計算惑星科学における粒子法

牧野淳一郎
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
コデザイン推進チーム
兼 粒子系シミュレータ開発チーム

2015/05/19 CPSセミナー

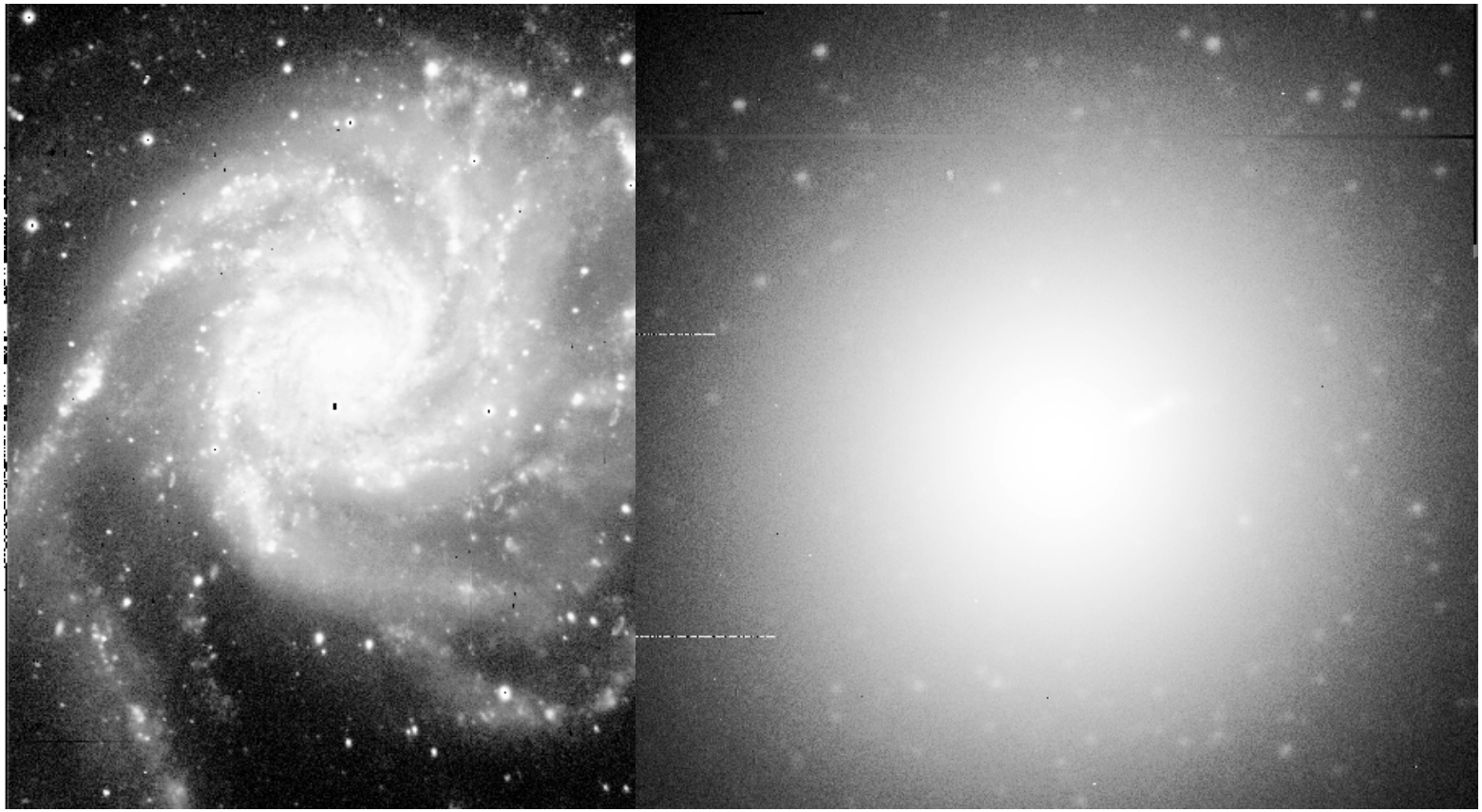
話の構成

- 数値スキームの話題いくつか
  1. 天体物理・惑星科学における粒子法の利用例
  2. 問題点は？
  3. 粒子法と陰解法
     - 原理
     - 実例 1: 音速抑制法
     - 実例 2: 慣性変化法
     - その他の可能性
  4. 「疑似密度法」
- (時間あれば)「粒子法の大規模並列化フレームワーク」
  1. どんなことをしたい (あるいはしたくない) か
  2. ではどうするか
  3. 実装方針
  4. 現状

天体物理・惑星科学における粒子法の利用例

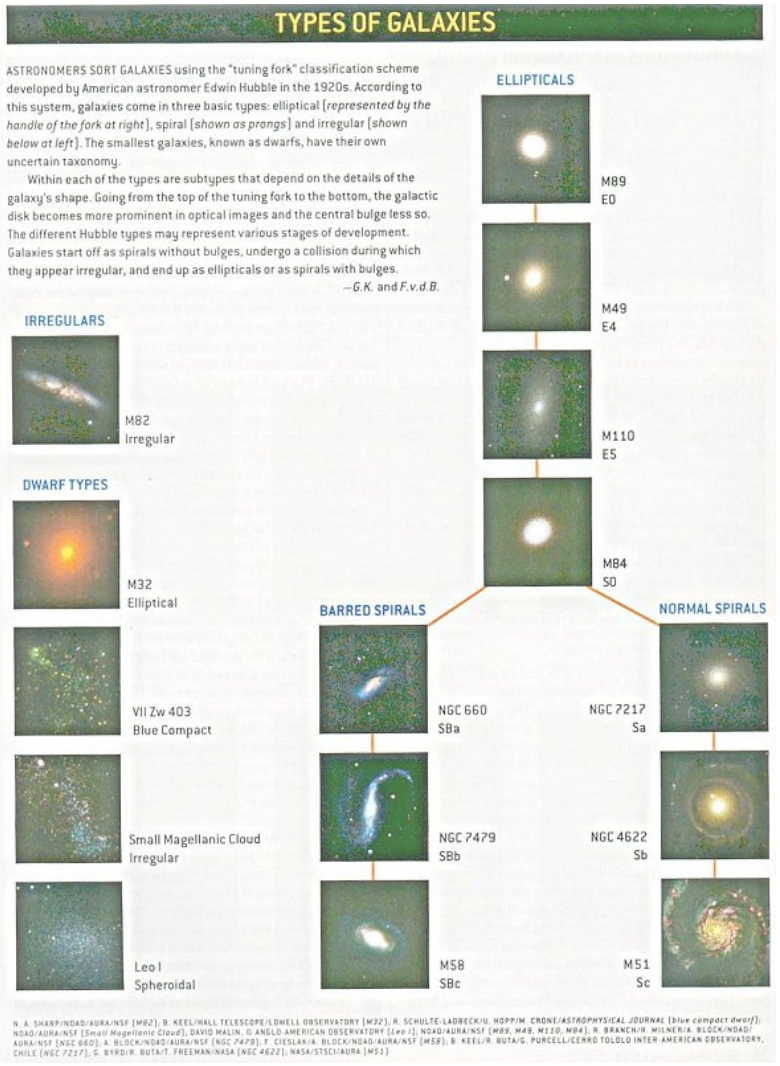
- 銀河形成
- 巨大衝突
- 他にも一杯あるけど、、、

銀河



M101　　M87

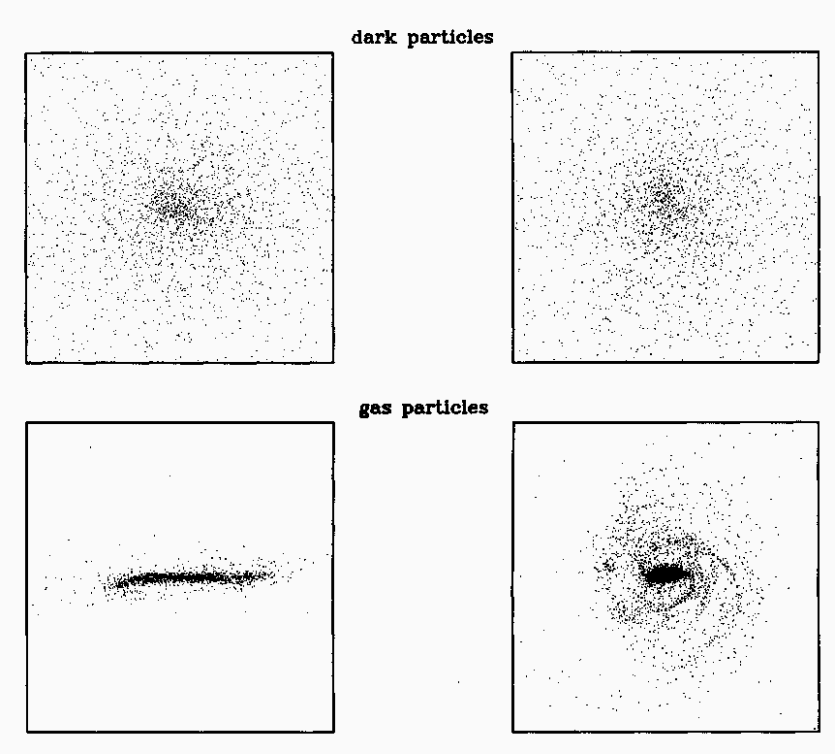
銀河形成シミュレーション



基本的な考え方

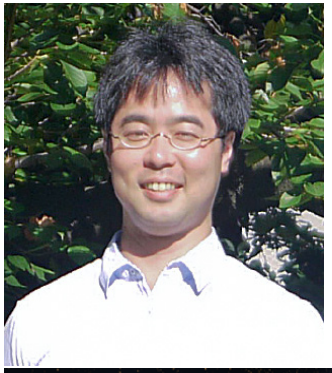
- 宇宙の初期ゆらぎから星形成までを「まるごと」シミュレーション
- 何故銀河には色々あるのか？どうやってできたのか、を明らかにしたい。

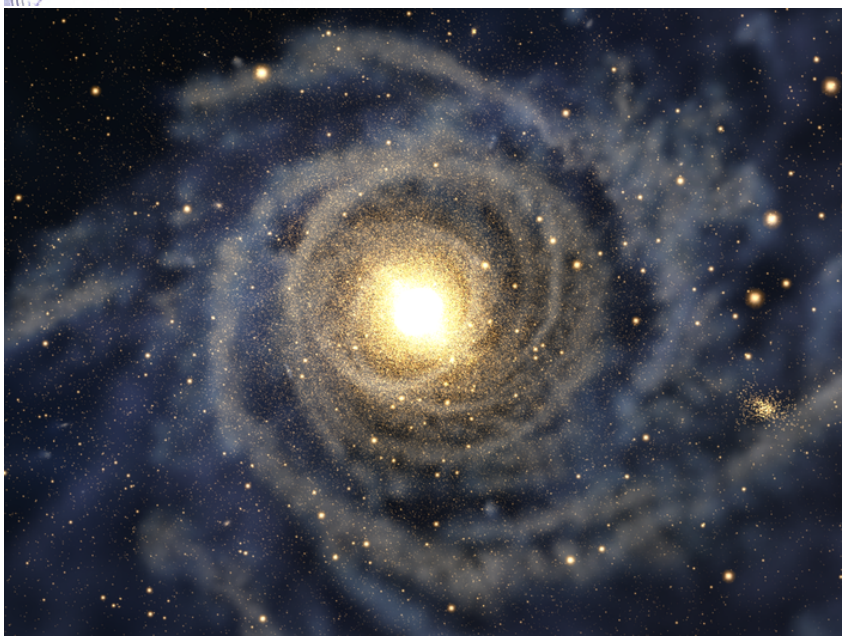
Katz and Gunn 1992



- Dark Matter + ガス + 星
- DM, 星: 粒子、
  ガス:SPH粒子
- 10^4 粒子、Cray YMP 500-1000時間
- 質量分解能: 10^7 太陽質量くらい

Saitoh et al. 2005

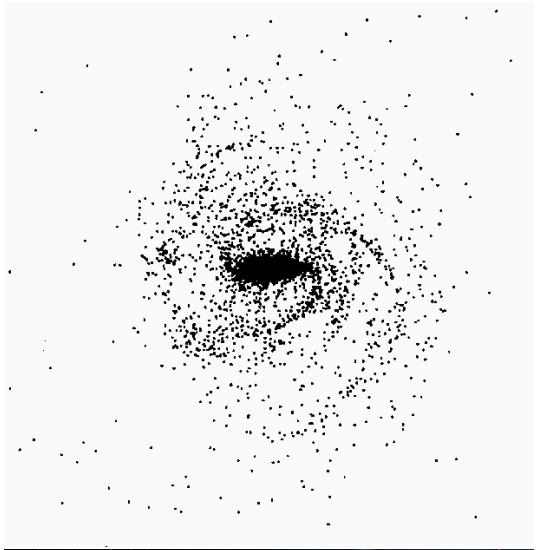


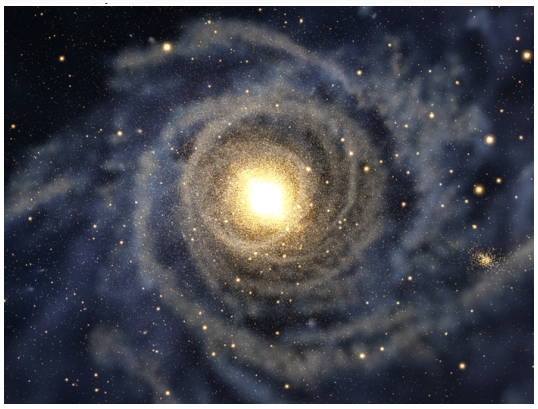


アニメーション

- Dark Matter + ガス + 星
- DM, 星: 粒子、
  ガス:SPH粒子
- 2×10^6 粒子、GRAPE-5
  11 ヶ月
- 質量分解能: 10^4 太陽質量
  くらい

分解能をあげたことの御利益

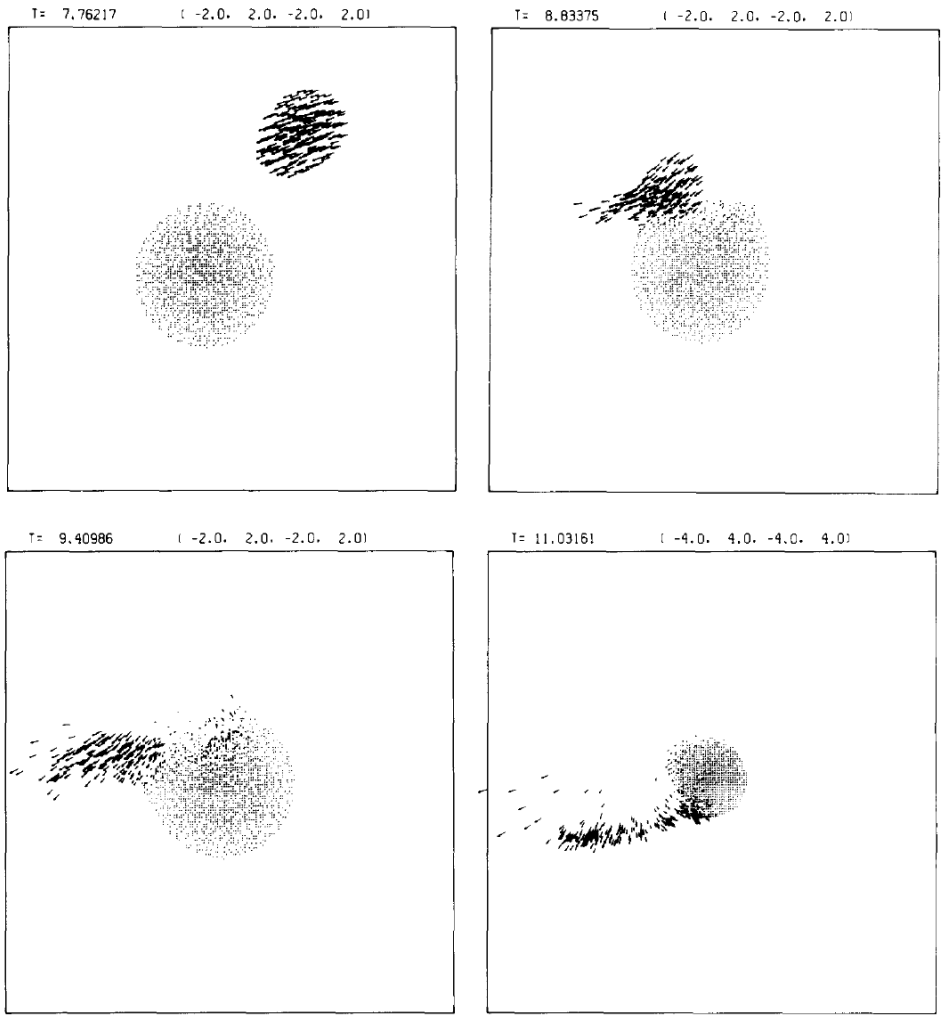




- たいして変わらない？
- 実はこの計算では本当はたいして変わらない。
- 「本当の」利益: 分解能向上 $\rightarrow$ 低温・高密度の星形成領域を解像 $\rightarrow$「モデルに依存しない」星形成

高分解能計算の例 (Baba et al. 2012)

ジャイアントインパクト: Benz et al. 1986

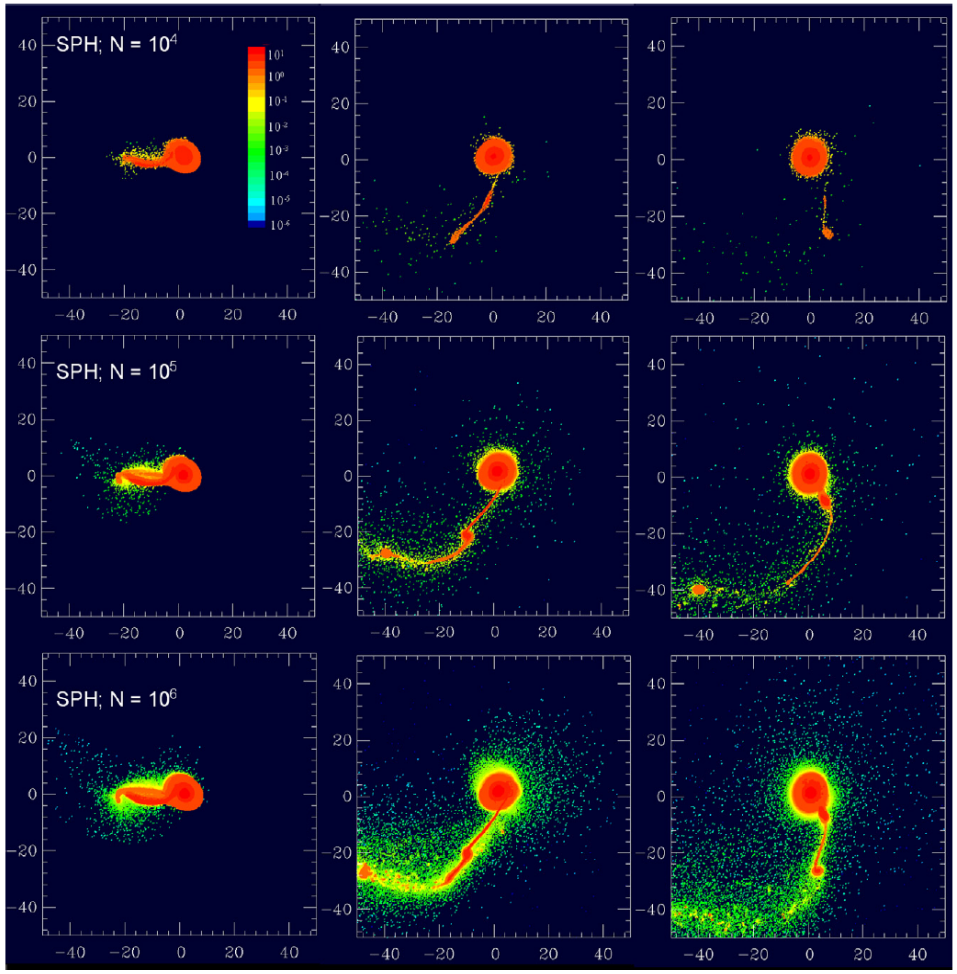


(多分) 世界で初めての巨大衝突シミュレーション

2048 粒子

90 年の Paper IV では 3000 粒子くらい

ジャイアントインパクト: Canup et al. 2013



(多分) 現状で最大
最大 100 万粒子
ほとんどの計算は 30 万粒子

粒子法のメリット・デメリット

メリット

- ラグランジュ描像である: 真空は計算しなくていい、ガリレイ不変である (CFL 条件が楽)、ある程度空間構造に適応的
- 非構造格子を使わないで適応的:(原理的には) 大規模並列化が容易
- コード開発も (AMR に比べれば) 容易

デメリット

- (少なくとも SPH について) 偏微分方程式の真っ当な差分化かどうか怪しい (理論的な問題だけでなく、ケルビン・ヘルムホルツ不安定が抑制されるとか)
- 「原理的には」容易なはずの大規模化、高分解能化ができていない

というわけで、今日はこれらのデメリットをもうちょっとなんとかできないかという話がメイン。

が、その前に

やや怪しげな話を。

- 色々なところで「非圧縮近似」「非弾性近似」といった「非○○近似」が使われる
- これは、CFL条件やフォン・ノイマン条件といった、時間刻みに関する制限を回避するため
- しかし、そうすると、放物型や双曲型の方程式が楕円型に変わるので、ポアソン方程式を解かないといけない。空間多次元だとこれは反復法になって大変
- なので、「非○○近似」を今までとは違う使い方をして、楕円型にしない方法を考える
- 特にマントル対流に使えそう

陰解法と陽解法

- 原理
- 実例 1: 音速抑制法
- 実例 2: 慣性変化法
- その他の可能性

原理

まず、陰解法とは？何故使うのか？という話。
典型的な例: 「非圧縮」流体

- 形式的には、本当の流体に「音速無限大」の近似をしたもの
- 本当の流体の遅い流れをそのまま解くと、CFL 条件によって音波モードでタイムステップが制限される
- 音速無限大の近似をして音波モードを消す代わりに、圧力に対するポアソン方程式がでてくる
- タイムステップは大きくとれるが、ポアソン方程式を解く必要あり

ポアソン方程式の問題点

- 空間 1 次元なら直接解けて計算量も少ない
- 2、3 次元では反復法。分解能あげたり、適応格子にしたりすると段々収束性悪くなる
- CG 反復にしてもマルチグリッド反復にしても、メモリアクセス負荷、通信負荷 (レイテンシに対する要求) が高く大規模並列化が困難

近似の方向を逆に

- 非圧縮近似: 音速無限大の極限
- 逆の近似: 音速を、解の性質が変わらない範囲で「小さく」する
  - CFL 条件が緩和されてタイムステップを大きくとれる
  - 格子マッハ数が 1 よりある程度小さければ大丈夫: 非圧縮近似と同程度のタイムステップがとれる
- 非圧縮近似に比べて悪いところはない
- (何故非圧縮近似がこれまで何十年も使われてきたのかよくわからない)

これは別に新発明ではなくて

(割合最近だけど) 既に例がある。

- Explicit-MPS (山田他 2011, MPS の S って semi-implicit じゃなかったっけ感満載)
- 音速抑制法 (Fan et al. 2003, Hotta et al. 2012)
  - 恒星内部や惑星大気のような、高さが圧力スケールハイトより大きい場合の方法
  - レファレンスの断熱温度分布からの摂動方程式に書き換える
  - それから連続の式だけいじる。

音速抑制法

$$\frac{\partial \rho_1}{\partial t} = -\frac{1}{\xi^2}\nabla\cdot(\rho_0 \boldsymbol{v}),$$

$$\frac{\partial \boldsymbol{v}}{\partial t} = -(\boldsymbol{v}\cdot\nabla)\boldsymbol{v} - \frac{\nabla p_1}{\rho_0} - \frac{\rho_1}{\rho_0} g\boldsymbol{e}_z + \frac{1}{\rho_0}\nabla\cdot\boldsymbol{\Pi},$$

$$\frac{\partial s_1}{\partial t} = -(\boldsymbol{v}\cdot\nabla)(s_0 + s_1) + \frac{1}{\rho_0 T_0}\nabla\cdot(K\rho_0 T_0 \nabla s_1)$$

$$+\frac{\gamma-1}{p_0}(\boldsymbol{\Pi}\cdot\nabla)\cdot\boldsymbol{v},$$

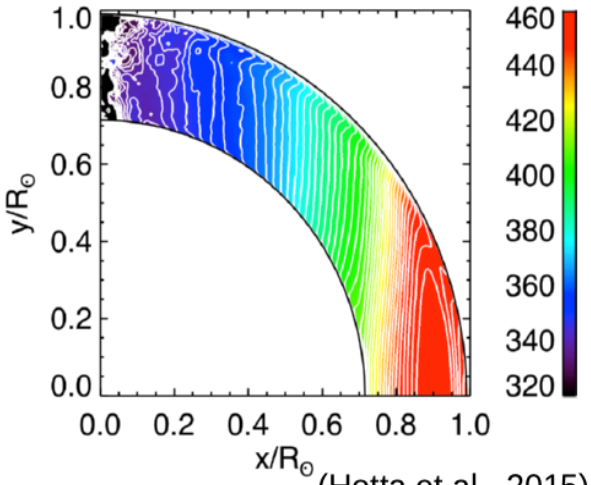
$$p_1 = p_0\left(\gamma\frac{\rho_1}{\rho_0} + s_1\right),$$

添字 0 がレファレンス、1 が摂動。ξ が音速を変える係数
ξ は場所の関数でもかまわない
「京」での大規模計算が初めて可能に

サイエンス成果

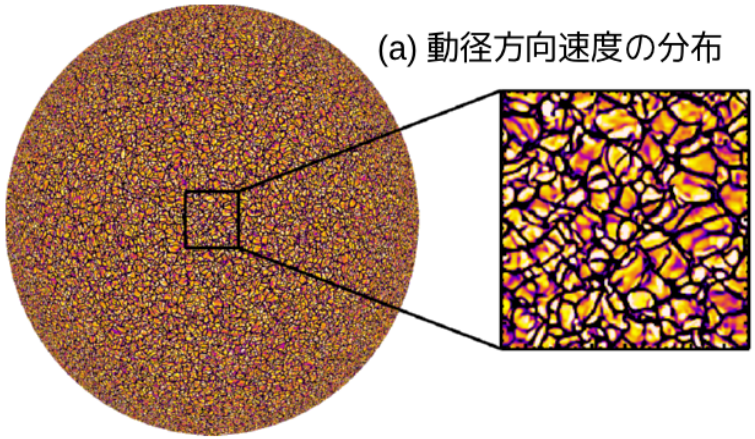
a) 世界最大解像度の太陽対流層全球計算をおこない太陽表面の付近に生成される小スケール熱対流の太陽内部への影響を明らかにした。
b) 太陽表面付近に存在する表面勾配層を世界で初めて数値計算により実現し、その維持物理機構を明らかにした。
c) これまでは熱対流に比べて弱いと考えられていた磁場が、高解像度計算をおこなうことにより、熱対流の運動エネルギーと同等程度まで磁場増幅されていることを明らかにした。

(b) 回転角速度の子午面分布



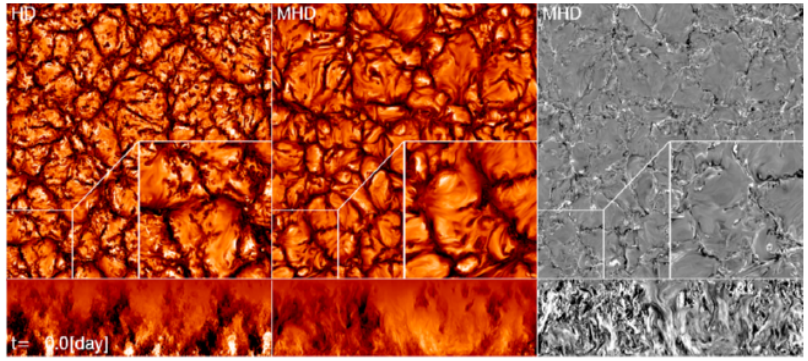
(Hotta et al., 2015)

(a) 動径方向速度の分布



(Hotta et al., 2014)

(c) 鉛直速度と鉛直磁場の分布



(Hotta et al., 2015 in prep)

慣性変化法

目的: マントル対流のような、高粘性流体の計算の陽解法化
過去数十年使われてきたマントル対流の数値計算法:

- **非弾性近似**
- **非圧縮近似 (拡張ブシネスク近似)**
- **運動方程式の慣性項を落とす (小さいので) 近似**

の組み合わせ。普通の音速抑制法では粘性で決まる時間ステップが短いまま

もう一つ近似の方向を逆に

- 普通の慣性項を落とす近似: 慣性項が小さいからできる
- 従って、慣性項を「大きく」してもいいはず
- どこまで大きくしていいか: ストークス流の性質が維持される範囲、すなわち、レイノルズ数が1よりある程度小さいところ

支配方程式とその変形

元々の方程式

$$\rho\frac{dv}{dt} = -\nabla p - \rho g e_z + \nabla\cdot\Pi$$
$$\rho c_p\frac{dT}{dt} = -\lambda\frac{dp}{dt} = \nabla\cdot(k\nabla T) + (others)$$

変形した式

$$\xi\rho\frac{dv}{dt} = -\nabla p - \rho g e_z + \frac{1}{c}\nabla\cdot\Pi$$
$$\rho c_p\frac{dT}{dt} = -\lambda\frac{dp}{dt} = \nabla\cdot(ck\nabla T) + (others)$$

導入したパラメータの意味

- ξ: 慣性項を変化させる。レイノルズ数、マッハ数が小さい限り解に影響しない。位置や状態量の関数でいい
- c: 熱伝導と粘性をコンシステントに変化させる。時間発展の速度を変えるので、グローバルな量

Q: 何故パラメータが1つではなく2つか？

A: マッハ数、レイノルズ数を同時に1に近付けるため。

Q: 粘性が大きく変化する系では破綻しないか？

A: 上手くパラメータとその粘性依存性を調整すると、粘性の比を Rとして、マッハ数、レイノルズ数共に $R^{-1/3}$ より小さくならないようにできる。

こんな方法が本当に上手くいくのか？

- やってみたら本当に上手くいく (Takeyama, et al., submitted to JCP)
- 粘性が変化する場合とかも大丈夫みたい
- マントル対流の超高分解能計算への可能性が開けたかも

これのどこが SPH の話かというと

- 今回 SPH で実装した。
- 対流を扱うのに、非圧縮とか拡張ブシネスクとかの近似は一切必要ない。元々の方程式のまま (で、慣性項を変えるのと拡散係数のスケーリングのみ行う)。
- (まだやってみてないけど)3 次元対流とかでもデカルト座標でそのままやれる。

計算例

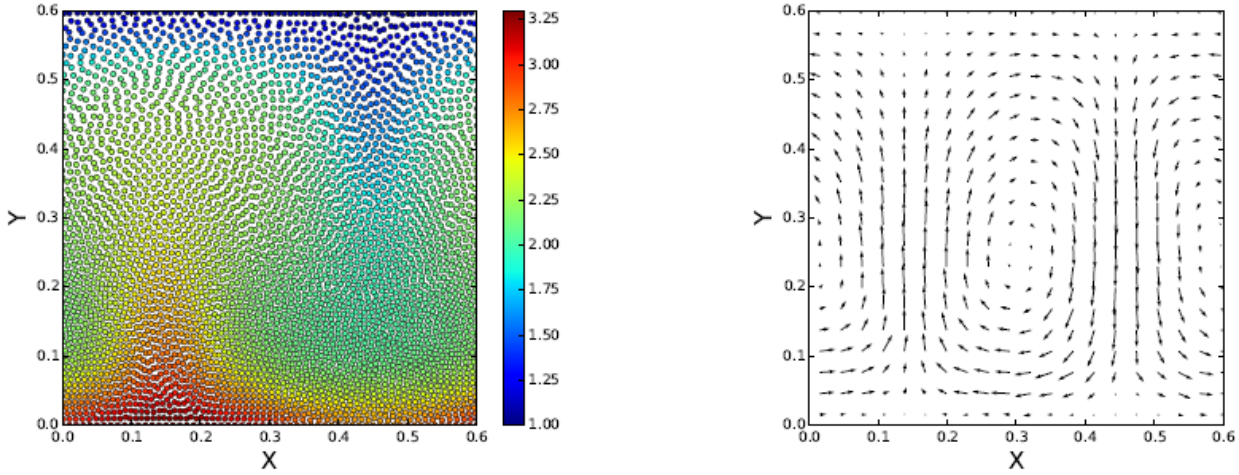
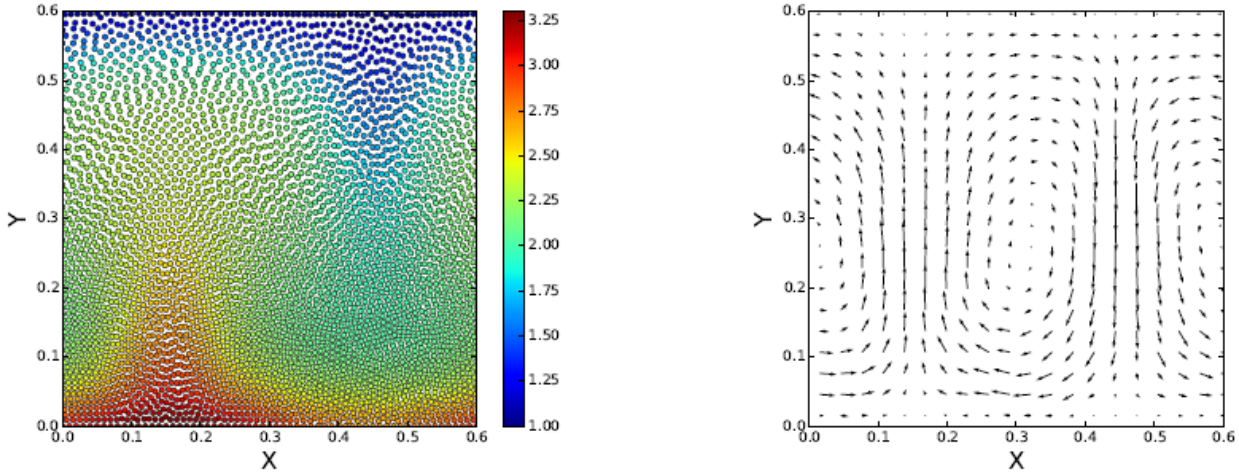
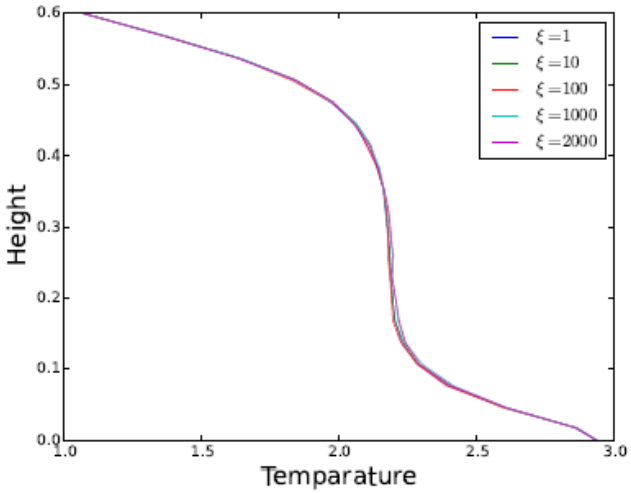


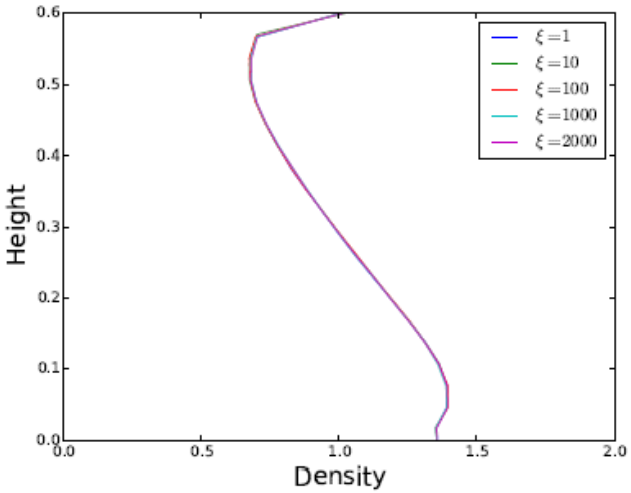
Figure 5: The temperature distribution and the velocity distribution when ξ=1.



下は慣性項を 1000 倍にしたもの。みため同じ。

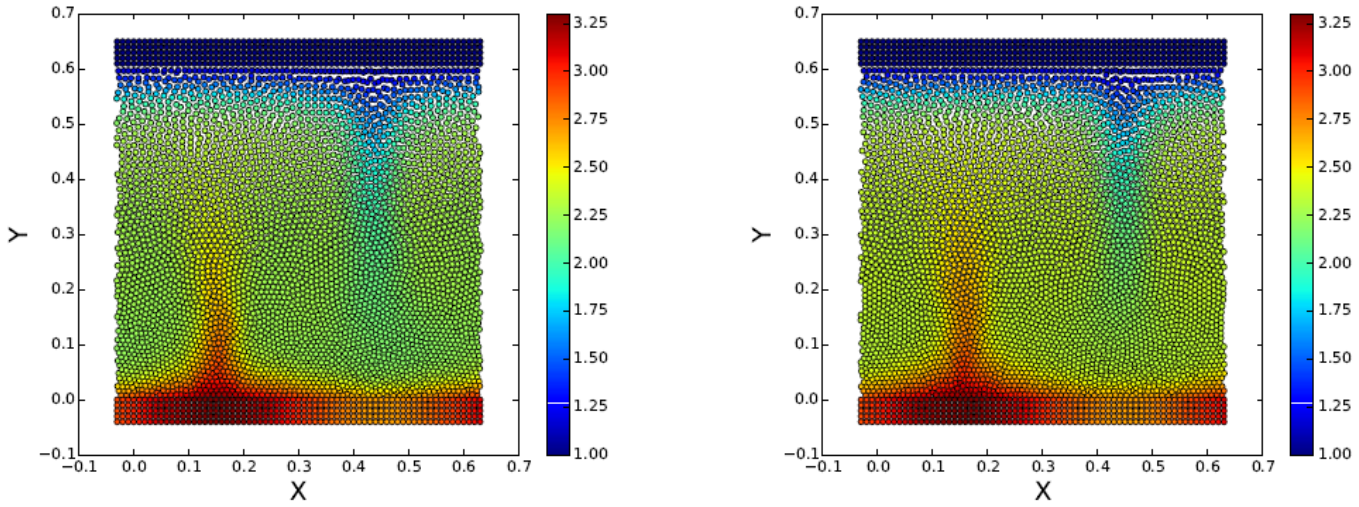
計算例続き





- 温度分布・密度分布ももちろん同じ。他の物理量、熱流等も同じ。
- 計算時間は慣性項を 1000 倍にすると $1/1000$ になる。ほぼダイナミカルな計算になる。レイノルズ数で 1 くらい。

温度依存する粘性



- 上と下で粘性２桁違う
- 慣性項を温度依存で変化させた。特に問題なく計算できた。

他の応用

時間発展のどこかに「速度無限大」をいれて陰解法にしている話なら原理的になんでも応用可能なはず

- **輻射伝達 (光速を遅くする)**
- **化学反応を伴う流れ**
- **電磁場、重力場 (??)**

まとめ(陰解法関係の話だけ)

- 亜音速流体に対しては、音速を遅くした上で圧縮性流体向きの解法を使うのが好ましい
  - 反復が不要、メモリバンド幅要求が下がる、通信レイテンシ要求も下がる
- 単純に状態方程式をいじることができないケースでも適用可能(音速抑制法、慣性変化法)
- 陰解法が必須な場面というのはそれほどないのではないか？

疑似密度法

話の順番

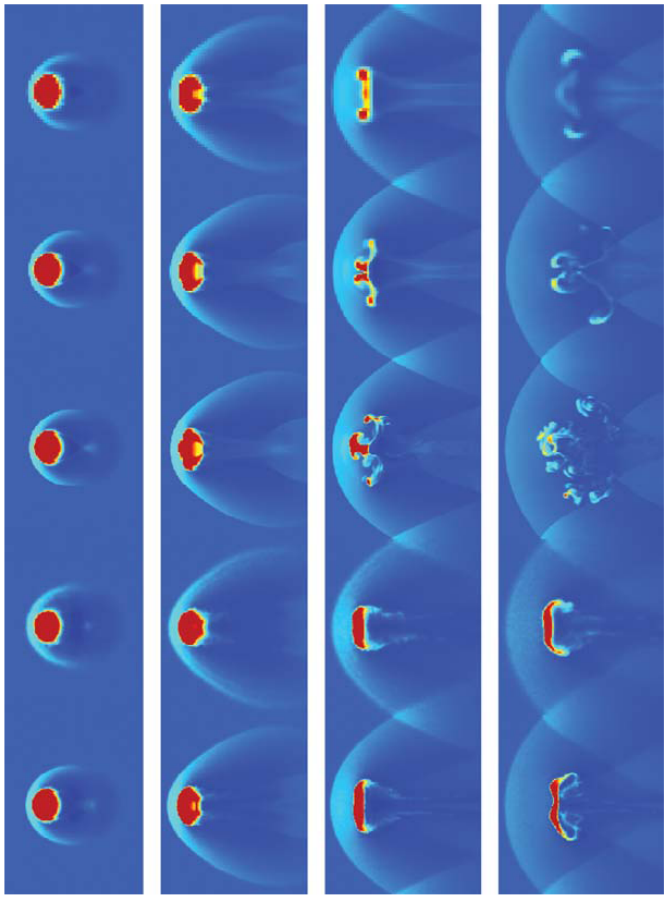
- 接触不連続
- 対策
- まとめ

SPH と接触不連続、KH 不安定

Agertz et al (MN 2007, 380,963)

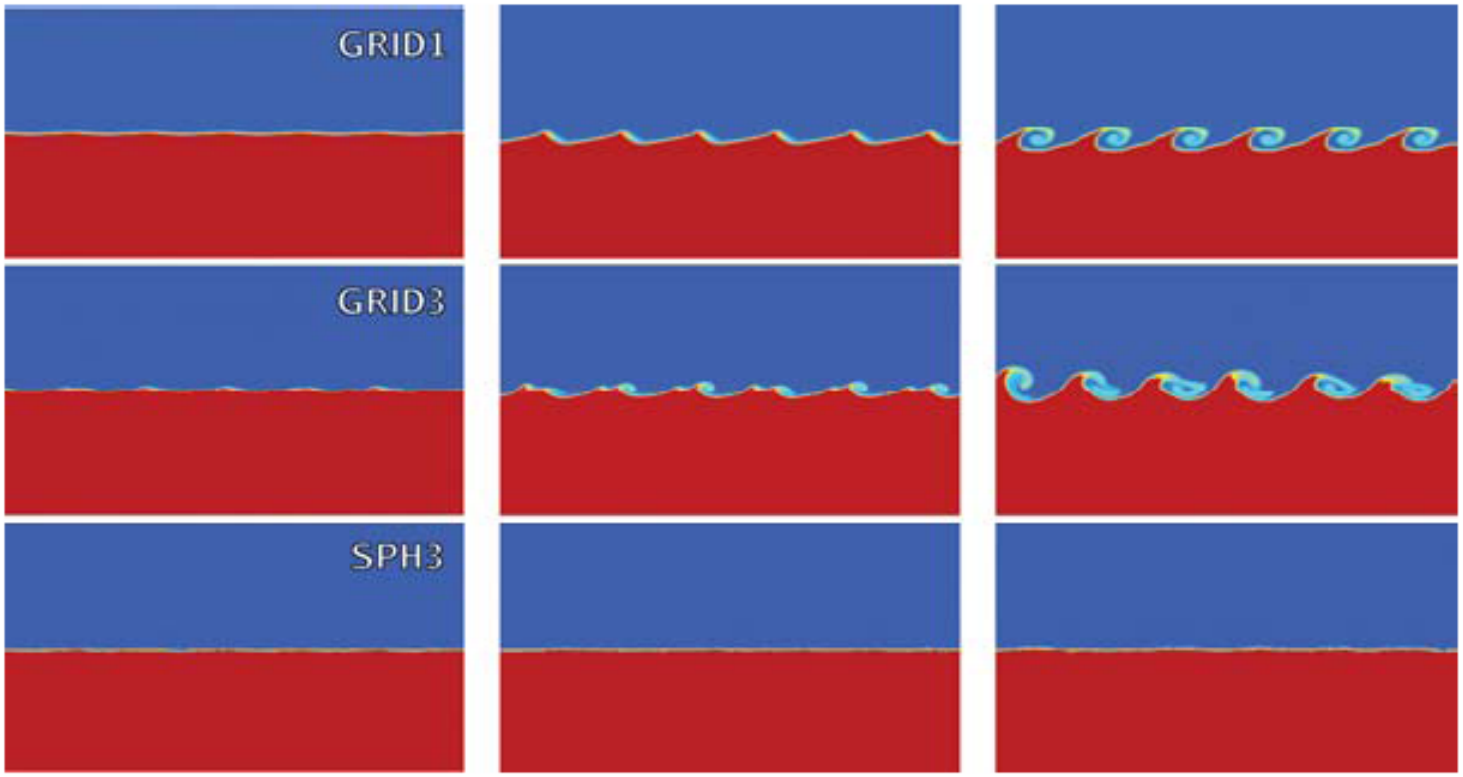
- SPH と Grid コードで、「Brob test」の答が全然違う
- もっと簡単な Kelvin-Helmholtz 不安定 (計算は 3 次元)
  でも全然違う
- SPH だめじゃん

どれくらい違うか (1)



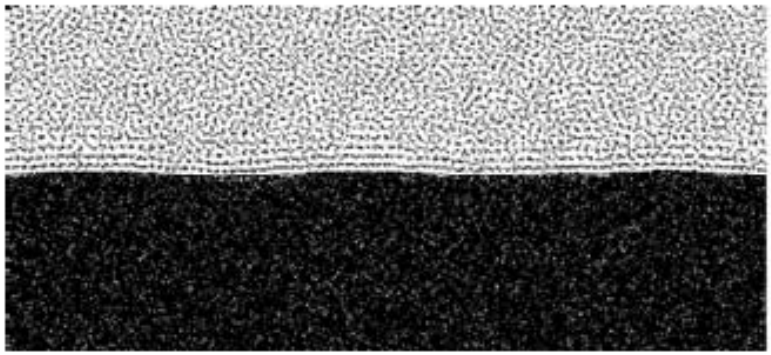
- 周りよりつめたい (温度 1/10、、密度 10 倍) ガスの球を超音速で動かす
- 上から 3 個は Grid
- 下の 2 つは Gasoline (下は 10M 粒子)
- SPH では 境界での不安定が起きないで、冷たい流体が固まりのまま。

どれくらい違うか (2)



SPH では KH 不安定が起きない。

どれくらい違うか (3)



2 流体の境界面で妙な隙間ができる。このため力が働かない？

密度不連続面での振る舞い

通常の SPH では、変形の 2 箇所で ρ の微分可能性を仮定している。以下の 2 つの「恒等式」である

$$1 = \sum_j m_j \frac{1}{\rho(\vec{x})} W(\vec{x} - \vec{x}_j). \tag{1}$$

$$\frac{1}{\rho} \nabla P = \frac{P}{\rho^2} \nabla \rho + \nabla \frac{P}{\rho^2}. \tag{2}$$

SPH でカーネル推定した密度は滑らか。このため、

- 接触不連続の低密度側では、密度を過大推定、高密度側では過小推定する
- その結果、圧力、その空間微分もデタラメになる。結果として粒子が再配置される

SPH の基本式

ある物理量 f の推定

$$\langle f \rangle(\vec{x}) = \int f(\vec{x}')W(\vec{x} - \vec{x}')d\vec{x}'. \tag{3}$$

密度の推定

$$\rho(\vec{x}) = \sum_j m_j W(\vec{x} - \vec{x}_j), \tag{4}$$

SPH 近似

$$\langle f \rangle = \sum_j m_j \frac{f_j}{\rho(\vec{x})} W(\vec{x} - \vec{x}_j). \tag{5}$$

SPH の基本のつづき (1)

f の微分: $\langle \nabla f \rangle = \nabla \langle f \rangle$ で、以下の恒等式

$$1 = \sum_j m_j \frac{1}{\rho(\vec{x})} W(\vec{x} - \vec{x}_j). \tag{6}$$

を使って、さらにもうちょっと近似して

$$\langle \nabla f \rangle(\vec{x}) \sim \sum_j m_j \frac{f(\vec{x}_j)}{\rho(\vec{x}_j)} \nabla W(\vec{x} - \vec{x}_j). \tag{7}$$

SPH の基本のつづき (2)

運動方程式は $-\frac{1}{\rho}\nabla P$ を計算する。この時に恒等式

$$\frac{1}{\rho}\nabla P = \frac{P}{\rho^2}\nabla\rho + \nabla\frac{P}{\rho^2}. \tag{8}$$

を使って対称化すると

$$\dot{v}_i = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \frac{\partial}{\partial x_i} W(x_i - x_j), \tag{9}$$

になる。

対策

「根本的」な理由:

> **ρ は滑らかだけど u (内部エネルギー) はジャンプがあるまま。**

この観点では、u を滑らかにすればよい。色々提案あり。

- **u にもカーネル推定した量を使う**
- **u を拡散させる (人工熱伝導)**
- **質量密度でない密度 (数密度とか) を使う**

それぞれ、それなりにうまくいくケースもある。

新しい提案 — 思想

圧力が本来変わってないのに、密度が不連続なだけでおかしなことが起こるのは何故か？

物理量 (とその微分) の推定式に密度を使うから:

$$\langle f \rangle(\vec{x}) = \sum_j \frac{m_j f(\vec{x}_j)}{\rho(\vec{x}_j)} W(\vec{x} - \vec{x}_j). \tag{10}$$

ここでやっていることは、本質的には体積要素 $d\vec{x}$ を $m_j/\rho(\vec{x}_j)$ で置き換えているだけ。

粒子の占める体積の推定さえできれば別に何を使ってもいいはず

新しい提案 — 実際

これまでいくつかやってみた

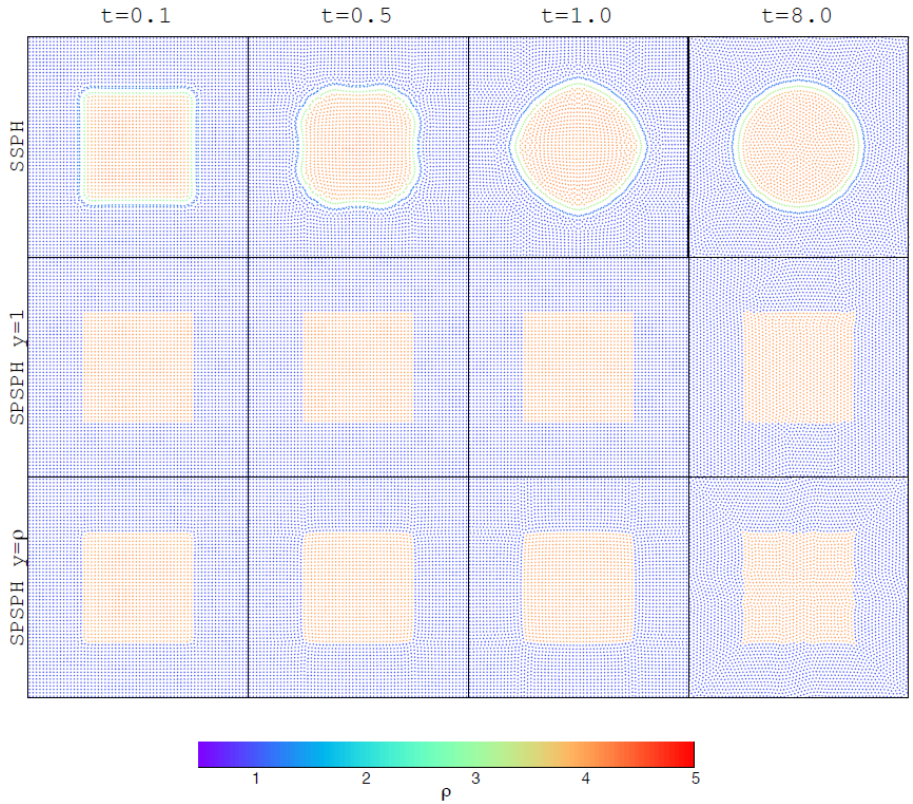
- 密度の代わりに圧力を使う (Saito and Makino 2013, Hosono et al. 2013)
- 圧力の弱いべき乗を使って、圧力変化が大きいところの振舞いを改善する (SM13, Hosono et al. 2014)

これはこれで上手くいくが、圧力のスケールハイトが粒子間隔くらい以下になるとやはり上手くない。ので別の方法を考えた。(Yamamoto et al. 2015)

- 体積要素を計算するためだけに「疑似質量」を導入
- 「疑似密度」は適当な拡散係数で拡散させることで不連続面を消す。これは粒子のエネルギー、圧力を (誤差の範囲でしか) 変えない。

定式化の細かい話は今日は省略。

数値例(1)



静水圧平衡の密度だけ違う流体。なにも起きないのが正しい。

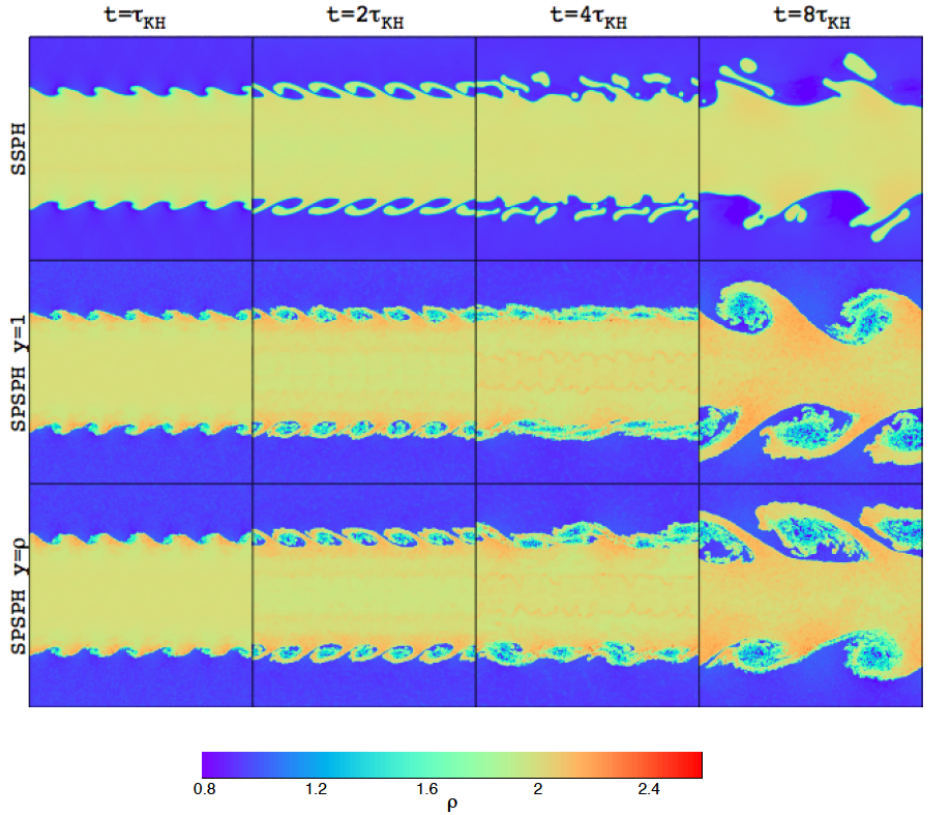
上:従来の。

中央:初期疑似密度$=1$

下:初期疑似密度$=$本当の密度

新しい方向では初期設定によらずちゃんと計算できた。

数値例(2)



KHI。ちゃんと渦まいて欲しい

上:従来の。

中央:初期疑似密度=1

下:初期疑似密度=
本当の密度

新しい方向では初期設定によらずちゃんと計算できた。

まとめ (疑似密度法)

- SPH で、接触不連続でおかしなことが起きないようにする方法を定式化した
- 水と空気のような、密度比が極端に大きい場合でも 0 次では問題ない
- 圧力勾配の不連続は表現できないので、運動方程式に高次の誤差項は残るはず。
- とはいえ、今までのよりはるかにまし。
- 自由表面、接触不連続をもうちょっとまともに扱う方法を検討中。

「粒子法の大規模並列化フレームワーク」の話

(何故大規模化ができてないか？その対応は？という話)

1. どんなことをしたい (あるいはしたくない) か
2. ではどうするか
3. 実装方針
4. 現状と将来

どんなことをしたいか

というよりむしろ、何がしたくないか

- MPI でプログラムなんか書きたくない
- キャッシュ再利用のためのわけのわからないループ分割とかしたくない
- 通信量減らすためにわけのわからない最適化とかするのも勘弁して欲しい
- SIMD 命令がでるようにコードをいじりまわすとかやめたい
- 機械毎にどういう最適化すればいいか全然違うとか、それ以前に言語から違うとかはもういやだ

そうはいっても――ではどうするか？

昔からある考え方はこんな感じ？

- 並列化コンパイラになんとかしてもらう
- 共有メモリハードウェアになんとかしてもらう
- 並列言語とコンパイラの組み合わせになんとかしてもらう

しかし……

- 若い人はそういう考え方があったことも既に知らないような気がする。「スパコンとはそういうものだ」みたいな。
- つまり、こういうアプローチはほぼ死滅した。
- 理由は簡単: 性能がでない。安価なハードウェアで高い性能がでるものがプログラミングが大変でも長期的には生き残る。

じゃあ本当のところどうするか？

1. 人生そういうものだと諦めて MPI でプログラム書いて最適化もする
   難点: 普通の人の場合性能がでない。難しいことをしようとすると無限に時間がかかって人生が終わってしまう。
2. 他人 (学生、ポスドク、外注、ベンダ等) に MPI でプログラム書かせて最適化もさせる
   難点: 他人が普通の人の場合、やはり性能でない。無限に人と時間とお金がかかる。あとでいじるにも無限に人と時間とお金がかかる。

- どちらも今一つというか今百くらいである。
- もちろん、「普通でない人」を確保できればなんとかなるがこれは希少資源である。

原理的には、「普通でない人」を有効利用すればいい？

どうやって有効利用するか？

色々な考え方がありえるが、我々の (というか私の) 考え方:

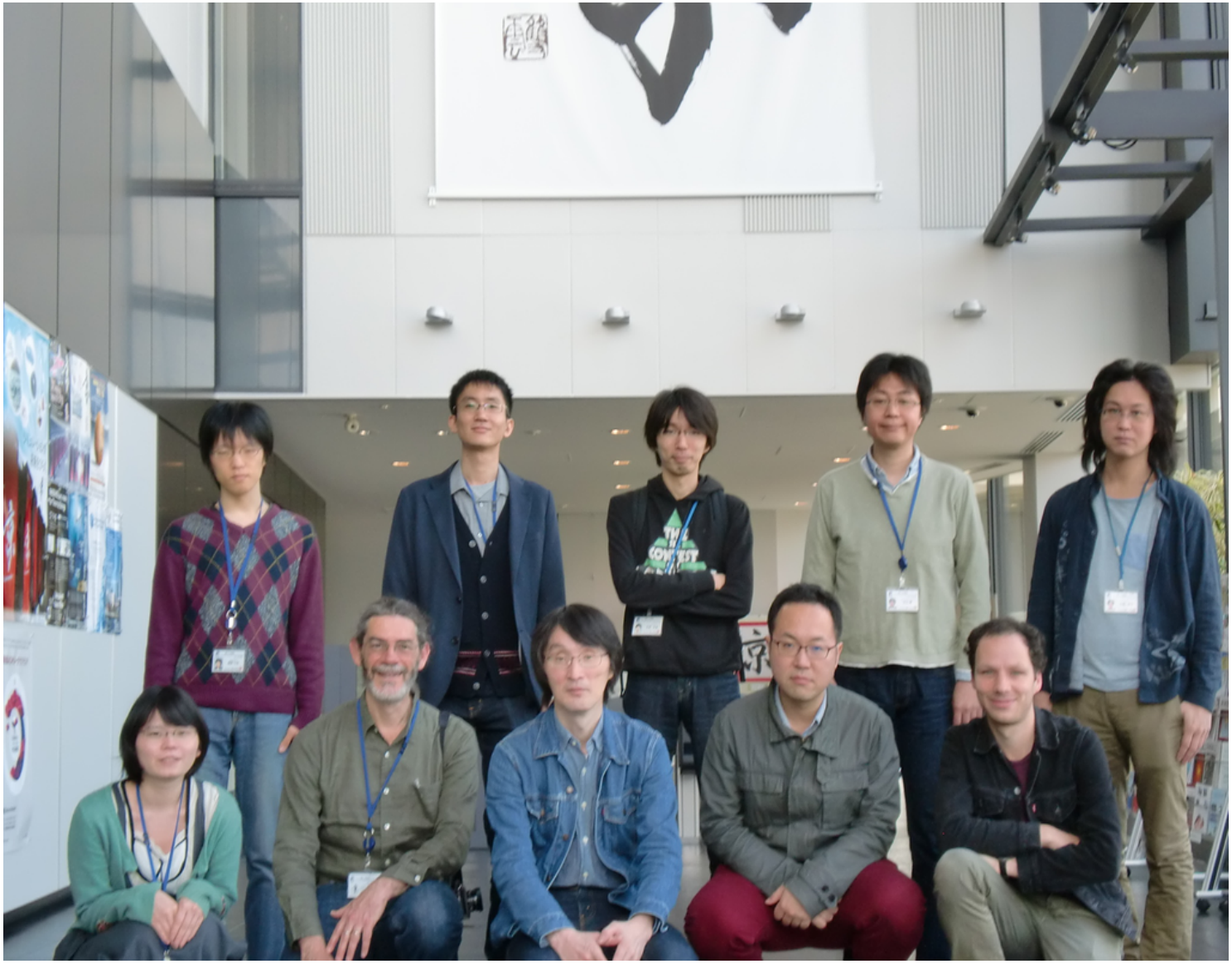
- 「普通でない人」がやった方法を一般化して、色々な問題に適用する。
- 例えば「粒子系一般」という程度
- DRY (Don't Repeat Yourself) の原則の徹底

どうやって有効利用するか？

色々な考え方がありえるが、我々の (というか私の) 考え方:

- 「普通でない人」がやった方法を一般化して、色々な問題に適用する。
- 例えば「粒子系一般」という程度
- DRY (Don't Repeat Yourself) の原則の徹底
- 「神戸人外王国」が詳細仕様策定及び実装

(神戸花鳥園は 2014 年 7 月から神戸どうぶつ王国に生まれ変わりました)



＋坪内 (テクニカルスタッフ)、若松 (アシスタント)

もうちょっと具体的には？

色々な粒子系計算

- 重力多体系
- 分子動力学
- 粒子法による流体 (SPH、MPS、MLS、その他)
- 構造解析等のメッシュフリー法

計算のほとんどは近傍粒子との相互作用 (遠距離力: Tree, FMM, PME その他)

なので、粒子分布を与えると

- 領域分割 (ロードバランスも考慮した)
- 粒子の移動
- 相互作用の計算 (そのために必要な通信も)

を高い効率 (実行効率・並列化効率) でやってくれるプログラムを「自動生成」できればいい

実装方針

- API は C++ で定義
- ユーザーは
  - 粒子データクラス
  - 粒子間相互作用を計算する関数 (現在のところ、ユーザーが機種毎に最適化。この自動生成は並行して開発中)

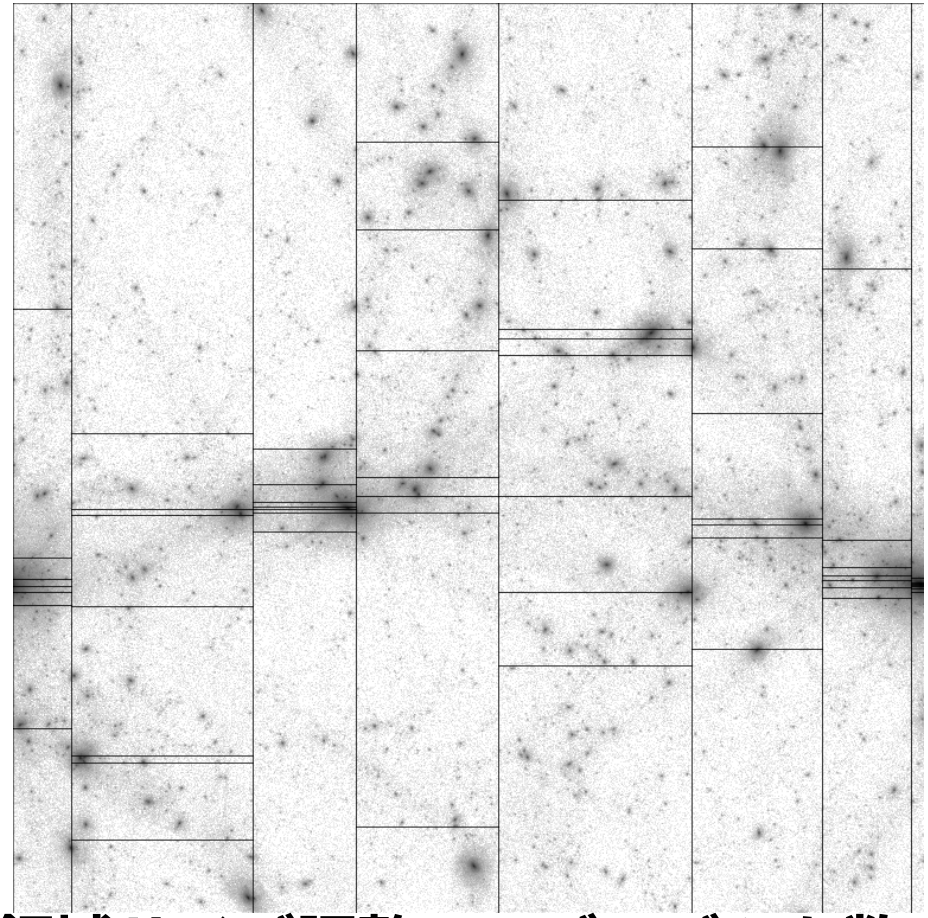
  を用意。さらにドライバープログラム (I/O ライブラリコール等も含む)、時間積分関数とかも書く
- 全体をコンパイルすると空間分割して粒子再配置して時間積分して、、、というプログラムになる

開発の現状

- 公開した (https://github.com/FDPS/FDPS/)
- 同一のユーザープログラムで、シングルスレッドでもマルチスレッドでもMPI並列でも動く。
- 動くだけでなく、並列化効率は「非常に良い」
- 重力多体(Barnes-Hut ツリー)、重力+SPH が「京」全ノードくらい(計算時間が、、、)までで動作、重力(遠距離相互作用)は数万ノードでも実行効率高い。SPH(近距離相互作用)は改良中
- 多数のユーザーから喜びのお手紙を(ちょっと嘘)
- GPU 対応等実装中

チュートリアル

空間分割と並列化



空間分割して計算ノードに割り当て

Recursive Multisection (JM 2004)

「京」の Tofu ネットワークに適した方法

計算時間が均等になるよう領域サイズ調整 (石山他 2009、2012)

領域サイズ調整アルゴリズムを数万ノードでもボトルネックにならないよう改良 (岩澤他 2015)

重力多体計算のサンプルユーザーコード

1. ヘッダと粒子クラス

```
#include <particle_simulator.hpp> //必須
using namespace PS;  //コードを簡潔に、、、、

class Nbody{                //名前は自由
public:
    F64    mass, eps;   //名前は自由
    F64vec pos, vel, acc; //名前は自由
    F64vec getPos() const {return pos;} //必須
    F64 getCharge() const {return mass;}//必須
    void copyFromFP(const Nbody &in){   //必須
        mass = in.mass;
        pos  = in.pos;
        eps  = in.eps;
    }
    void copyFromForce(const Nbody &out) { //必須
        acc = out.acc;
    }
```

粒子クラス(2)

```
    void clear() { //必須
        acc = 0.0;
    }
    void readAscii(FILE *fp) {//FDPSのI/O使うなら
        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) { //ユーザー利用
        vel += (0.5 * dt) * acc;
        pos += dt * vel;
    }
    void correct(F64 dt) { //ユーザー利用
        vel += (0.5 * dt) * acc;
    }
};
```

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;
```

相互作用関数(続き)

```
            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;
        }
    }
};
```

時間積分(ユーザープログラム内)

```
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);
}
```

ユーザーの相互作用計算関数

```
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);
}
```

ここで FDPS の関数群を使っている。

ユーザープログラム本体

```
int main(int argc, char *argv[]) {
    F32 time  = 0.0;
    const F32 tend  = 10.0;
    const F32 dtime = 1.0 / 128.0;
    // FDPS 初期化
    PS::Initialize(argc, argv);
    PS::DomainInfo dinfo;
    dinfo.initialize();
    PS::ParticleSystem<Nbody> ptcl;
    ptcl.initialize();
    // FDPS に相互作用関数を渡す
    PS::TreeForForceLong<Nbody, Nbody,
        Nbody>::Monopole grav;
    grav.initialize(0);
    // 粒子データ読み込み
    ptcl.readParticleAscii(argv[1]);
```

ユーザープログラム本体

```
    // 相互作用計算
    calcGravAllAndWriteBack(dinfo,
                            ptcl,
                            grav);
    while(time < tend) {
        predict(ptcl, dtime);
        calcGravAllAndWriteBack(dinfo,
                                ptcl,
                                grav);
        correct(ptcl, dtime);
        time += dtime;
    }
    PS::Finalize();
    return 0;
}
```

コメント等

- 粒子クラスに謎関数があるのは、本来の粒子クラスと相互作用計算に最適化した粒子クラスを別に定義できるようにするため。
- 粒子種も複数定義可能 (ダークマターと流体等)。このために相互作用を定義するところが若干複雑
- ユーザー定義の相互作用計算は今のところアーキテクチャ向け最適化が性能出すためには必須。
- 高度に最適化された並列化ツリーアルゴリズムを 150 行くらいのユーザープログラムで使える。

まとめ

- 粒子法は天体物理・惑星科学で色々使われている
- が、問題点も結構沢山ある。
  - 接触不連続で適合的でない
  - 並列化が結構困難
- 前者は「改善」するスキームを開発したがまだ問題は残る
- 後者は「汎用フレームワーク」で解決した $(?)$
- 高粘性流体にも適用できる計算スキームを開発した。これからもうちょっと実験。