FDPS 新機能について

牧野淳一郎
神戸大学理学研究科惑星学専攻
および 理化学研究所 計算科学研究センター
兼 粒子系シミュレータ開発チーム
＋ FDPS 開発チーム (特に野村昴太郎)

2021/9/9 FDPS 講習会

話の構成

- 新機能追加の歴史
- **PIKG** (カーネル「自動」生成)
- 極座標/円筒座標対応
- 高度なシミュレーションコードの紹介

新機能追加の歴史

- **2016/1 V2.0:** GPU 等アクセラレータ対応
- **2016/12 V3.0:** Fortran インターフェース
- **2017/11 V4.0:** 細かい新機能
- **2018/11 V5.0:** C 言語インターフェース (他に SPH+N 体コードサンプル)
- **2020/8 6.0:** PIKG インターフェース (カーネルコード自動生成)
- **2021/8 7.0:** 円筒/極座標対応 (リング/ディスクコード用)

PIKG (カーネル「自動」生成)

- 何故こういうものが欲しいか？
- どういう考え方で作ってあるか？
- PIKG 言語仕様
- どうやって使うか？

何故こういうものが欲しいか？

- **FDPS** 使えば並列化はやってくれて結構性能がでる
- 一方、相互作用計算関数は「自分で書く」必要あり
  - **x86** の重力なら **Phantom-GRAPE** がある
  - **GPU(Cuda)** の重力なら サンプルコードもある
  - それ以外で性能出すのは？重力でも富岳とかもっと新しいプロセッサ・命令セットが、、、
- 相互作用関数の中身を一つ書いたら色々なマシンでちゃんと速く走ってくれるとすごく嬉しい。

PIKG (Particle-particle Interaction Kernel Generator)

どういう考え方で作ってあるか？

- ユーザーは、「相互作用本体」を、力を及ぼす粒子 (**EPJ**)、力をうける粒子 (**EPI**)、の物理量 (というか変数) を使って書く。構造体同士のイメージ。
- それから **AVX(2,512)**、**Cuda**、富岳向けイントリンシック、**FDPS** とのインターフェース関数を生成する「ドメイン特化言語 (**DSL**) コンパイラ」
- コンパイラといってもアセンブリコードを生成するわけではなく (今後するかも)、イントリンシックを使ったコードや **Cuda** コードを生成。

PIKG 言語仕様

FDPS sample/c++/nbody の例にそって

```
EPI F32vec xi:pos    i 粒子変数の型、名前。 pos: FDPS の構造体で pos。

EPJ F32vec xj:pos    j 粒子変数の型、名前。 pos: FDPS の構造体で pos。
EPJ F32    mj:mass   j 粒子変数の型、名前。 mass: FDPS の構造体で mass。

FORCE F32vec acc:acc 力を返す構造体での変数の型、名前。
FORCE F32    pot:pot 力を返す構造体での変数の型、名前。

F32 eps2             グローバル変数も宣言できる。値は呼ぶ時に

rij    = xi - xj     ベクトルの計算できる。結果の型は推論される
r2 = rij * rij + eps2
r_inv  = rsqrt(r2)   逆数平方根関数はある
r2_inv = r_inv * r_inv
mr_inv  = mj * r_inv
mr3_inv = r2_inv * mr_inv
acc -= mr3_inv * rij FORCE 型には積算できる
pot -= mr_inv
```

PIKG 変数型

- **F32/F64**：浮動小数点数 (**float, double** に相当)
- **S32/S64**：符号付き整数 (**int32t, int64t** に相当)
- **U32/U64**：符号なし整数 (**uint32t, uint64t** に相当)
- **F32vec/F64vec**：浮動小数点数ベクトル型 (**3** 次元)
- **F32vec2/F64vec2**：浮動小数点数ベクトル型 (**2** 次元)
- **F32vec3/F64vec3**：浮動小数点数ベクトル型 (**3** 次元)
- **F32vec4/F64vec4**：浮動小数点数ベクトル型 (**4** 次元)

FDPS の **PS:F64** とかに対応

演算子、定義済み関数

`+ - / *`	: 四則演算
`&& ||`	: 論理演算
`== != < > <= >=`	: 等号・比較
`( )`	: 括弧
`[ ]`	: 配列アクセス

sqrt : 平方根、 rsqrt : 逆数平方根、 inv : 逆数、 max / min : 最大/最小

関数、条件分岐

関数

```
function muladd(a,b,c)  型は推論される
   ret = a*b + c
   return ret           返り値は必須。
end
```

条件分岐

```
if ...
  ...
[elsif  ...     elsif ブロックは不要ならなしで
   ... ]
endif
```

SIMD の時適切なマスクとかで上手くやってくれる

どうやって使うか？

```
$(PIKG)/bin/pikg [options] -i INPUT -o OUTPUT
```

オプション

- **conversion-type ARCH : reference/AVX2/AVX-512/A64FX** でターゲットアーキテクチャを指定
- **epi-name NAME : EPI** クラスの **C++** での名前を指定
- **epj-name NAME : EPJ** クラスの **C++** での名前を指定
- **force-name NAME : FORCE** クラスの (以下略)

- コード書く、使うのは **sample/c++/nbody/Makefile** とそこのコードを参考にするのが吉
- この サンプルには **Cuda** のコードだして **GPU** 使う例も

性能

こんな感じ

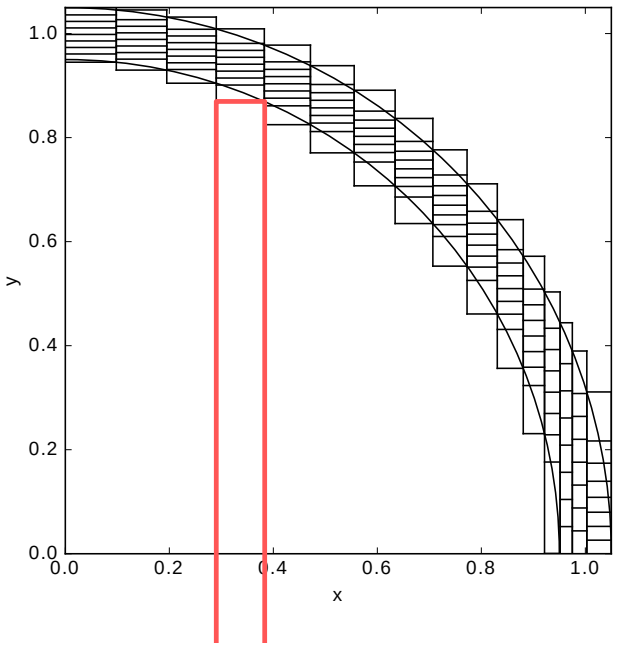
- 重力:
  - 吉川版 **PhantomGRAPE (AVX512)** と同等 (粒子数とかの条件によっては速いとか)
  - **A64fx** で理論ピークの **30%**くらい。似鳥版ハンドチューニングには今のところ負けている (コード生成の考え方から違う)
- **SPH:** 富岳で書いて複雑なものがちゃんと動いた。効率 **20%**くらい。

(富岳でこれくらいでるのはすごく偉大)

極座標/円筒座標対応

- 何故必要か
- 実装の考え方
- 使い方

何故必要か

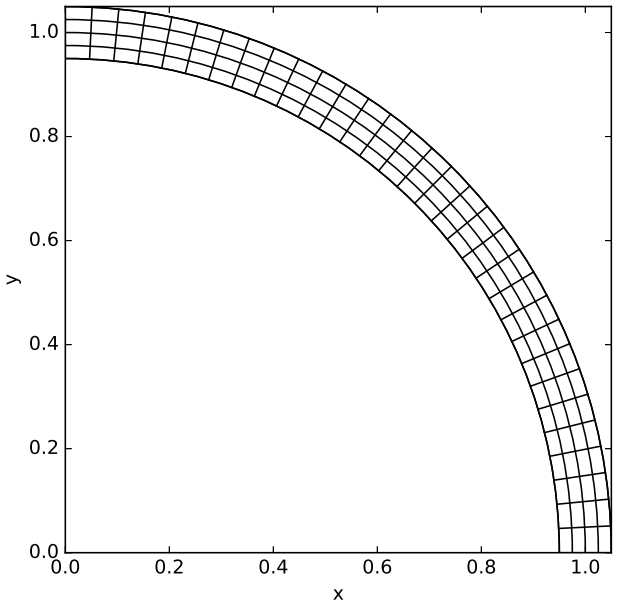


細いリングを **FDPS** の標準の方法で領域分割するとすごく具合の悪いことが起こる

- 細長い領域
- 象限にまたがったものすごく大きな領域

通信が増えて計算効率が落ちる。

解決方法



円柱座標なり極座標なりで領域分割すれば解決。領域はほぼ長方形。正方形にもできる

- 通信が最小化
- 座標系回転させてさらに粒子の移動を減らすことも

実装の考え方

- 領域分割、ツリー構築、ツリー辿る：極座標または円筒座標で
  - 非常に幅が広いリング：半径方向を対数座標で
  - 局所的にはカーテシアンなので、粒子近傍でツリー辿ることはできる
  - 角度方向だけ周期境界
- ツリーの物理量、相互作用計算はカーテシアンで
- (現在のところ) 後者をユーザーコードで対応。
  - **FDPS** の `get_pos` には極座標
  - 相互作用計算カーネルには別に物理量を
  - **4** 重極計算にはそれ用の **FDPS** の「モーメントクラス」追加する必要あり。**sample/c++/planetary_ring** にサンプルあり。

使い方

割とややこしいので、**sample/c++/planetary_ring** をみて下さい、、、

効果

- ノード数がすごく多い時には、「動くか動かないか」くらい違う：象限にまたがった領域とかがなくなるため
- ノード数少ないとまあまあんまり、、、

高度なシミュレーションコードの紹介

- **GPLUM:** 惑星形成シミュレーション用 N 体コード
  https://github.com/YotaIshigaki/GPLUM
  - **P^3T** スキーム使ってて速い。富岳でも (なんかとまる問題はあるが) 動く。
- **PeTar:** 球状星団シミュレーション用 N 体コード
  https://github.com/lwang-astro/PeTar
  - **P^3T** スキーム使ってて速い。球状星団で連星の扱いもちゃんとできて大規模並列化できてるのは世界でこれだけ
- **FDPS_SPH:** 巨大衝突とかできる自己重力 SPH コード
  https://github.com/NatsukiHosono/FDPS_SPH
- **planetary_ring:** 大粒子数・大規模並列でちゃんと動くグローバルな惑星リングコード **FDPS 7.0** の **sample/c++/planetary_ring**

それぞれ、世界トップクラスの計算ができるコードなので、目的にあってればこれらの利用も御検討をみたいな。

DEM コードもそのうちに。