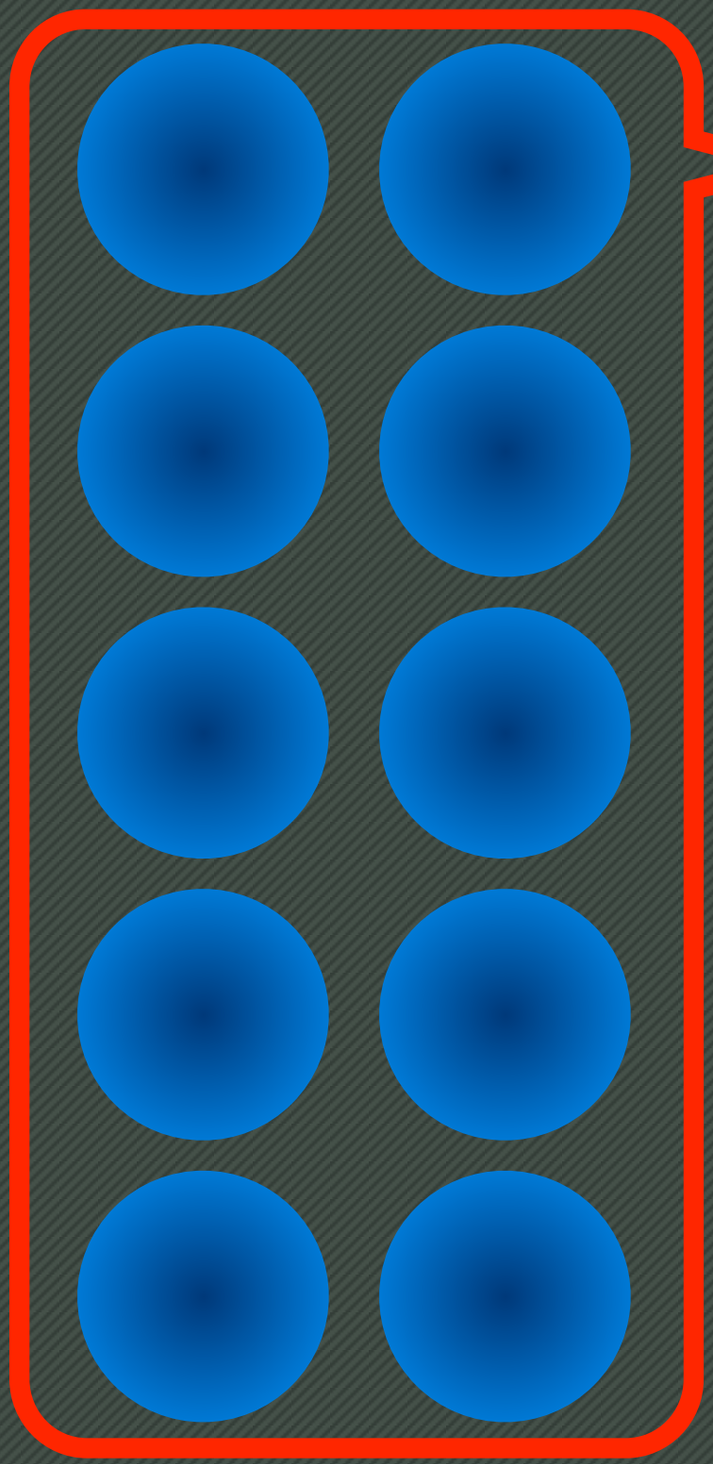
```
1  class FPGrav{
2     double mass;
3     double position[3];
4     double velocity[3];
5     double acceleration[3];
6  }body[Nbody];
```

再掲・クラスのメリット

- クラスを使うとサブルーチンに値を送るときに楽。
- 後からデータ付け足すのも楽。

```
1  //without class
2  void calcSomething(double mass[], double position[][3], double velocity[][3], double acceleration[][3]){
3    //calculate something...
4  }
5
6  //with class
7  void calcSomething(FPGrav particle[]){
8    //calculate something...
9  }
```

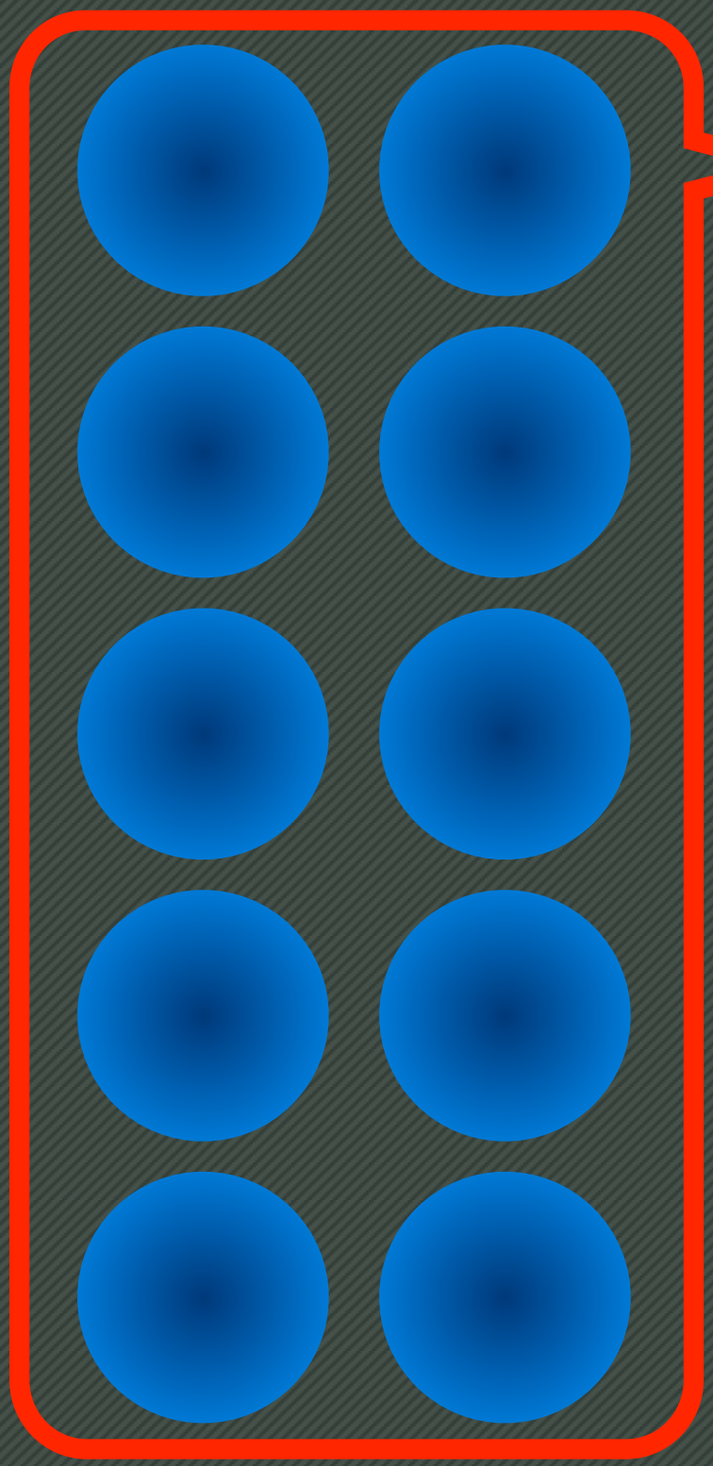

粒子クラス



Nbody個の粒子

●ところで、FDPSは数値計算に便利な
ベクトルクラスを用意してあった。

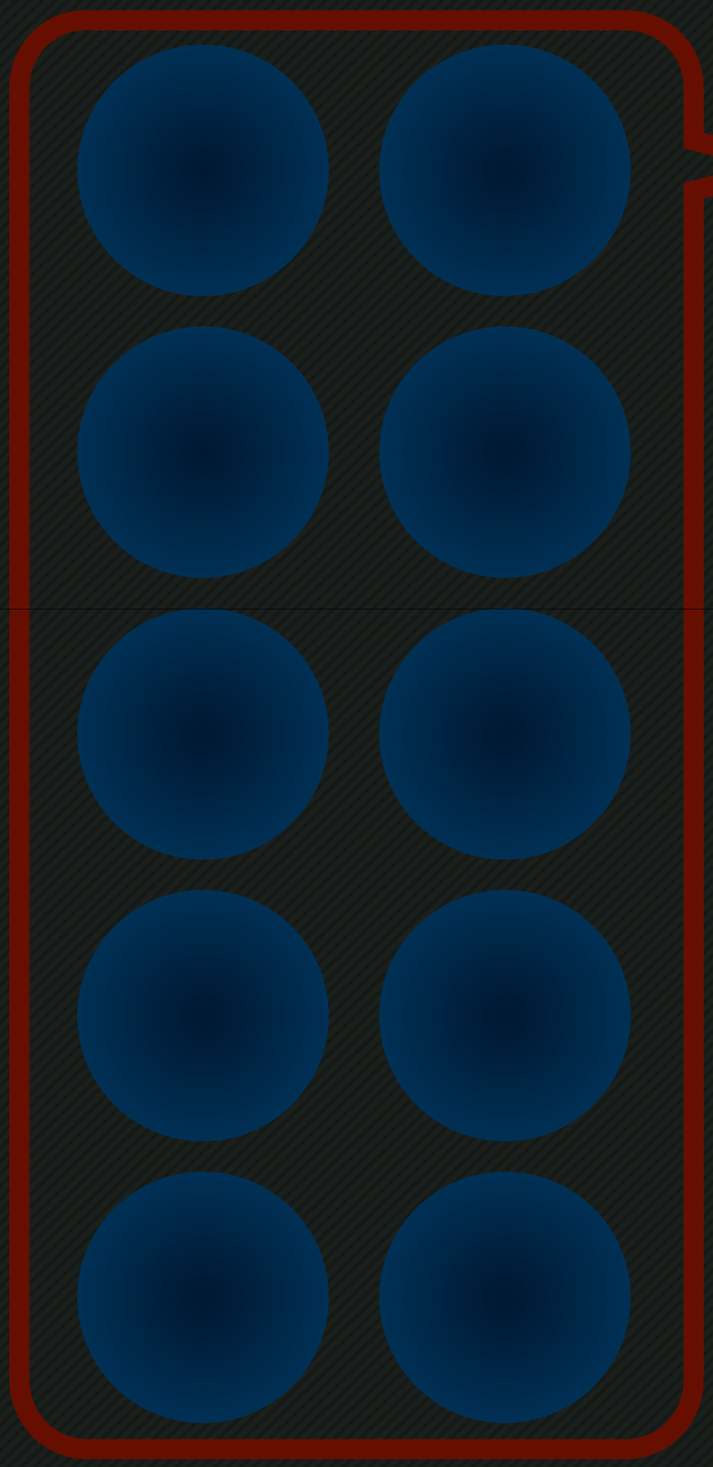
粒子クラス



Nbody個の粒子

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6  }body[Nbody];
```

粒子クラス



Nbody個の粒子

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6  }body[Nbody];
```

クラス内クラスも可能

(2)メンバ関数

member function

2015/07/22 FDPS講習会

メンバ関数とは

●**クラス内の変数**(メンバ変数)**達に対して、**
何らかの処理を加えたい時に記述するもの。

●FDPSを用いる場合、
ユーザーはFDPSとユーザーコード間でデータのやりとりをする
ためのメンバ関数を書く必要がある。

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6    //member functions
7    PS::F64vec getPos(){
8      return position;
9    }
10 }body[Nbody];
11
12 //使い方
13 position = body[0].getPos();
```

メンバ関数とは

- **クラス内の変数**(メンバ変数)**達に対して、**
  何らかの処理を加えたい時に記述するもの。

- FDPSを用いる場合、
  ユーザーはFDPSとユーザーコード間でデータのやりとりをする
  ためのメンバ関数を書く必要がある。

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6    //member functions
7    PS::F64vec getPos(){
8      return position;
9    }
10 }body[Nbody];
11
12 //使い方
13 position = body[0].getPos();
```

メンバ関数とは

- **クラス内の変数**(メンバ変数)**達に対して、**
  何らかの処理を加えたい時に記述するもの。

- FDPSを用いる場合、
  ユーザーはFDPSとユーザーコード間でデータのやりとりをするためのメンバ関数を書く必要がある。

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6    //member functions
7    PS::F64vec getPos(){
8      return position;
9    }
10 }body[Nbody];
11
12 //使い方
13 position = body[0].getPos();
```

(3)テンプレート

template

2015/07/22 FDPS講習会

テンプレート

- **同じ処理を違う型に対して行いたい時に書く**もの。
  クラスをテンプレート化した**クラステンプレート**と、
  関数をテンプレート化した**関数テンプレート**が存在する。

- まずクラステンプレートについて解説し、
  その後関数テンプレートに関して解説する。

2015/07/22 FDPS講習会

クラステンプレート

●FDPSでは、粒子群クラステンプレートの< >の中に
粒子クラスを入れる事で、粒子群クラスの実体が作られる。
同様に、相互作用ツリークラスの実体が作られる。

Name this file…　Language: C++

```
1  //粒子群クラス
2  PS::ParticleSystem<FPGrav> system_grav;
3  //相互作用ツリークラス
4  PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>Monopole tree_grav;
```

クラステンプレート

●FDPSでは、粒子群クラステンプレートの< >の中に
粒子クラスを入れる事で、粒子群クラスの実体が作られる。
同様に、相互作用ツリークラスの実体が作られる。

Name this file…　Language: C++

```
1  //粒子群クラス
2  PS::ParticleSystem<FPGrav> system_grav;
3  //相互作用ツリークラス
4  PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>Monopole tree_grav;
```

<i粒子クラス, j粒子クラス, リザルト>

クラステンプレート

●FDPSでは、粒子群クラステンプレートの< >の中に
粒子クラスを入れる事で、粒子群クラスの実体が作られる。
同様に、相互作用ツリークラスの実体が作られる。

```
1  //粒子群クラス
2  PS::ParticleSystem<FPGrav> system_grav;
3  //相互作用ツリークラス
4  PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>Mo
```

```
1  class FPGrav{
2    PS::F64 mass;
3    PS::F64vec position;
4    PS::F64vec velocity;
5    PS::F64vec acceleration;
6    //member functions
7    PS::F64vec getPos(){
8      return position;
9    }
10 }body[Nbody];
11
12 //使い方
13 position = body[0].getPos(
```

2015/07

関数テンプレート

●FDPSでは相互作用関数は関数テンプレートを用いて記述出来る。

●もちろん、関数テンプレートを用いずに記述する事も可能。
しかし、相互作用が長距離力の場合、
近傍の粒子からの相互作用と、ツリーノードからの相互作用
2つを書かなければならない。

●**関数テンプレートならば1つだけで済む**ので、
今回は関数テンプレートを用いたサンプルコードを紹介する。

関数テンプレート

- FDPSでは、ツリーノードのクラスは既に用意されているため、
  今回はそれ(PS::SPJMonopole)を使う。

- 相互作用関数はSPJMonopoleも使うので、
  getPos();などを用いて書く必要がある。

```
1  template <class TParticleJ>
2  void CalcGravity(const FPGrav * ep_i,
3                   const PS::S32 n_ip,
4                   const TParticleJ * ep_j,
5                   const PS::S32 n_jp,
6                   FPGrav * force) {
7    for(PS::S32 i = 0; i < n_ip; i++){
8      PS::F64vec xi = ep_i[i].getPos();
9      for(PS::S32 j = 0; j < n_jp; j++){
10       PS::F64vec rij    = xi - ep_j[j].getPos();
11       //相互作用計算
12     }
13   }
14 }
```

関数テンプレート

- FDPSでは、ツリーノードのクラスは既に用意されているため、
  今回はそれ(PS::SPJMonopole)を使う。

- 相互作用関数はSPJMonopoleも使うので、
  getPos();などを用いて書く必要がある。

```
1  template <class TParticleJ>
2  void CalcGravity(const FPGrav * ep_i,
3                   const PS::S32 n_ip,
4                   const TParticleJ * ep_j,
5                   const PS::S32 n_jp,
6                   FPGrav * force) {
7    for(PS::S32 i = 0; i < n_ip; i++){
8      PS::F64vec xi = ep_i[i].getPos();
9      for(PS::S32 j = 0; j < n_jp; j++){
10       PS::F64vec rij    = xi - ep_j[j].getPos();
11       //相互作用計算
12     }
13   }
14 }
```

関数テンプレート

- FDPSでは、ツリーノードのクラスは既に用意されているため、
  今回はそれ(PS::SPJMonopole)を使う。

- 相互作用関数はSPJMonopoleも使うので、
  getPos();などを用いて書く必要がある。

```
1  template <class TParticleJ>
2  void CalcGravity(const FPGrav * ep_i,
3                   const PS::S32 n_ip,
4                   const TParticleJ * ep_j,
5                   const PS::S32 n_jp,
6                   FPGrav * force) {
7    for(PS::S32 i = 0; i < n_ip; i++){
8      PS::F64vec xi = ep_i[i].getPos();
9      for(PS::S32 j = 0; j < n_jp; j++){
10       PS::F64vec rij    = xi - ep_j[j].getPos();
11       //相互作用計算
12     }
13   }
14 }
```

コード構成

- ユーザーが書くべきもの

  #include <particle_simulator.h>

  粒子クラスと必要なメンバ関数

  相互作用関数

  時間積分ルーチン

  I/O (粒子クラスのI/Oと、FileHeaderクラス)

サンプルコード

●今回のサンプルコードの内容

重力 (w/ and w/o Phantom-GRAPE (Tanikawa+, 2011; 2012))

時間積分法はleap-frog法

初期条件はその場生成(ファイル読み込みではない)

●ファイル構成

user-defined.hpp

nbody.cpp

user-defined.hpp

```
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
                fscanf(fp, "%lf\n", &time);
                fscanf(fp, "%lld\n", &n_body);
                return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};

class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
    PS::F64vec acc;
    PS::F64    pot;

```

user-defined.hpp

クラス

```
1  class FileHeader{
2  public:
3      PS::S64 n_body;
4      PS::F64 time;
5      PS::S32 readAscii(FILE * fp) {
6                  fscanf(fp, "%lf\n", &time);
7                  fscanf(fp, "%lld\n", &n_body);
8                  return n_body;
9      }
10     void writeAscii(FILE* fp) const {
11         fprintf(fp, "%e\n", time);
12         fprintf(fp, "%lld\n", n_body);
13     }
14 };
15
16 class FPGrav{
17 public:
18     PS::S64    id;
19     PS::F64    mass;
20     PS::F64vec pos;
21     PS::F64vec vel;
22     PS::F64vec acc;
23     PS::F64    pot;
24
```

user-defined.hpp

```
1  class FileHeader{
2  public:
3      PS::S64 n_body;
4      PS::F64 time;
5      PS::S32 readAscii(FILE * fp) {
6                  fscanf(fp, "%lf\n", &time);
7                  fscanf(fp, "%lld\n", &n_body);
8                  return n_body;
9      }
10     void writeAscii(FILE* fp) const {
11         fprintf(fp, "%e\n", time);
12         fprintf(fp, "%lld\n", n_body);
13     }
14 };
15
16 class FPGrav{
17 public:
18     PS::S64    id;
19     PS::F64    mass;
20     PS::F64vec pos;
21     PS::F64vec vel;
22     PS::F64vec acc;
23     PS::F64    pot;
24
```

FileHeaderクラス

```
15
16  class FPGrav{
17  public:
18      PS::S64    id;
19      PS::F64    mass;
20      PS::F64vec pos;
21      PS::F64vec vel;
22      PS::F64vec acc;
23      PS::F64    pot;
24
25      static PS::F64 eps;
26
27      PS::F64vec getPos() const {
28          return pos;
29      }
30
31      PS::F64 getCharge() const {
32          return mass;
33      }
34
35      void copyFromFP(const FPGrav & fp){
36          mass = fp.mass;
37          pos  = fp.pos;
38      }
39
40      void copyFromForce(const FPGrav & force) {
41          acc = force.acc;
42          pot = force.pot;
43      }
44
```

粒子クラス

```
15
16  class FPGrav{
17  public:
18      PS::S64    id;
19      PS::F64    mass;
20      PS::F64vec pos;
21      PS::F64vec vel;
22      PS::F64vec acc;
23      PS::F64    pot;
24
25      static PS::F64 eps;
26
27      PS::F64vec getPos() const {
28          return pos;
29      }
30
31      PS::F64 getCharge() const {
32          return mass;
33      }
34
35      void copyFromFP(const FPGrav & fp){
36          mass = fp.mass;
37          pos  = fp.pos;
38      }
39
40      void copyFromForce(const FPGrav & force) {
41          acc = force.acc;
42          pot = force.pot;
43      }
44
```

粒子物理量

```
23  PS::F64    pot;
24
25  static PS::F64 eps;
26
27  PS::F64vec getPos() const {
28      return pos;
29  }
30
31  PS::F64 getCharge() const {
32      return mass;
33  }
34
35  void copyFromFP(const FPGrav & fp){
36      mass = fp.mass;
37      pos  = fp.pos;
38  }
39
40  void copyFromForce(const FPGrav & force) {
41      acc = force.acc;
42      pot = force.pot;
43  }
44
45  void clear() {
46      acc = 0.0;
47      pot = 0.0;
48  }
49
50      void writeAscii(FILE* fp) const {
51              fprintf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
52              this->id, this->mass,
```

FDPSにデータを渡すための
メンバ関数

```
44
45      void clear() {
46          acc = 0.0;
47          pot = 0.0;
48      }
49
50          void writeAscii(FILE* fp) const {
51                  fprintf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
52                  this->id, this->mass,
53                  this->pos.x, this->pos.y, this->pos.z,
54                  this->vel.x, this->vel.y, this->vel.z);
55          }
56
57          void readAscii(FILE* fp) {
58                  fscanf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
59                 &this->id, &this->mass,
60                  &this->pos.x, &this->pos.y, &this->pos.z,
61                  &this->vel.x, &this->vel.y, &this->vel.z);
62          }
63
64  };
65
66  PS::F64 FPGrav::eps = 1.0/32.0;
67
68  #ifdef ENABLE_PHANTOM_GRAPE_X86
69
70
71  template <class TParticleJ>
72  void CalcGravity(const FPGrav * iptcl,
73                   const PS::S32 ni,
```

I/Oメンバ関数

相互作用
テンプレート関数
(w/ Phantom-GRAPE)

```
        }

};

PS::F64 FPGrav::eps = 1.0/32.0;

#ifdef ENABLE_PHANTOM_GRAPE_X86


template <class TParticleJ>
void CalcGravity(const FPGrav * iptcl,
                 const PS::S32 ni,
                 const TParticleJ * jptcl,
                 const PS::S32 nj,
                 FPGrav * force) {
    const PS::S32 nipipe = ni;
    const PS::S32 njpipe = nj;
    PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
    PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
    PS::F64  *pi     = (PS::F64  *    )malloc(sizeof(PS::F64) * nipipe);
    PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpipe * PS::DIMENSION);
    PS::F64  *mj     = (PS::F64  *    )malloc(sizeof(PS::F64) * njpipe);
    for(PS::S32 i = 0; i < ni; i++) {
        xi[i][0] = iptcl[i].getPos()[0];
        xi[i][1] = iptcl[i].getPos()[1];
        xi[i][2] = iptcl[i].getPos()[2];
        ai[i][0] = 0.0;
        ai[i][1] = 0.0;
        ai[i][2] = 0.0;
        pi[i]    = 0.0;
```

```
        mj[j]    = jptcl[j].mass;
    }
#ifdef PARTICLE_SIMULATOR_THREAD_PARALLEL
    PS::S32 devid = omp_get_thread_num();
#else
    PS::S32 devid = 0;
#endif
    g5_set_xmjMC(devid, 0, nj, xj, mj);
    g5_set_nMC(devid, nj);
    g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
    for(PS::S32 i = 0; i < ni; i++) {
        force[i].acc[0] += ai[i][0];
        force[i].acc[1] += ai[i][1];
        force[i].acc[2] += ai[i][2];
        force[i].pot    -= pi[i];
    }
    free(xi);
    free(ai);
    free(pi);
    free(xj);
    free(mj);
}

#else

template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParticleJ * ep_j,
                 const PS::S32 n_jp,
```

```
126 template <class TParticleJ>
127 void CalcGravity(const FPGrav * ep_i,
128                   const PS::S32 n_ip,
129                   const TParticleJ * ep_j,
130                   const PS::S32 n_jp,
131                   FPGrav * force) {
132     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
133     for(PS::S32 i = 0; i < n_ip; i++){
134         PS::F64vec xi = ep_i[i].getPos();
135         PS::F64vec ai = 0.0;
136         PS::F64 poti = 0.0;
137         for(PS::S32 j = 0; j < n_jp; j++){
138             PS::F64vec rij    = xi - ep_j[j].getPos();
139             PS::F64    r3_inv = rij * rij + eps2;
140             PS::F64    r_inv  = 1.0/sqrt(r3_inv);
141             r3_inv  = r_inv * r_inv;
142             r_inv  *= ep_j[j].getCharge();
143             r3_inv *= r_inv;
144             ai     -= r3_inv * rij;
145             poti   -= r_inv;
146         }
147         force[i].acc += ai;
148         force[i].pot += poti;
149     }
150 }
151
152 #endif
```

相互作用

テンプレート関数

(w/o Phantom-GRAPE)

```
126 template <class TParticleJ>
127 void CalcGravity(const FPGrav * ep_i,
128                   const PS::S32 n_ip,
129                   const TParticleJ * ep_j,
130                   const PS::S32 n_jp,
131                   FPGrav * force) {
132     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
133     for(PS::S32 i = 0; i < n_ip; i++){
134         PS::F64vec xi = ep_i[i].getPos();
135         PS::F64vec ai = 0.0;
136         PS::F64 poti = 0.0;
137         for(PS::S32 j = 0; j < n_jp; j++){
138             PS::F64vec rij    = xi - ep_j[j].getPos();
139             PS::F64    r3_inv = rij * rij + eps2;
140             PS::F64    r_inv  = 1.0/sqrt(r3_inv);
141             r3_inv  = r_inv * r_inv;
142             r_inv  *= ep_j[j].getCharge();
143             r3_inv *= r_inv;
144             ai     -= r3_inv * rij;
145             poti   -= r_inv;
146         }
147         force[i].acc += ai;
148         force[i].pot += poti;
149     }
150 }
151
152 #endif
```

相互作用
テンプレート関数
(w/o Phantom-GRAPE)

```
126 template <class TParticleJ>
127 void CalcGravity(const FPGrav * ep_i,
128                  const PS::S32 n_ip,
129                  const TParticleJ * ep_j,
130                  const PS::S32 n_jp,
131                  FPGrav * force) {
132     PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
133     for(PS::S32 i = 0; i < n_ip; i++){
134         PS::F64vec xi = ep_i[i].getPos();
135         PS::F64vec ai = 0.0;
136         PS::F64 poti = 0.0;
137         for(PS::S32 j = 0; j < n_jp; j++){
138             PS::F64vec rij    = xi - ep_j[j].getPos();
139             PS::F64    r3_inv = rij * rij + eps2;
140             PS::F64    r_inv  = 1.0/sqrt(r3_inv);
141             r3_inv  = r_inv * r_inv;
142             r_inv  *= ep_j[j].getCharge();
143             r3_inv *= r_inv;
144             ai     -= r3_inv * rij;
145             poti   -= r_inv;
146         }
147         force[i].acc += ai;
148         force[i].pot += poti;
149     }
150 }
151
152 #endif
```

相互作用
テンプレート関数
(w/o Phantom-GRAPE)

nbody.cpp

327 lines (294 sloc) | 10.569 kB

Raw | Bla

```
1  #include<iostream>
2  #include<fstream>
3  #include<unistd.h>
4  #include<sys/stat.h>
5  #include<particle_simulator.hpp>
6  #ifdef ENABLE_PHANTOM_GRAPE_X86
7  #include <gp5util.h>
8  #endif
9  #include "user-defined.hpp"
10
11 void makeColdUniformSphere(const PS::F64 mass_glb,
12                            const PS::S64 n_glb,
13                            const PS::S64 n_loc,
14                            PS::F64 *& mass,
15                            PS::F64vec *& pos,
16                            PS::F64vec *& vel,
17                            const PS::F64 eng = -0.25,
18                            const PS::S32 seed = 0) {
19
20     assert(eng < 0.0);
21     {
22         PS::MTTS mt;
23         mt.init_genrand(0);
24         for(PS::S32 i = 0; i < n_loc; i++){
25             mass[i] = mass_glb / n_glb;
```

nbody.cpp

327 lines (294 sloc) | 10.569 kB

Raw | Bla

```
1  #include<iostream>
2  #include<fstream>
3  #include<unistd.h>
4  #include<sys/stat.h>
5  #include<particle_simulator.hpp>
6  #ifdef ENABLE_PHANTOM_GRAPE_X86
7  #include <gp5util.h>
8  #endif
9  #include "user-defined.hpp"
10
11 void makeColdUniformSphere(const PS::F64 mass_glb,
12                            const PS::S64 n_glb,
13                            const PS::S64 n_loc,
14                            PS::F64 *& mass,
15                            PS::F64vec *& pos,
16                            PS::F64vec *& vel,
17                            const PS::F64 eng = -0.25,
18                            const PS::S32 seed = 0) {
19
20     assert(eng < 0.0);
21     {
22         PS::MTTS mt;
23         mt.init_genrand(0);
24         for(PS::S32 i = 0; i < n_loc; i++){
25             mass[i] = mass_glb / n_glb;
```

FDPSをincludeする

```
79  template<class Tpsys>
80  void kick(Tpsys & system,
81            const PS::F64 dt) {
82      PS::S32 n = system.getNumberOfParticleLocal();
83      for(PS::S32 i = 0; i < n; i++) {
84          system[i].vel  += system[i].acc * dt;
85      }
86  }
87
88  template<class Tpsys>
89  void drift(Tpsys & system,
90             const PS::F64 dt) {
91      PS::S32 n = system.getNumberOfParticleLocal();
92      for(PS::S32 i = 0; i < n; i++) {
93          system[i].pos  += system[i].vel * dt;
94      }
95  }
96
97  template<class Tpsys>
98  void calcEnergy(const Tpsys & system,
99                  PS::F64 & etot,
100                 PS::F64 & ekin,
101                 PS::F64 & epot,
102                 const bool clear=true){
103     if(clear){
104         etot = ekin = epot = 0.0;
105     }
106     PS::F64 etot_loc = 0.0;
107     PS::F64 ekin_loc = 0.0;
```

leap-frog法による
時間積分

```
78
79  template<class Tpsys>
80  void kick(Tpsys & system,
81            const PS::F64 dt) {
82      PS::S32 n = system.getNumberOfParticleLocal();
83      for(PS::S32 i = 0; i < n; i++) {
84          system[i].vel  += system[i].acc * dt;
85      }
86  }
87
88  template<class Tpsys>
89  void drift(Tpsys & system,
90             const PS::F64 dt) {
91      PS::S32 n = system.getNumberOfParticleLocal();
92      for(PS::S32 i = 0; i < n; i++) {
93          system[i].pos  += system[i].vel * dt;
94      }
95  }
96
97  template<class Tpsys>
98  void calcEnergy(const Tpsys & system,
99                  PS::F64 & etot,
100                 PS::F64 & ekin,
101                 PS::F64 & epot,
102                 const bool clear=true){
103     if(clear){
104         etot = ekin = epot = 0.0;
105     }
106     PS::F64 etot_loc = 0.0;
107     PS::F64 ekin_loc = 0.0;
```

getNumberOfParticleLocal()
で粒子数が取得できる

```
78
79  template<class Tpsys>
80  void kick(Tpsys & system,
81            const PS::F64 dt) {
82      PS::S32 n = system.getNumberOfParticleLocal();
83      for(PS::S32 i = 0; i < n; i++) {
84          system[i].vel  += system[i].acc * dt;
85      }
86  }
87
88  template<class Tpsys>
89  void drift(Tpsys & system,
90             const PS::F64 dt) {
91      PS::S32 n = system.getNumberOfParticleLocal();
92      for(PS::S32 i = 0; i < n; i++) {
93          system[i].pos  += system[i].vel * dt;
94      }
95  }
96
97  template<class Tpsys>
98  void calcEnergy(const Tpsys & system,
99                  PS::F64 & etot,
100                 PS::F64 & ekin,
101                 PS::F64 & epot,
102                 const bool clear=true){
103     if(clear){
104         etot = ekin = epot = 0.0;
105     }
106     PS::F64 etot_loc = 0.0;
107     PS::F64 ekin_loc = 0.0;
```

粒子群クラスに[i]をつけると
i粒子のデータが取得できる

メイン関数開始

```
158
159 int main(int argc, char *argv[]) {
160     std::cout<<std::setprecision(15);
161     std::cerr<<std::setprecision(15);
162
163     PS::Initialize(argc, argv);
164     PS::F32 theta = 0.5;
165     PS::S32 n_leaf_limit = 8;
166     PS::S32 n_group_limit = 64;
167     PS::F32 time_end = 10.0;
168     PS::F32 dt = 1.0 / 128.0;
169     PS::F32 dt_diag = 1.0 / 8.0;
170     PS::F32 dt_snap = 1.0;
171     char dir_name[1024];
172     PS::S64 n_tot = 1024;
173     PS::S32 c;
174     sprintf(dir_name,"./result");
175     opterr = 0;
176     while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
177         switch(c){
178         case 'o':
179             sprintf(dir_name,optarg);
180             break;
181         case 't':
182             theta = atof(optarg);
183             std::cerr << "theta =" << theta << std::endl;
184             break;
185         case 'T':
186             time_end = atof(optarg);
187             std::cerr << "time_end = " << time_end << std::endl;
```

```
158
159 int main(int argc, char *argv[]) {
160     std::cout<<std::setprecision(15);
161     std::cerr<<std::setprecision(15);
162
163     PS::Initialize(argc, argv);
164     PS::F32 theta = 0.5;
165     PS::S32 n_leaf_limit = 8;
166     PS::S32 n_group_limit = 64;
167     PS::F32 time_end = 10.0;
168     PS::F32 dt = 1.0 / 128.0;
169     PS::F32 dt_diag = 1.0 / 8.0;
170     PS::F32 dt_snap = 1.0;
171     char dir_name[1024];
172     PS::S64 n_tot = 1024;
173     PS::S32 c;
174     sprintf(dir_name,"./result");
175     opterr = 0;
176     while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
177         switch(c){
178         case 'o':
179             sprintf(dir_name,optarg);
180             break;
181         case 't':
182             theta = atof(optarg);
183             std::cerr << "theta =" << theta << std::endl;
184             break;
185         case 'T':
186             time_end = atof(optarg);
187             std::cerr << "time_end = " << time_end << std::endl;
```

FDPS初期化

```
while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
    switch(c){
    case 'o':
        sprintf(dir_name,optarg);
        break;
    case 't':
        theta = atof(optarg);
        std::cerr << "theta =" << theta << std::endl;
        break;
    case 'T':
        time_end = atof(optarg);
        std::cerr << "time_end = " << time_end << std::endl;
        break;
    case 's':
        dt = atof(optarg);
        std::cerr << "time_step = " << dt << std::endl;
        break;
    case 'd':
        dt_diag = atof(optarg);
        std::cerr << "dt_diag = " << dt_diag << std::endl;
        break;
    case 'D':
        dt_snap = atof(optarg);
        std::cerr << "dt_snap = " << dt_snap << std::endl;
        break;
    case 'l':
        n_leaf_limit = atoi(optarg);
        std::cerr << "n_leaf_limit = " << n_leaf_limit << std::endl;
        break;
```

コマンドライン
引数の解析

```
200             break;
201         case 'l':
202             n_leaf_limit = atoi(optarg);
203             std::cerr << "n_leaf_limit = " << n_leaf_limit << std::endl;
204             break;
205         case 'n':
206             n_group_limit = atoi(optarg);
207             std::cerr << "n_group_limit = " << n_group_limit << std::endl;
208             break;
209         case 'N':
210             n_tot = atoi(optarg);
211             std::cerr << "n_tot = " << n_tot << std::endl;
212             break;
213         case 'h':
214             if(PS::Comm::getRank() == 0) {
215                 printHelp();
216             }
217             PS::Finalize();
218             return 0;
219         default:
220             if(PS::Comm::getRank() == 0) {
221                 std::cerr<<"No such option! Available options are here."<<std::endl;
222                 printHelp();
223             }
224             PS::Abort();
225         }
226     }
227
228     makeOutputDirectory(dir_name);
229
```

```
239 }
240
241 PS::ParticleSystem<FPGrav> system_grav;
242 system_grav.initialize();
243 PS::S32 n_loc    = 0;
244 PS::F32 time_sys = 0.0;
245 if(PS::Comm::getRank() == 0) {
246     setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
247 } else {
248     system_grav.setNumberOfParticleLocal(n_loc);
249 }
250
251 const PS::F32 coef_ema = 0.3;
252 PS::DomainInfo dinfo;
253     dinfo.initialize(coef_ema);
254     dinfo.collectSampleParticle(system_grav);
255     dinfo.decomposeDomain();
256     system_grav.exchangeParticle(dinfo);
257     n_loc = system_grav.getNumberOfParticleLocal();
258
259 #ifdef ENABLE_PHANTOM_GRAPE_X86
260     g5_open();
261     g5_set_eps_to_all(FPGrav::eps);
262 #endif
263
264     PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
265     tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
266     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
267                                        CalcGravity<PS::SPJMonopole>,
```

粒子群クラスの
生成・初期化

```
239     }
240
241     PS::ParticleSystem<FPGrav> system_grav;
242     system_grav.initialize();
243     PS::S32 n_loc    = 0;
244     PS::F32 time_sys = 0.0;
245     if(PS::Comm::getRank() == 0) {
246         setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
247     } else {
248         system_grav.setNumberOfParticleLocal(n_loc);
249     }
250
251     const PS::F32 coef_ema = 0.3;
252     PS::DomainInfo dinfo;
253     dinfo.initialize(coef_ema);
254
255     dinfo.decomposeDomainAll(system_grav);
256     system_grav.exchangeParticle(dinfo);
257     n_loc = system_grav.getNumberOfParticleLocal();
258
259 #ifdef ENABLE_PHANTOM_GRAPE_X86
260     g5_open();
261     g5_set_eps_to_all(FPGrav::eps);
262 #endif
263
264     PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
265     tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
266     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
267                                        CalcGravity<PS::SPJMonopole>,
```

ドメイン情報の
生成・初期化
領域分割

```
239     }
240
241     PS::ParticleSystem<FPGrav> system_grav;
242     system_grav.initialize();
243     PS::S32 n_loc    = 0;
244     PS::F32 time_sys = 0.0;
245     if(PS::Comm::getRank() == 0) {
246         setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
247     } else {
248         system_grav.setNumberOfParticleLocal(n_loc);
249     }
250
251     const PS::F32 coef_ema = 0.3;
252     PS::DomainInfo dinfo;
253     dinfo.initialize(coef_ema);
254
255     dinfo.decomposeDomainAll(system_grav);
256     system_grav.exchangeParticle(dinfo);
257     n_loc = system_grav.getNumberOfParticleLocal();
258
259 #ifdef ENABLE_PHANTOM_GRAPE_X86
260     g5_open();
261     g5_set_eps_to_all(FPGrav::eps);
262 #endif
263
264     PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
265     tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
266     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
267                                        CalcGravity<PS::SPJMonopole>,
```

粒子交換

相互作用ツリークラスの
生成・初期化

```
263
264 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
265 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
266 tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
267                                    CalcGravity<PS::SPJMonopole>,
268                                    system_grav,
269                                    dinfo);
270 PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
271 calcEnergy(system_grav, Etot0, Ekin0, Epot0);
272 PS::F64 time_diag = 0.0;
273 PS::F64 time_snap = 0.0;
274 PS::S64 n_loop = 0;
275 PS::S32 id_snap = 0;
276 while(time_sys < time_end){
277     if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
278         char filename[256];
279         sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
280         FileHeader header;
281         header.time   = time_sys;
282         header.n_body = system_grav.getNumberOfParticleGlobal();
283         system_grav.writeParticleAscii(filename, header);
284         time_snap += dt_snap;
285     }
286
287     calcEnergy(system_grav, Etot1, Ekin1, Epot1);
288
289     if(PS::Comm::getRank() == 0){
290         if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys)
291             fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 << std::endl;
292             fprintf(stderr, "time: %10.7f energy error: %+e\n",
```

```
263
264 PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
265 tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
266 tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
267                                    CalcGravity<PS::SPJMonopole>,
268                                    system_grav,
269                                    dinfo);
270 PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
271 calcEnergy(system_grav, Etot0, Ekin0, Epot0);
272 PS::F64 time_diag = 0.0;
273 PS::F64 time_snap = 0.0;
274 PS::S64 n_loop = 0;
275 PS::S32 id_snap = 0;
276 while(time_sys < time_end){
277     if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
278         char filename[256];
279         sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
280         FileHeader header;
281         header.time   = time_sys;
282         header.n_body = system_grav.getNumberOfParticleGlobal();
283         system_grav.writeParticleAscii(filename, header);
284         time_snap += dt_snap;
285     }
286
287     calcEnergy(system_grav, Etot1, Ekin1, Epot1);
288
289     if(PS::Comm::getRank() == 0){
290         if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys)
291             fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 << std::endl;
292             fprintf(stderr, "time: %10.7f energy error: %+e\n",
```

相互作用関数を用いた
力の計算

時間積分

```
PS::F64 time_diag = 0.0;
PS::F64 time_snap = 0.0;
PS::S64 n_loop = 0;
PS::S32 id_snap = 0;
while(time_sys < time_end){
    if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
        char filename[256];
        sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
        FileHeader header;
        header.time   = time_sys;
        header.n_body = system_grav.getNumberOfParticleGlobal();
        system_grav.writeParticleAscii(filename, header);
        time_snap += dt_snap;
    }

    calcEnergy(system_grav, Etot1, Ekin1, Epot1);

    if(PS::Comm::getRank() == 0){
        if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys)
            fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 << std::endl;
            fprintf(stderr, "time: %10.7f energy error: %+e\n",
                    time_sys, (Etot1 - Etot0) / Etot0);
            time_diag += dt_diag;
        }
    }


    kick(system_grav, dt * 0.5);

    time_sys += dt;
```

```
298
299     kick(system_grav, dt * 0.5);
300
301     time_sys += dt;
302     drift(system_grav, dt);
303
304     if(n_loop % 4 == 0){
305         dinfo.decomposeDomainAll(system_grav);
306     }
307
308     system_grav.exchangeParticle(dinfo);
309
310     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
311                                        CalcGravity<PS::SPJMonopole>,
312                                        system_grav,
313                                        dinfo);
314
315         kick(system_grav, dt * 0.5);
316
317         n_loop++;
318     }
319
320 #ifdef ENABLE_PHANTOM_GRAPE_X86
321     g5_close();
322 #endif
323
324     PS::Finalize();
325     return 0;
326 }
```

時間積分終了

```
298
299     kick(system_grav, dt * 0.5);
300
301     time_sys += dt;
302     drift(system_grav, dt);
303
304     if(n_loop % 4 == 0){
305         dinfo.decomposeDomainAll(system_grav);
306     }
307
308     system_grav.exchangeParticle(dinfo);
309
310     tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
311                                        CalcGravity<PS::SPJMonopole>,
312                                        system_grav,
313                                        dinfo);
314
315         kick(system_grav, dt * 0.5);
316
317         n_loop++;
318     }
319
320 #ifdef ENABLE_PHANTOM_GRAPE_X86
321     g5_close();
322 #endif
323
324     PS::Finalize();
325     return 0;
326 }
```

FDPS終了

```
298
299         kick(system_grav, dt * 0.5);
300
301         time_sys += dt;
302         drift(system_grav, dt);
303
304         if(n_loop % 4 == 0){
305             dinfo.decomposeDomainAll(system_grav);
306         }
307
308         system_grav.exchangeParticle(dinfo);
309
310         tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
311                                            CalcGravity<PS::SPJMonopole>,
312                                            system_grav,
313                                            dinfo);
314
315         kick(system_grav, dt * 0.5);
316
317         n_loop++;
318     }
319
320 #ifdef ENABLE_PHANTOM_GRAPE_X86
321     g5_close();
322 #endif
323
324     PS::Finalize();
325     return 0;
326 }
```

326行！

最後に

●ユーザーが書かなければならないのは大体これくらい。

→重力の場合は326+152行で終わる。

●コード内に並列化を意識するようなところは無かった。

→**コンパイルの仕方だけでOpenMPやMPIを切り替えられる。**

2015/07/22 FDPS講習会

実習の流れ

- 詳しくはFDPS講習会の手引を御覧ください。
  (http://www.jmlab.jp/?p=530)
- 実習用のFOCUSスパコンにログイン
  サンプルコードを
  (1)並列化無し (2)OpenMP (3)OpenMP + MPI
  の3パターンについでコンパイル・実行
  【計算内容】
  重力
  cold collapse
  流体 (Smoothed Particle Hydrodynamics法)
  adiabatic sphere collapse
- 結果の解析