惑星形成シミュレーションの現状と将来

牧野淳一郎
神戸大学 理学研究科 惑星学専攻

2017/9//6 STE シミュレーション研究会

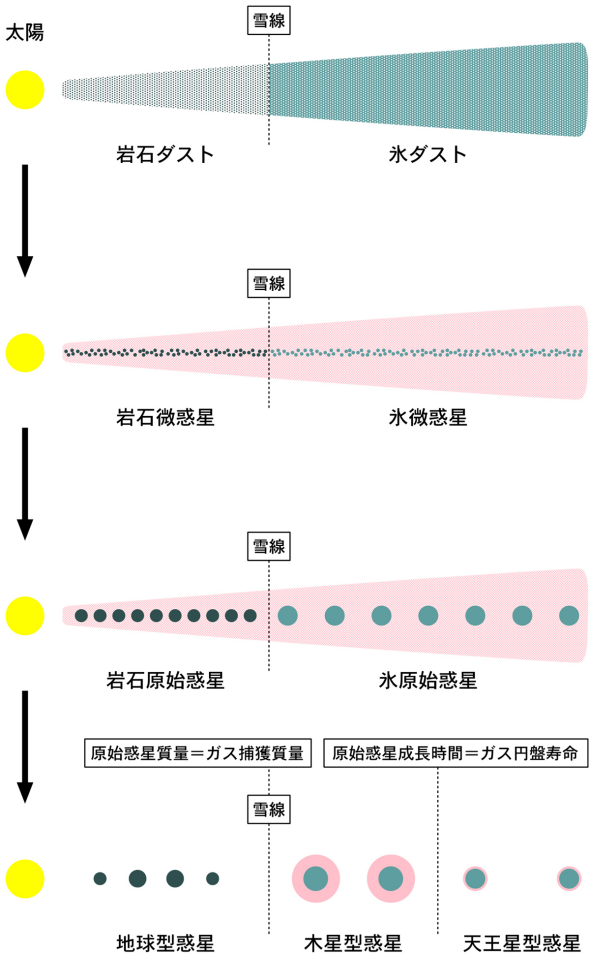
話の概要

- 惑星形成の多体シミュレーションを中心に、これまでのシミュレーション手法の発展、サイエンスの発展を概観
- 数値計算アルゴリズムの話題も

話の構成

- 現状の惑星形成シナリオの概観
- シミュレーション研究の歴史
- 手法の概観

現状の惑星形成シナリオ



(理科年表から。小久保による)

- 太陽の周りに原始惑星系円盤。水素、ヘリウム＋それ以外。
- 太陽に近いところでは水は気体。外側は氷: 惑星材料の量が違う
- ダストは赤道面に沈降、集まって「微惑星」になる。($10^{15}kg$ くらい)
- 微惑星同士がさらに重力相互作用で衝突・合体して「原始惑星」に (10 万年くらい？ $10^{23}kg$ くらい)
- 原始惑星がさらに合体して地球型、あるいはガスを集めて木星型に

30年くらい前の状況

Hayashi, et al. 1985

- 微惑星から惑星へ、という基本的な描像は既にあった
- しかし、理論的には惑星ができるのに時間がかかりすぎる、という問題があった

何故時間がかかるということになっていたか？

- 惑星が成長すると成長速度が遅くなる (質量の1/3乗)
- 太陽から遠いと成長速度が遅くなる (距離の3乗)

海王星は存在しないはず (形成時間100億年以上)

形成時間問題への解

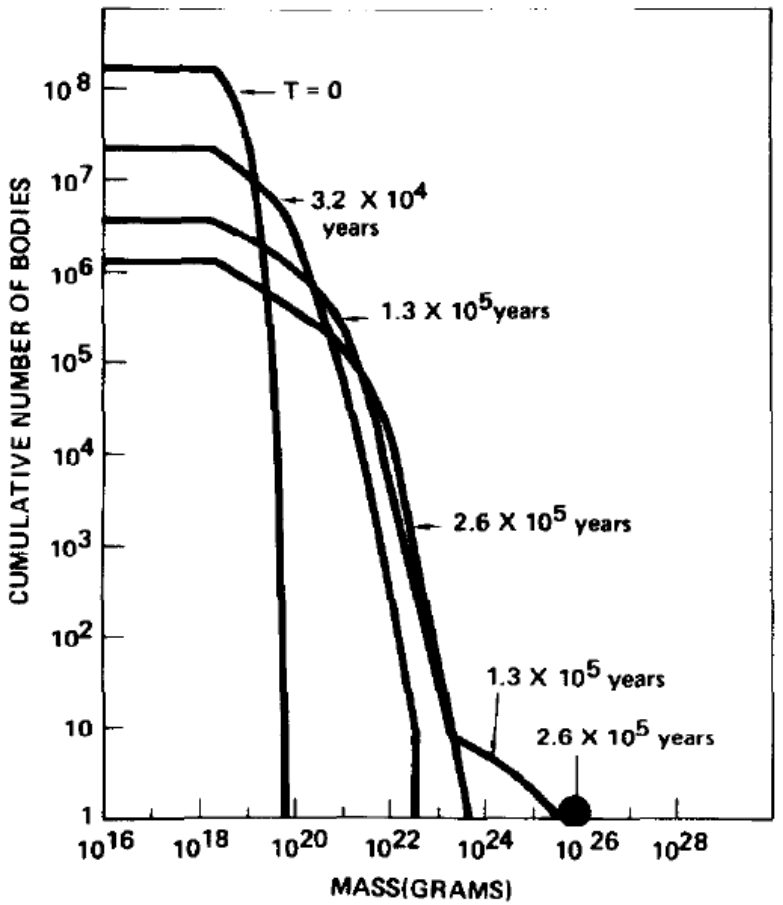
暴走的成長 (**Wetherill and Stewart 1989**)

- それまでの理解:秩序的成長。微惑星は同じように重くなる
- 暴走的成長: 周りよりも少し重くなったものが他より速く成長してどんどん大きくなる

速く成長する理由

- 大きいので衝突しやすい
- 重いので、微惑星同士の重力の効果も大きい
- 円軌道に近いので、重力の効果がさらに大きい

Wetherill and Stewart 1989

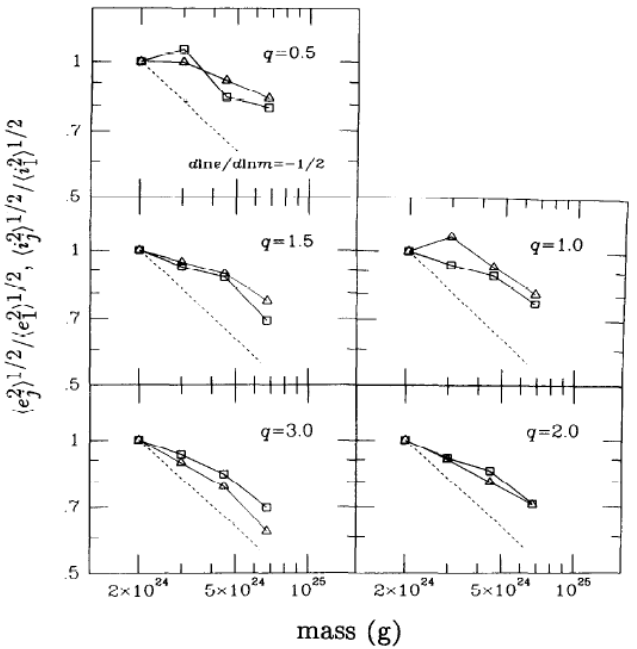


- 微惑星の質量分布の時間変化を「モンテカルロ」計算
- 衝突・合体の効果、速度分散等はモデル
- 水平方向空間分布は「一様」
- 最初深いべき（-2.5 乗くらい) の質量分布ができて、そこからさらに重いものができる

N体シミュレーション研究

- そういうわけで井田さんがN体計算始めたのが 90 年頃
- それまでにも研究あり。**Lecar and Aarseth (1986):200**体、月くらいから始める。
- この結果を今見直すと、暴走成長とか色々起きてるようにも、、、

Ida and Makino 1992a,b, 1993



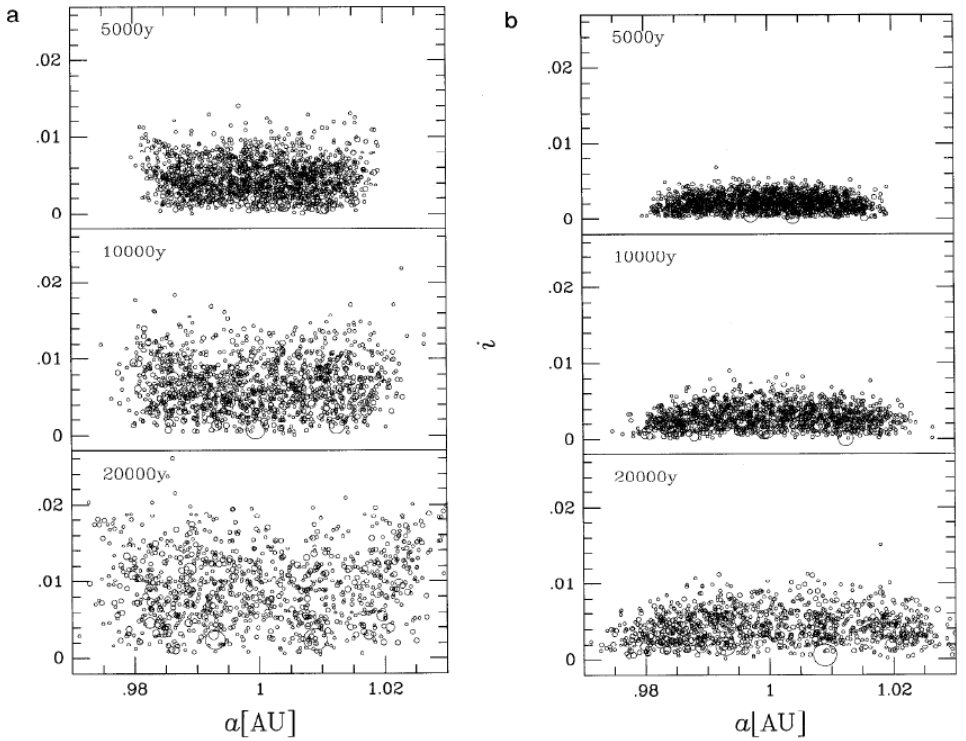
- **(1992a)** 単一質量での速度分散の時間進化を計算機シミュレーション
- **(1992b)** 複数質量での速度分散の質量依存性を計算機シミュレーション
- 重いものが速度分散小さくなることを確認

暴走的成長には限界があることを指摘。ある程度重くなると、自分自身が周りの微惑星の速度分散を大きくするので成長できなくなる
(=原始惑星)

(実際の合体・成長過程をシミュレーションで調べてはいない)

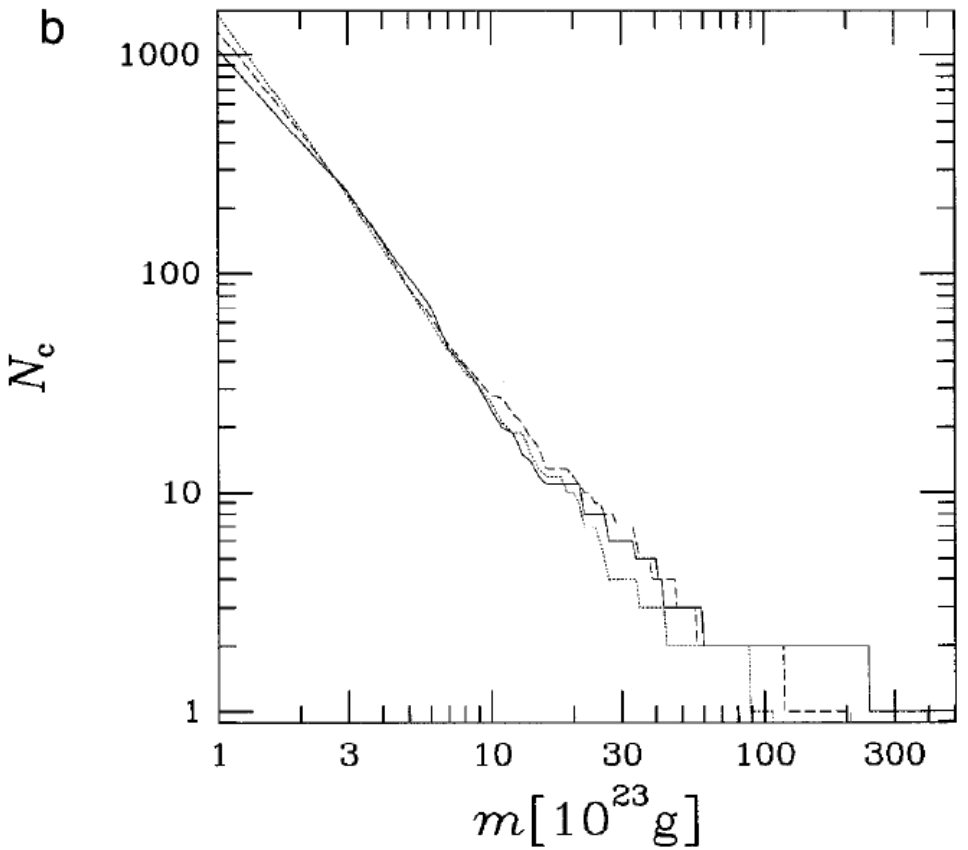
Kokubo and Ida 1996

グラフの横軸は太陽からの距離、縦軸は「離心率」、〇の大きさは微惑星の質量



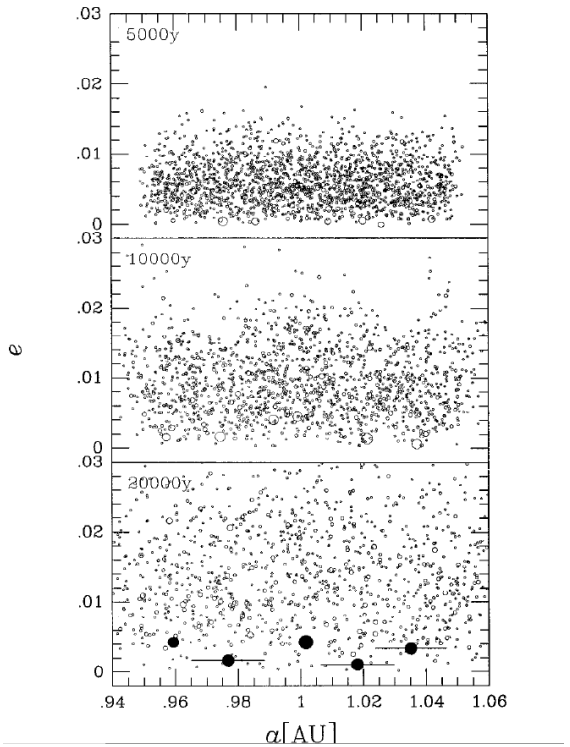
- 細いリング状領域のシミュレーション、衝突・合体も扱う
- 衝突の時間スケールは惑星大きくして短く
- 暴的成長が起きることを確認

累積質量分布



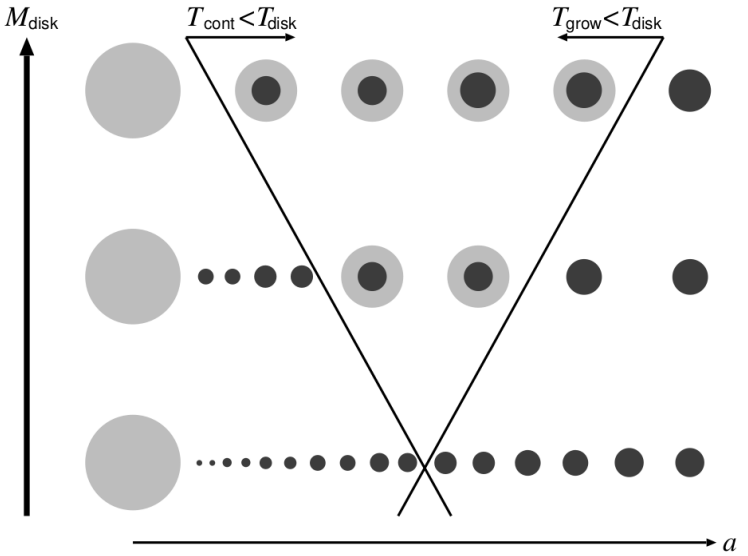
- 等質量だったものが累積質量では $n \propto m^{-1.5}$ にまず進化
- もっとも重い一つがさらに成長
- 基本的に、Wetherill and Stewart 1989 の結果を確認

寡占的成長



- **Kokubo and Ida 1998**
- 少し広い領域を計算
- ほぼ同じ質量の原始惑星がほぼ等間隔に並ぶ
- 大雑把には、この間隔にある質量を集める、ということで原始惑星の質量が決まる。

大領域の計算



- **Kokubo and Ida 2002**
- さらに広い領域を計算
- ディスクの面密度と最終状態の関係を「予想」

暴走的成長＋寡占的成長

- 形成時間の問題 (特に木星型) を解決 (？)
- 地球型惑星: 原始惑星からさらに作らないといけない
  - 少数多体問題。理論的理解も計算も難しい
  - 普通にやると、地球が作れないわけではないが現在のような離心率の小さい状態にはなかなかならない
  - 色々なモデルが提案されている

問題は形成時間だけ？

実はなんとか問題というのは他にもある

- ダスト落下問題 (微惑星形成問題)
- 惑星落下問題

ダスト落下問題

- ダストは最初は小さい。これが原始太陽系星雲の中で衝突･合体で成長していくと考えると、途中の１メートルくらいになったところでガスの抵抗でエネルギー、角運動量を失って、太陽のほうに落ちてしまう。
- 落ちないようにするのが、「自己重力不安定モデル」。合体とかする前に赤道面に薄い層を作り、それが自己重力で一気に分裂、いきなりキロメートルサイズになるとする。
- 静かに赤道面につもるのは無理 (乱流が起こるはず) という批判あり
- ガス抵抗は普通の流体力学的抵抗

未解決の問題

惑星落下問題

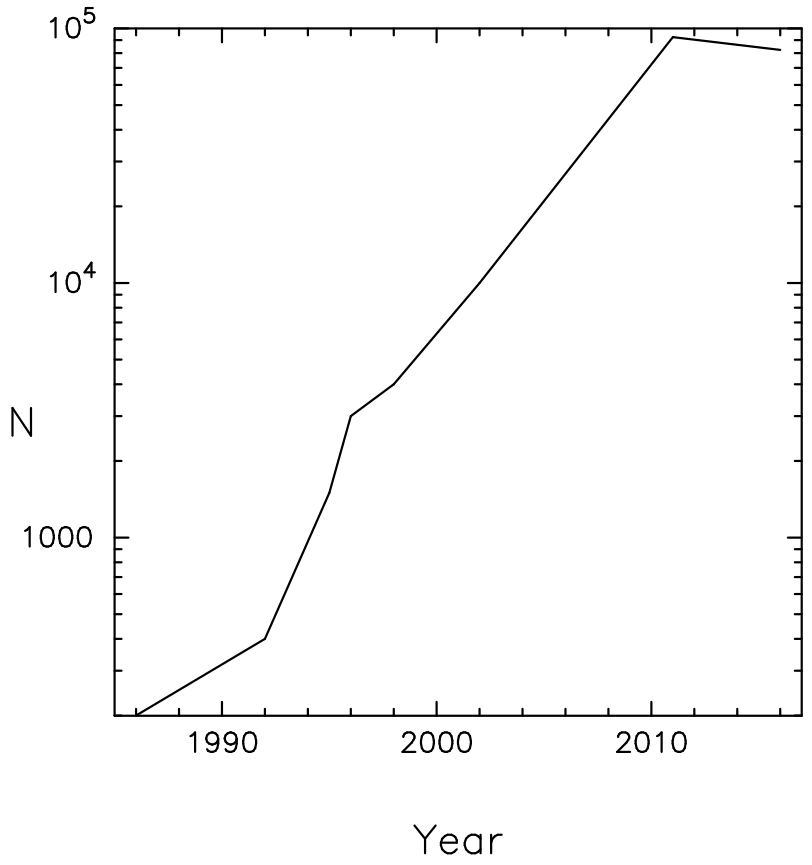
- 微惑星が原始惑星に成長していく途中で、やっぱりガスの抵抗でエネルギー、角運動量を失って、太陽のほうに落ちてしまう。
- 落ちないようにする都合の良いモデルもあまりない
- ガス抵抗は重力相互作用によるもの。

これも未解決

2002年以降

- 個人的な印象としては停滞感。
- 系外惑星の観測、原始惑星系円盤の観測はどんどん進むが、、、
- シミュレーションの規模があんまり上がってない
- なので、できることがあんまり増えてない

シミュレーションの粒子数



- 30 年で 300 倍くらい
- 計算機は 7 桁くらい速くなった
- 宇宙論的 N 体は 8 桁粒子数増えた、、、

手法

各粒子の運動方程式

$$\frac{d^2 \boldsymbol{x}_i}{dt^2} = \sum_{j \neq i} G m_j \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^3}, \tag{1}$$

数値計算は基本的にはこれを使う

計算法 — 時間領域

算数としては、単に常微分方程式の初期値問題の数値解。
ナイーブに考えると、いろんな公式がライブラリであるので、それを使えば済みそうな気がする。
それだけでは済まないのが問題。
済まない理由:

- 粒子によって非常に大きく軌道のタイムスケールが違うことがある
- 連星とかそういったものができる

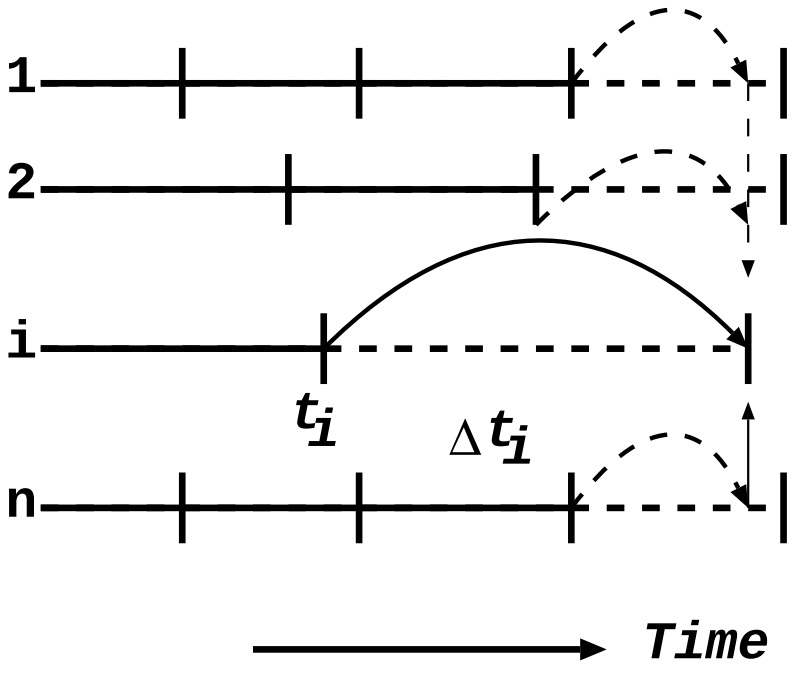
独立時間刻みの原理

粒子毎にばらばらの時刻 t_i と時間ステップ Δt_i を与える

1. $t_i + \Delta t_i$ が最小の粒子を選ぶ。
2. その粒子の軌道を新しい時刻まで積分する。
3. その粒子の新しい時間刻みを決める。
4. ステップ 1 に戻る。

ある粒子の時刻 $t_i + \Delta t_i$ で他の粒子の位置が高精度で必要:
予測子・修正子型の公式を使う。

独立時間刻み



(Aarseth 1963)

- 各粒子にそれぞれ時刻と時間刻みを与える
- 「イベント駆動」時間積分 — $t_i + \Delta t_i$ がもっとも小さい粒子が積分される

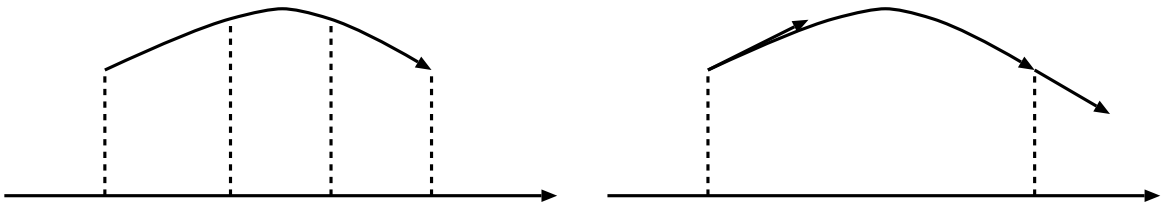
時間積分公式に対する要請

- 高次の予測子が必要 (他の粒子の位置が必要)
- 可変時間刻みが必要
- 積分区間の途中で加速度を計算するような方法は使えない。
  - 線形多段階法は使える
  - ルンゲ・クッタは使えない
  - シンプレクティック法は単純には使えない

時間積分公式

次数： 「4次くらいが適当」(JM 1990)。
もっと高いほうが良い: (Nitadori and JM 2007)

- **Aarseth scheme (Aarseth 1963):** 可変刻みのアダムス法、 PEC モード、4次、2階の方程式用。
- **Hermite scheme (JM 1990):** ラグランジュ補間（ニュートン補間）の代わりに、加速度の一階時間導関数も使ってエルミート補間を構成。



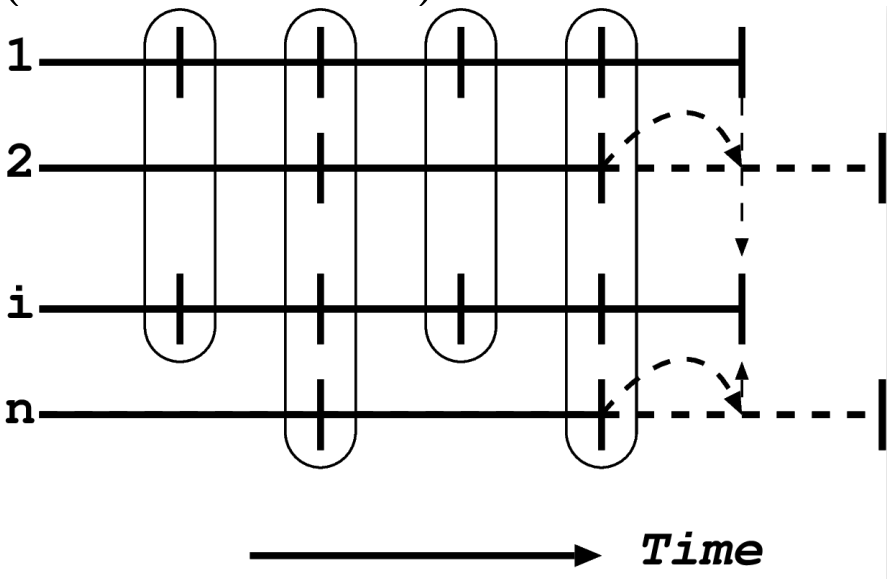
高次エルミート

Nitadori and JM 2007

- **2**階導関数まで直接計算してエルミート補間: **6**次公式
- **3**階導関数まで直接計算してエルミート補間: **8**次公式
- 予測子は前のステップの値を使って構成

ブロック時間刻み法

(McMillan 1986) ベクター／パラレル計算機のための改良



- 時間刻みを 2^{-k} に制限する。
- $t_i + \Delta t_i$ が同じ（Δt_i は違ってもよい）粒子は同時に積分される。

$O(N_c^{2/3})$ 個の粒子（N_c は高密度コアのなかの粒子数）を並列計算

— そんなに大きな数ではない。

計算法 — 空間領域

運動方程式の右辺をどうやって評価するか？という問題。
現在、宇宙論等で広く使われている: **Barnes-Hut** ツリー法。

- 基本的な考え方: 遠くのほうはグループにまとめて重心とか多重展開を評価
- そのためにツリー構造 (八分木) を使う
- 大規模並列化も昔から研究あり。**AICS** ではツリー法にも対応する粒子系シミュレータ開発フレームワークを開発 (**FDPS**)

ツリー法の効果

- 計算量のオーダーが $O(N^2)$ から $O(N \log N)$ に減る。
- 現状では、例えば天文台の Cray 1024 コアを使って 2048^3 粒子が 1 ステップ数分とか。
- もしも直接計算したら 1 ステップ数十年かかる。

独立時間刻みとツリー法の組合せ

- 原理的にはこれが望ましいに決まっている
- 研究も昔からある。**McMillan and Aarseth 1993** とか
- (牧野は **1987** 年あたりに色々やったけど論文書いてない)
- あまり上手くいっていない

問題点:

- 現在の殆どの実装は、一番短いステップ毎にツリーを作り直す。そうすると、ツリーを作る時間が全部になって速度があがらない。
- 部分的にツリーを作り直すとかも試みられているが、特に並列化と組み合せるとコードが複雑になりすぎて手に負えない。

ハミルトニアン分割に基づく方法

MVS (混合変数シンプレクティック) に類似

単純な陽的シンプレクティック: ハミルトニアンを運動エネルギーとポテンシャルに分解、交互に積分

MVS: 惑星の運動を、太陽の回りのケプラー運動とそれ以外の相互作用に分解。

MVS のさらに発展。粒子間重力を

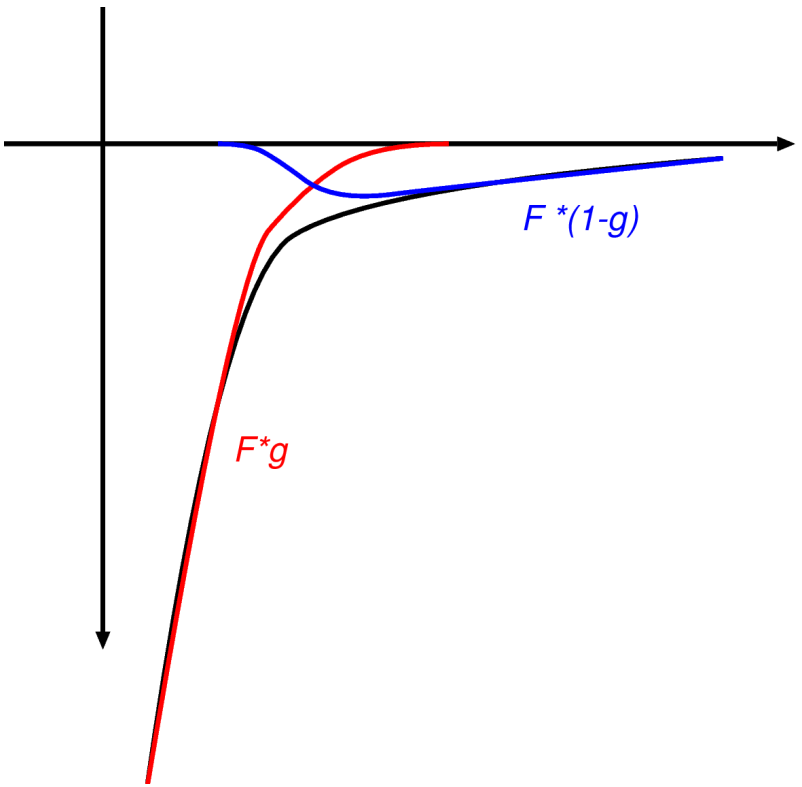
- 近距離分
- 長距離分

に分割、長距離分はシンプレクティック。近距離分は高次精度公式で。

具体的には

2粒子間の重力を形式的に2つのタームに分ける。

$$\boldsymbol{F_{ij} = -Gm_i m_j \frac{r_{ij}}{|r_{ij}|^3} = F_{ij}(1 - g(|r_{ij}|) + F_{ij} g(|r_{ij}|)}$$



こんな感じに分解

- $\boldsymbol{F * g}$ + 運動エネルギー を独立時間刻みで高精度に積分
- $\boldsymbol{F * (1 - g)}$ はツリー+リープフロッグで積分 (こっちはシンプレクティック)
- $\boldsymbol{g}$ はコンパクトサポートで、積分公式に必要な回数だけ微分できる必要あり

開発状況

Oshino et al. 2011, Iwasawa et al 2016

- とりあえず、惑星形成の計算向けに実装、テスト中
- 上手く動いているような気がする。
- **MVS** と同じく、太陽重力は高精度側に入れる。
- **FDPS** を使うことで大規模並列化に対応

計算量を粒子数の 1 乗にできる目処はたった。10^{10} 粒子くらいまではできるように。

そこまで増やす必要あるのか？

- やってみないと分からない
- 少なくとも、衝突による破壊、破壊されてできた小さな粒子のガスとの相互作用、それらの再集積は意味があるいれかたをする必要がある (現在は単に無視している)
- ガス円盤を **self-consistent** に扱って、ダストとの相互作用を正しく取扱う必要もおそらくある (そもそも微惑星ができない問題との関連)

本当は粒子数がいくらあっても足りない

適合性のある粒子法流体スキーム

ここでの適合性: 偏微分方程式に対する差分スキームの適合性

「格子幅 (粒子法なら粒子間隔) が 0 の極限で差分スキームが偏微分方程式に収束すること」 (数値解が収束することよりゆるい条件)

数値スキームに適合性があるのは最低限の条件ではないか?

ところが

圧縮性流体に対して広く使われている SPH 法には適合性がない

SPH 法に適合性がない理由

SPHの基本式:物理量 $\boldsymbol{f}$ の推定

$$\langle \boldsymbol{f} \rangle(\boldsymbol{x}) = \int \boldsymbol{f}(\boldsymbol{x}')\boldsymbol{W}(\boldsymbol{x}-\boldsymbol{x}')d\boldsymbol{x}'. \tag{2}$$

密度の推定

$$\boldsymbol{\rho}(\boldsymbol{x}) = \sum_j \boldsymbol{m}_j \boldsymbol{W}(\boldsymbol{x}-\boldsymbol{x}_j), \tag{3}$$

SPH 近似

$$\langle \boldsymbol{f} \rangle = \sum_j \boldsymbol{m}_j \frac{\boldsymbol{f}_j}{\boldsymbol{\rho}(\boldsymbol{x})} \boldsymbol{W}(\boldsymbol{x}-\boldsymbol{x}_j). \tag{4}$$

ここで $\boldsymbol{W}$ はカーネル関数。固定形状 (空間スケールは適応的な場合もある)

根本的な問題

密度の式

$$\boldsymbol{\rho}(\boldsymbol{x}) = \sum_{\boldsymbol{j}} \boldsymbol{m}_j \boldsymbol{W}(\boldsymbol{x} - \boldsymbol{x}_j) \tag{5}$$

が、粒子配置から密度を与える形式になっている。

- 一様密度の状態から、一様密度を保つ流れ場に沿って粒子を動かしただけで密度推定は一様からずれる
- 密度の不連続面 (例えば接触不連続) を適切に表現する方法がない
- そもそも境界条件を表現できないし、境界近くでの密度推定がデタラメになる

では何故そんな方法が使われているのか

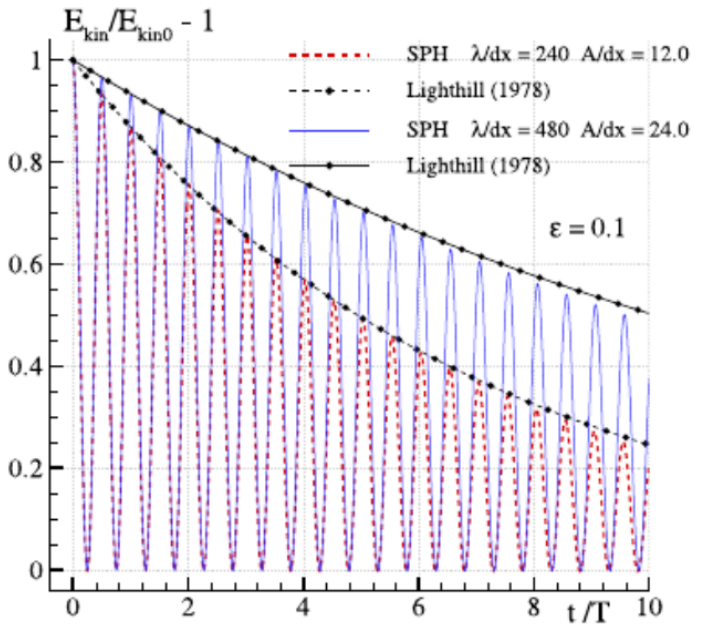
- 密度変化の大きな圧縮性流体に対して他に良い方法が知られていなかった
- 保存形式であり、質量・運動量・エネルギーが保存する。

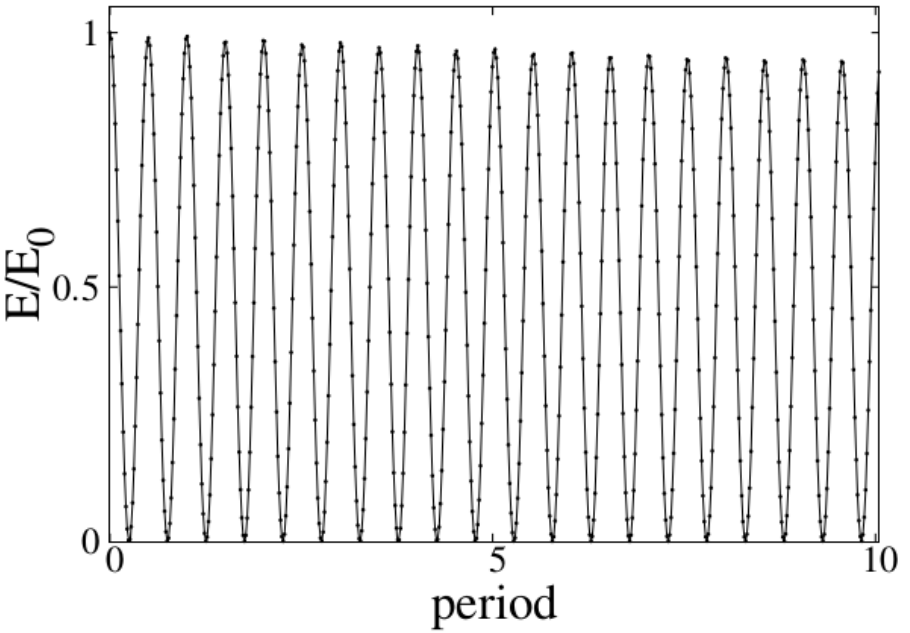
我々の提案する方法

Yamamoto et al. arXiv:1701.05316, PASJ accepted

- カーネル関数をウェイトにした高次多項式基底による最小自乗フィットから直接差分スキームを構成する
- 「粒子」は**SPH**と違って示量変数をもたない。示強変数のみをもつ
- 衝撃波捕獲は人工粘性でやる。フォンノイマン・リヒトメイヤー型のものを正しく多次元化する (これまで粒子法では正しくやられていなかった)
- 保存形式は諦める。空間・時間高次にすることで保存させる。
  - ラグランジュ形式なので、局所的なフラックスは小さい。このため保存則の破れはあまり見えない

数値例





表面重力波の振幅の時間変化 (数値粘性による減衰)

左:SPH の最良の方法の１つ
ある程度粘性をいれないと
計算が破綻する

右:我々の方法
ずっと小さな粘性でよい

問題点

SPH ではごまかしていたことが全部見える

- 境界条件を与える必要がある (逆にいうと移動境界、自由表面もちゃんと扱える)
- 接触不連続も「境界条件」

↓

普通にやると拘束条件になって時間陰解法が必要。

粒子法流体スキームまとめ

- **SPH** とは違う、ちゃんと適合性をもつ圧縮性流体用ラグランジュ形式粒子法スキームを構築した。
- 空間高次精度を正しく実現した。
- 保存形式ではないが、十分な精度がある。
- 強形式なので連立方程式を解く必要はない。
- 但し、境界条件は (現在のところ) 拘束条件なので時間陰解法が必要。
- 陽解法で境界条件扱えそうな怪しげな方法を定式化はできたので実験中。

FDPSの話(宣伝)

- 粒子系シミュレーション並列化の問題
- 対応
- FDPS の中身
- 性能
- 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

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** その他で計算量減らす)

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

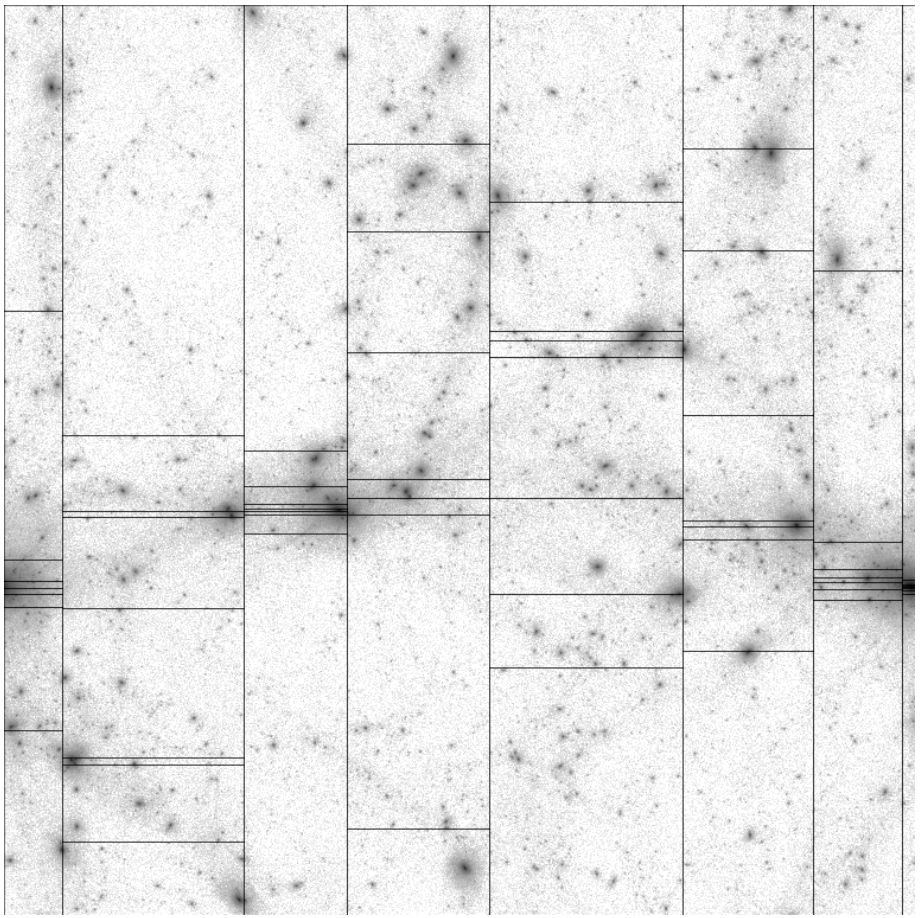
- **API** は **C++** で定義
- ユーザーは
  - 粒子データクラス
  - 粒子間相互作用を計算する関数 (現在のところ、ユーザーが機種毎に最適化。この自動生成は並行して開発中)

  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

- **2015**年度終わりに公開した (**https://github.com/FDPS/F**
- 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
- ユーザーは増加中？
- **GPGPU** 対応、**Fortran** 対応、性能向上等を実施 (あとでもうちょっと詳しく)

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

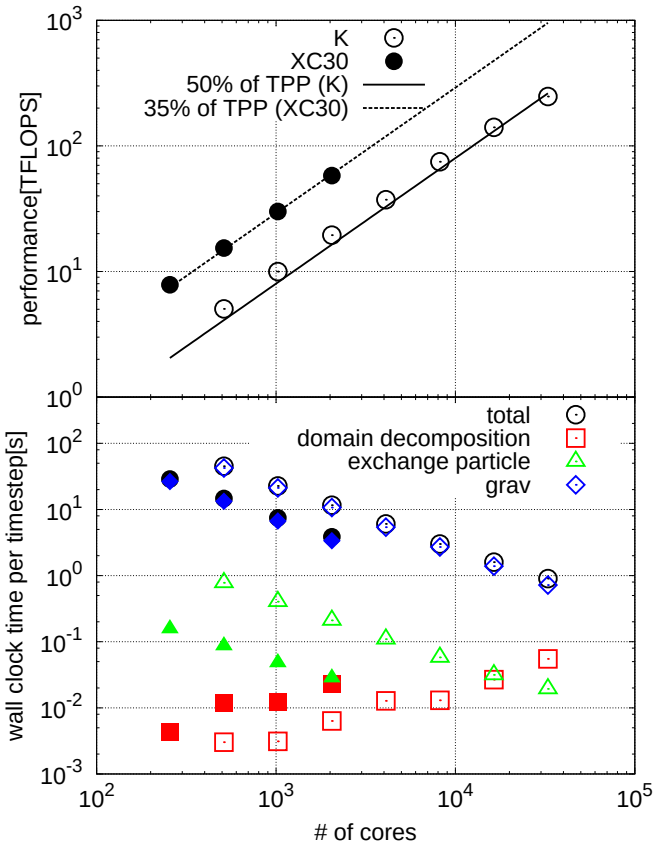
計算時間が均等になるよう領域サイズ調整 (石山他 **2009**、**2012**)

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**)

ユーザーコードは何を書くか？

- 粒子の定義 (クラス) と相互作用関数の定義。相互作用関数は性能出すためには **SIMD** 化とかは必要。並列化は不要 (**FDPS** 側でやる)
- あとは、ユーザーから見るとシリアルコードと同様に、初期条件を読んで時間積分ループ。相互作用計算は **FDPS** の関数を呼ぶと、勝手に領域分割と粒子の再配置とツリー法を並列化した相互作用計算をする。
- 短距離相互作用ならカットオフ (アダプティブなのも可能)、長距離 (べき乗) 力なら **Barnes-Hut** ツリーアルゴリズムが使われる。
- 計算された相互作用使って時間積分し、診断メッセージやスナップショット出力をする。

性能の例



5.5 億粒子での強スケーリング性能

「京」と天文台 X30 の両方で測定

重力のみ、単一の銀河の計算

これくらいの粒子数だと 10 万コアくらいまではスケールする。

理論ピークの 30-50% でる。

Version 2.0

GPGPU 他、アクセラレータに対応した。(**PEZY-SC** にも対応)

- **Version 1.0** の **FDPS** では、ユーザーが定義する相互作用関数は単純に複数の粒子から複数の粒子への相互作用
- **FDPS** は、ツリーを辿って相互作用リストを作り、ユーザー定義の相互作用関数を呼ぶ。
- **Version 2.0** では、「複数の粒子から複数の粒子への相互作用」(相互作用リスト) を複数一度に渡すインターフェースも可能にした
- アクセラレータの高い並列度、大きな呼び出しオーバーヘッドを隠蔽するため

Version 3.0

Fortran で書かれたユーザープログラムに対応した。

- 「**C++** は難しい」(まあ確かに、、、)
- 某 **F** 社のコンパイラは **Fortran** のほうが信用できる
- 実際問題として **Fortran** ユーザーが多い、、、

Fortran 対応の実際

- とはいえ F77 で書くのはさすがにどうかと思う、、、
- というわけで、粒子のデータ型は新しい Fortran の構造型。
- 新しい Fortran の iso_c_binding という機能を使って、Fortran 側で定義した粒子データ型や相互作用関数を C++ 側から参照可能にする
- C++ 側での等価な構造体やそのメンバー関数をソースコードのコメントから生成

Version 4.0 にむけて

- 短距離相互作用の高速化
- 非常に大きなプロセス数での効率向上
- その他非効率的な部分の改善
- 分子動力学での、原子間結合への対応

まとめ

- 惑星形成多体シミュレーションでは、過去 30 年間で粒子数が 300 倍くらいにしかなってない
- これは、精度、時間スケールの問題から、粒子間重力計算に直接法が使われてきたため
- 最近 (といってももう 6 年前だが)、シンプレクティック法の拡張で、遠距離相互作用にはツリー法を使う方法を開発した。並列化のフレームワーク FDPS と組み合わせて、今後は大規模な計算がもうちょっとできるようにする
- FDPS を使えばは相互作用計算する粒子法の大規模並列プログラムを割合簡単に作れる。
- 粒子法での圧縮性流体スキームも、SPH じゃなくて適合性くらいはあるスキームも構築できそう。