富岳における宇宙論的ニュートリノのVlasovシミュレーション

HPC-Phys勉強会
2022年2月4日

筑波大学 計算科学研究センター
吉川耕司

共同研究者：　田中 賢 (京都大学　基礎物理学研究所)

吉田 直紀 (東京大学　Kavli 数物連携宇宙機構)

自己紹介

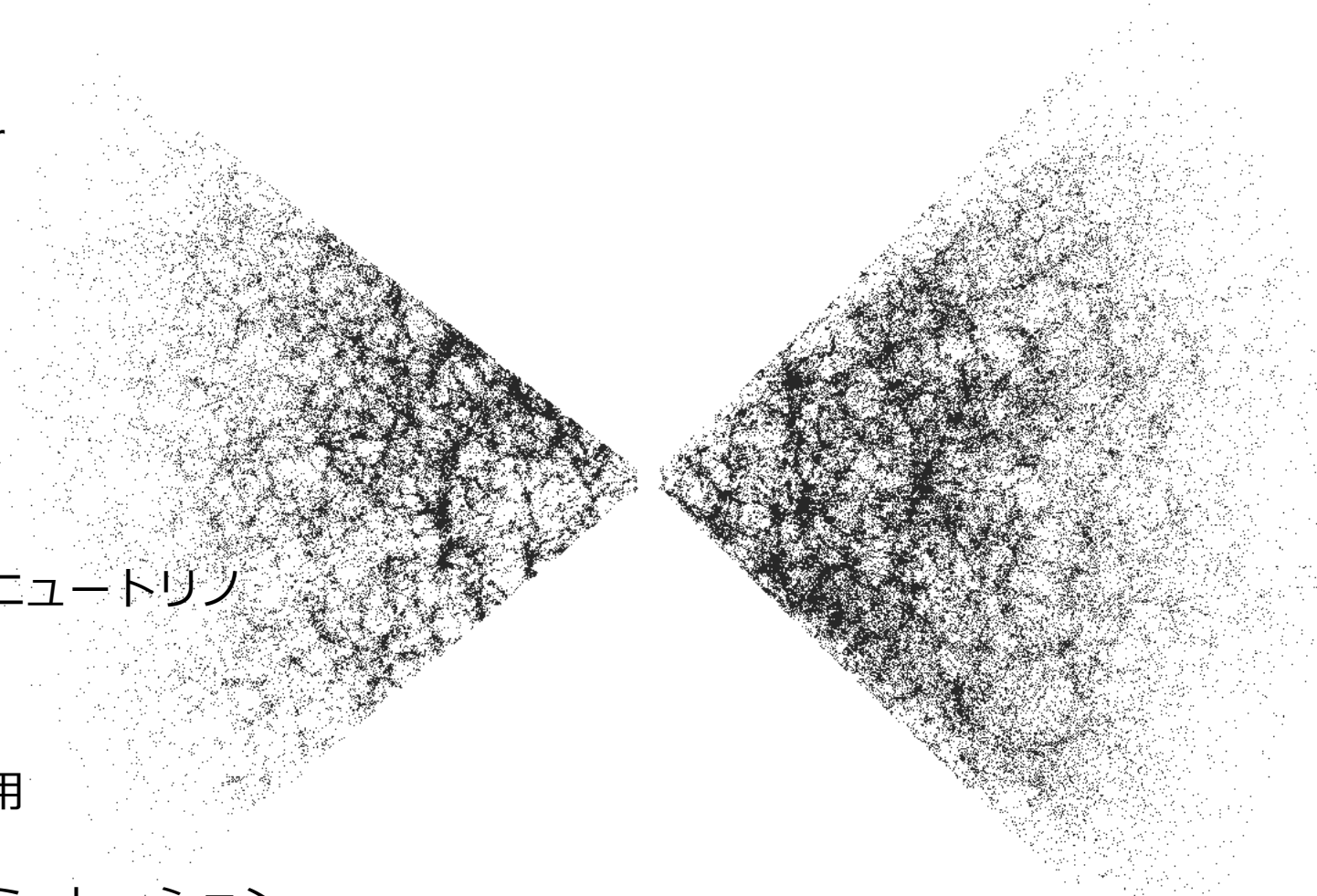
- 筑波大学　計算科学研究センター
  宇宙物理研究部門
- 専門分野
  - 宇宙大規模構造、銀河団、銀河間物質
  - 宇宙物理学におけるHPC
    (重力多体、流体力学、輻射輸送)
- 2012年ころからVlasovシミュレーションの研究を始める
  - Yoshikawa, Yoshida, Umemura, ApJ, 762, 116 (2013)
  - Tanaka, Yoshikawa, Minoshima, Yoshida, ApJ, 849, 76 (2017)
  - Yoshikawa, Tanaka, Saito, ApJ, 904, 159 (2020)

内容

- 数値シミュレーションの概要
- 移流方程式の新しい数値解法：セミ・ラグランジュ法に基づく時間単段・空間高次精度スキーム
- 移流方程式の数値解法におけるSIMD命令の効率的な活用
- テーブル参照を利用した重力多体計算コードのSIMD高速化

宇宙大規模構造形成におけるニュートリノ

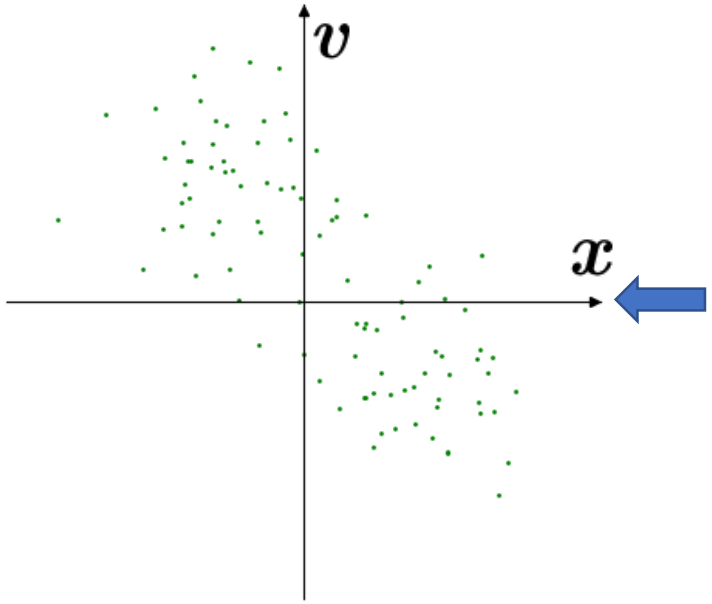
- 宇宙大規模構造
  - 宇宙の質量の大半を占める cold dark matter (CDM) の重力によって形成される構造
  - ダークマター・ダークエネルギーの正体、宇宙初期での密度揺らぎを調べる手掛かり
- 宇宙論的ニュートリノ
  - 宇宙初期において他の成分から脱結合したニュートリノ
  - 質量を持つため現在の宇宙では非相対論的
  - ニュートリノとダークマターの重力相互作用
- ニュートリノを取り入れた大規模構造形成シミュレーション
  - これまでの研究はN体シミュレーション



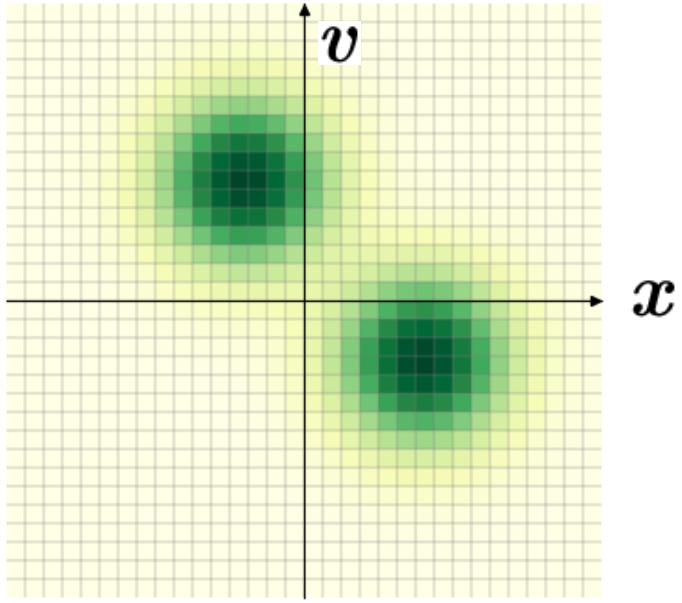
Credit: 2dF Galaxy Redshift Survey

Vlasov シミュレーション

N体シミュレーション



Vlasov シミュレーション



■ 運動方程式

$$\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}t} = \boldsymbol{v}_i$$

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\nabla\phi(\boldsymbol{x}_i)$$

$\phi(\boldsymbol{x})$ ： 重力ポテンシャル

$f(\boldsymbol{x}, \boldsymbol{v}, t)$: 6次元位相空間における分布関数

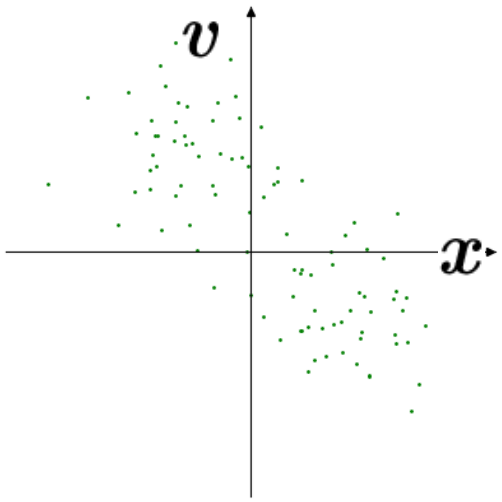
■ Vlasov 方程式
(無衝突 ボルツマン方程式)

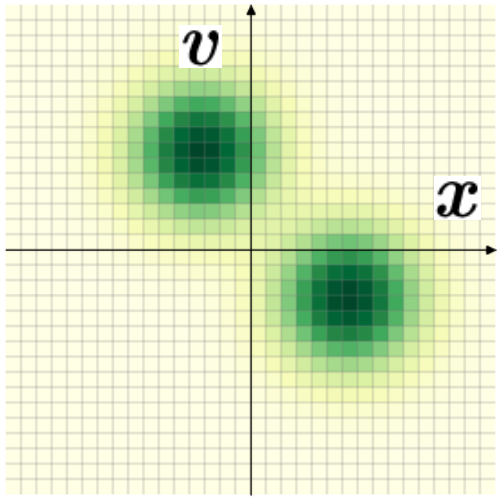
$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{x}} - \nabla\phi \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0$$

✓ 物質分布を6次元位相空間で離散化

✓ Vlasov方程式を有限体積法で解く

N-body vs Vlasov





- N体シミュレーション
  - ✓ 密度の高いところで空間分解能が高い
  - ✓ 多数の粒子数を扱うための計算手法・並列化手法が良く研究されている
  - ✓ 物質分布の統計的なサンプリングによるショットノイズが常にある
  - ✓ 自由流減衰などの速度分布のテイル部分が重要となる運動論的な物理過程を解くのが苦手
- Vlasov シミュレーション
  - ✓ 速度分布のテイル部分の情報を持っているので、自由流減衰を正確に解くことができる。
  - ✓ ショットノイズの影響は原理的にない。
  - ✓ 6次元空間を離散化するのに膨大なメモリが必要で空間分解能はあまり良くない
  - ✓ 6次元空間でVlasov方程式を解く計算コストもかなり大きい。
- 長所と短所は互いに相補的で扱う物質の性質で選ぶべき。

Hybrid Vlasov/N-body Simulation

- ダークマターとニュートリノの2成分系での宇宙大規模構造の数値シミュレーション
- ダークマター成分はN体シミュレーションで計算

  速度分散が極めて小さいので速度空間での広がりはほぼ無い

$$\frac{\mathrm{d}^2\boldsymbol{x}_i}{\mathrm{d}t^2} + 2H(t)\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}t} = -\frac{\nabla\phi(\boldsymbol{x}_i)}{a(t)^2}$$

- ニュートリノ成分はVlasovシミュレーションで計算

$$\frac{\partial f}{\partial t} + \frac{\boldsymbol{p}}{a^2}\cdot\frac{\partial f}{\partial \boldsymbol{x}} - \frac{\partial \phi}{\partial \boldsymbol{x}}\cdot\frac{\partial f}{\partial \boldsymbol{p}} = 0$$

- ダークマターとニュートリノが感じる重力ポテンシャル

  ダークマターとニュートリノの密度分布からPoisson 方程式を用いて計算

$$\nabla^2\phi = 4\pi G\bar{\rho}a^2(f_{\mathrm{cdm}}\delta_{\mathrm{cdm}} + f_\nu\delta_\nu)$$

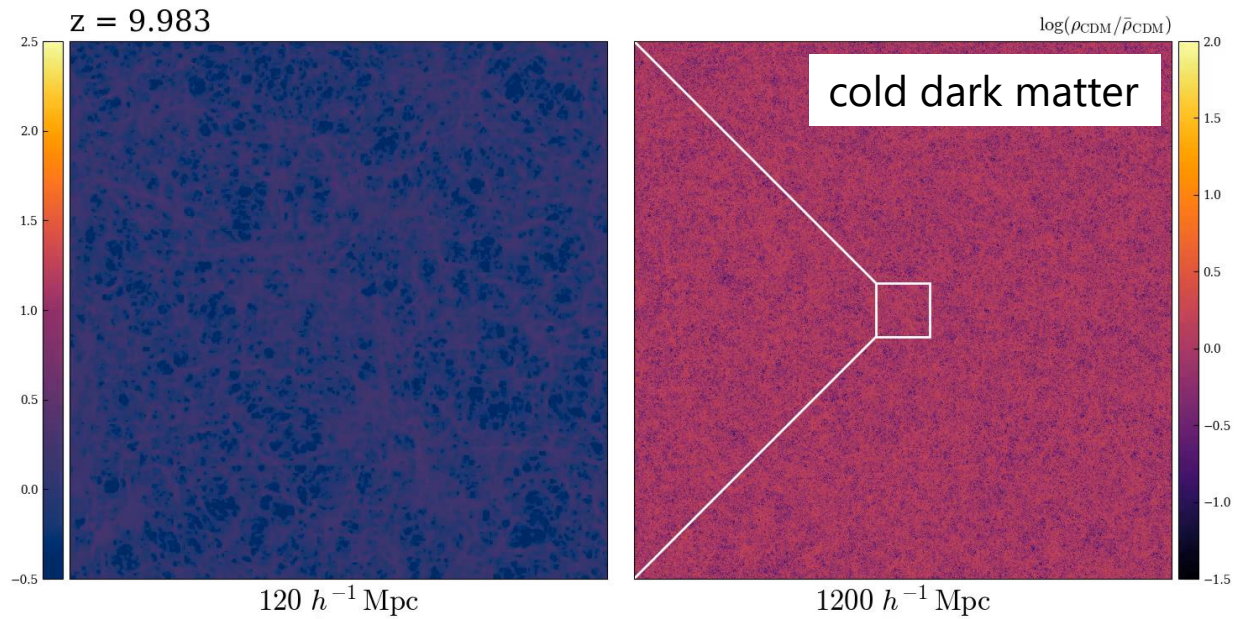
Gordon-Bell Prize

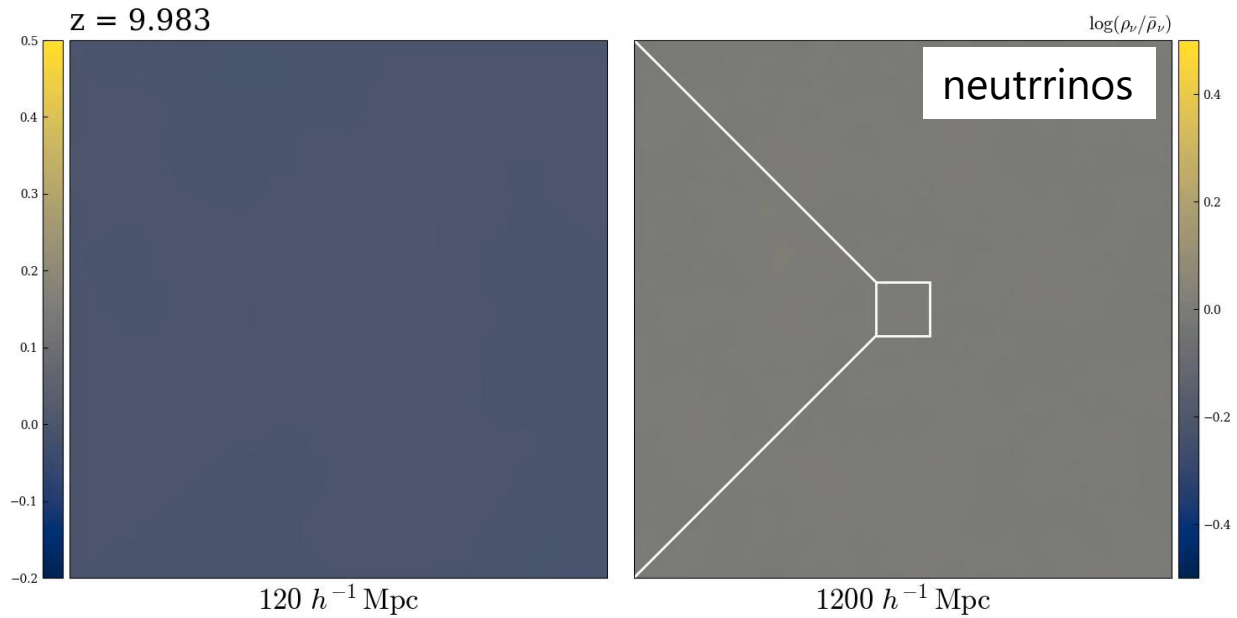
- 「富岳」全系規模の大規模ジョブ実行課題に応募し、採択
  - 2021年2月~3月にかけて得られた最大11万ノードまでの実行結果でACM Gordon-Bell Prizeに論文投稿

    “A 100 Trillion-Grid Vlasov Simulation on Fugaku Supercomputer:
    Large-Scale Distribution of Cosmic Relic Neutrinos in a Six-dimensional Phase Space”

- 2021年7月 ACM Gordon-Bell Prizeのファイナリストに選ばれる
  - 富岳を使ったチームでファイナリストに選ばれたのは我々のチームのみ
  - 2021年7月16日から3日間の全システム(15万ノード)を用いた単独占有実行（しんどかった）
  - 2021年8月上旬に最終版の論文を投稿

    “A **400** Trillion-Grid Vlasov Simulation on Fugaku Supercomputer:
    Large-Scale Distribution of Cosmic Relic Neutrinos in a Six-dimensional Phase Space”


z = 9.983
$\log(\rho_{\rm CDM}/\bar{\rho}_{\rm CDM})$
cold dark matter
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
120 h^{-1} Mpc
1200 h^{-1} Mpc


z = 9.983
$\log(\rho_{\nu}/\bar{\rho}_{\nu})$
neutrrinos
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0.4
0.2
0.0
−0.2
−0.4
120 h^{-1} Mpc
1200 h^{-1} Mpc

移流方程式の新しい数値解法

Vlasov 方程式の数値シミュレーション

- Vlasov方程式の directional splitting

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{x}} - \nabla\phi \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0$$

1次元の移流方程式に方向分割

$$\begin{cases} \dfrac{\partial f}{\partial t} + v_i \dfrac{\partial f}{\partial x_i} = 0 \\ \dfrac{\partial f}{\partial t} - \dfrac{\partial \phi}{\partial x_i}\dfrac{\partial f}{\partial v_i} = 0 \end{cases} \quad (i = 1, 2, 3)$$

有限体積法で離散化

- 時間積分

$$f(\vec{x}, \vec{v}, t^{n+1}) = T_{v_x}(\Delta t/2) T_{v_y}(\Delta t/2) T_{v_z}(\Delta t/2)$$
$$T_x(\Delta t) T_y(\Delta t) T_z(\Delta t)$$
$$T_{v_x}(\Delta t/2) T_{v_y}(\Delta t/2) T_{v_z}(\Delta t/2) \quad f(\vec{x}, \vec{v}, t^n)$$

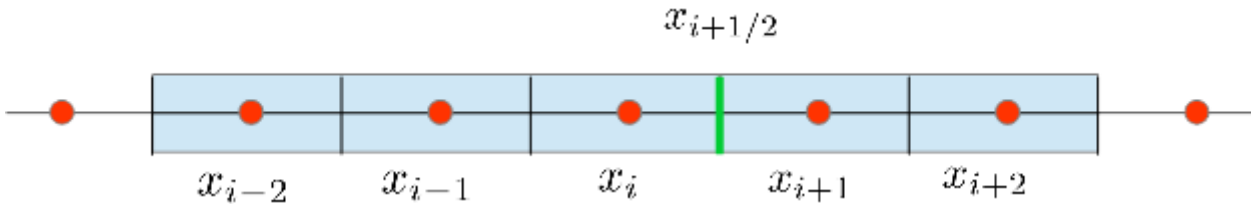
$T_\ell(\Delta t)$ ： ℓ方向への Δt だけの時間発展

Vlasov 方程式の数値シミュレーション

- 1次元移流方程式の数値解法（有限体積法）

$$\frac{\partial f(x,t)}{\partial t} + c\frac{\partial f(x,t)}{\partial x} = 0 \quad \Rightarrow \quad f_i^{n+1} = f_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n)$$

数値流速： $F_{i+1/2}^n = c f_{i+1/2}^n$



- メッシュ中心の物理量からメッシュ境界の $f_{i+1/2}$ を決定
  - 一番簡単なのは1次精度風上差分スキーム

$$f_{i+1/2} = \begin{cases} f_i & (c > 0) \\ f_{i+1} & (c < 0) \end{cases}$$

  - 数値解に対する数学的な要請：単調性、正値性　　流束制限関数とか monotonicity preserving limiter

Vlasovシミュレーションの課題

- 6次元空間を離散化するためメモリ容量が膨大
  - 方向あたりのメッシュ数をあまり増やせないので空間分解能が良くない。
  - だけど数値拡散は減らしたい。

    時間発展につれて数値解が鈍ってくる。(メッシュ数を増やせば簡単に解決するけど、、、)
  - 空間高次精度スキームを採用する。

- 計算量が膨大
  - 単純にメッシュ数が多い
  - 空間高次精度化すると時間精度も同時に向上させないといけない

    空間5次精度 ➡ 時間3次精度が必要

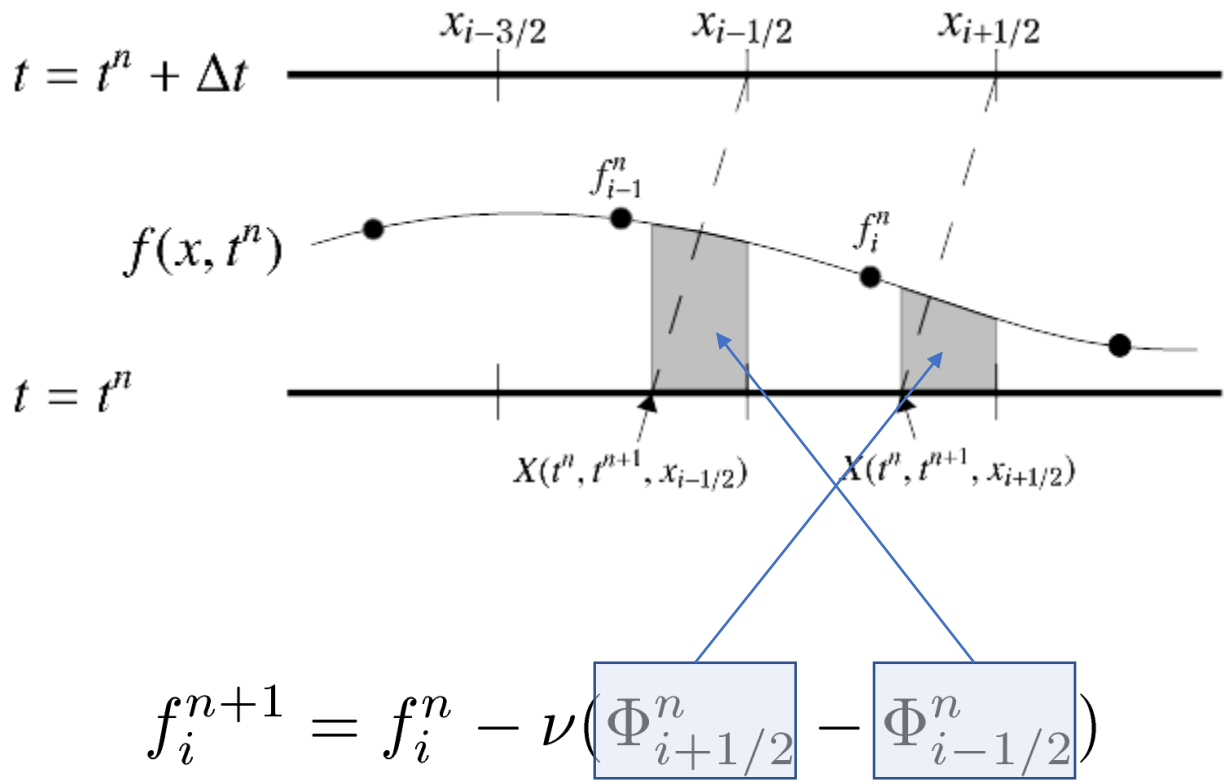
    数値流束を時間ステップあたり3回も計算しないといけない

移流方程式の新しい数値解法

■ 保存型セミ・ラグランジュ法

- メッシュ中心の物理量を多項式で補間してメッシュ境界から次の時間ステップに流入する量を積分することで流束を決定。
- 空間方向の精度がそのまま時間積分の精度になる。
- 空間方向の精度に依らず、数値流束は1回だけ計算すれば良い。(時間単段スキーム)
- 計算量を大幅に削減することが可能
- 通常の流体スキームにも応用可能

  FVS スキームなど



■ Tanaka, Yoshikawa, Minoshima, Yoshida, ApJ, 849, 76 (2017)

空間5次精度：SL-MPP5スキーム

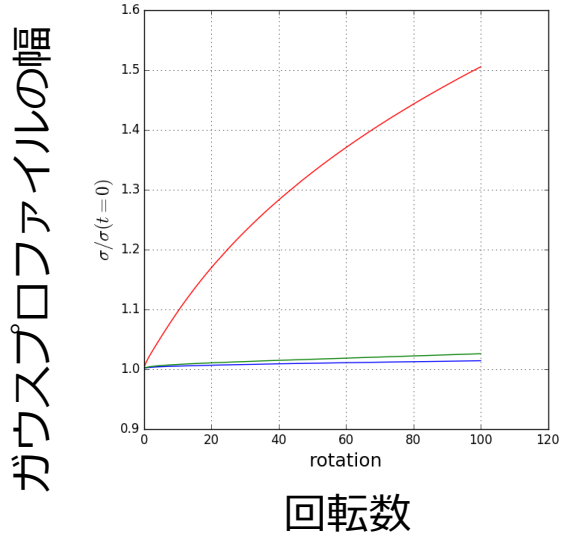
空間7次精度：SL-MPP7スキーム

計算例：2次元剛体回転移流問題

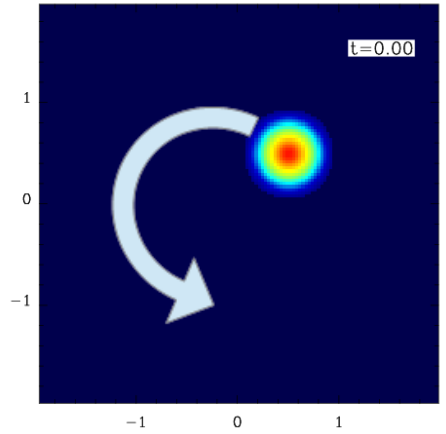
- 移流方程式

$$\frac{\partial f}{\partial t} + (\boldsymbol{v} \times \boldsymbol{\omega}) \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0$$

- 2D ガウシアンプロファイルを剛体回転
- 空間3次精度スキームでは数値拡散によって100回転で分布の幅が50%増しに。
- SL-MPP5スキームとSL-MPP7スキームは、1~2%の増加にとどまる。

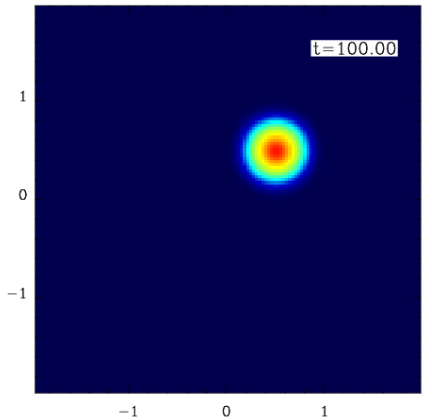


初期条件

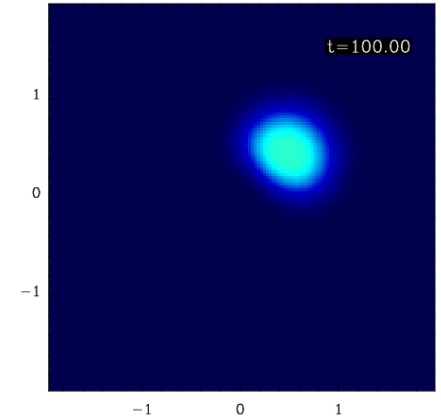


100回転

SL-MPP7スキーム

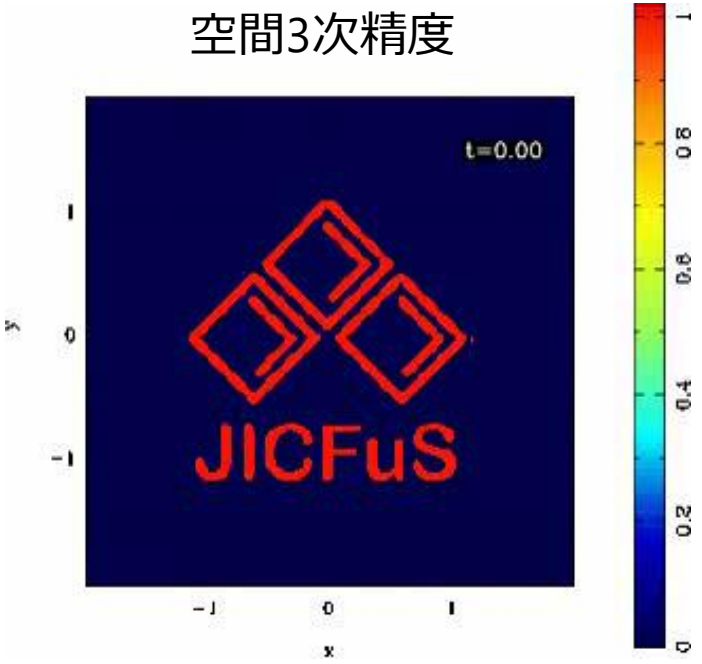


空間3次精度スキーム

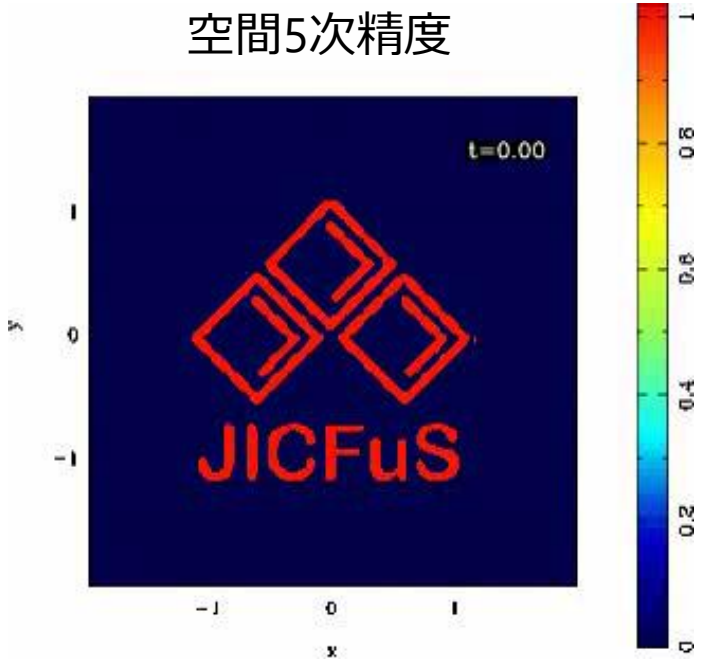


計算例：2次元剛体回転移流問題

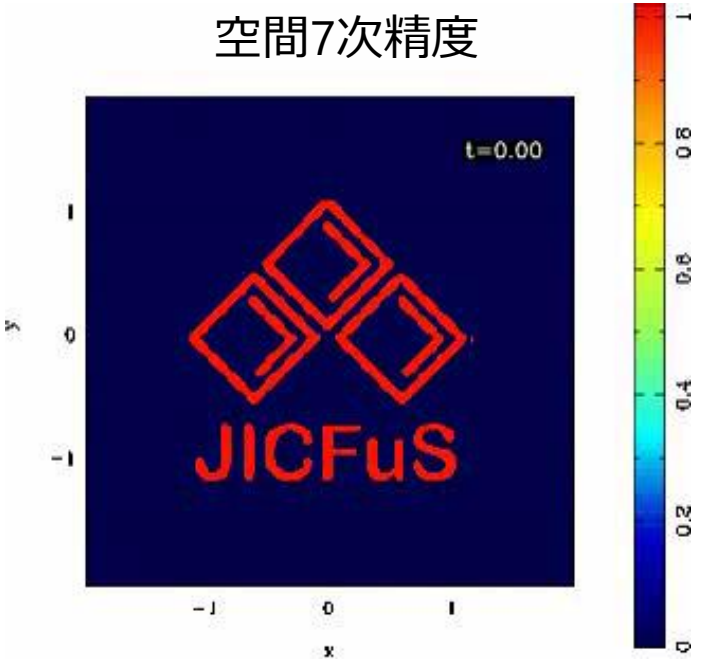
空間3次精度



空間5次精度



空間7次精度

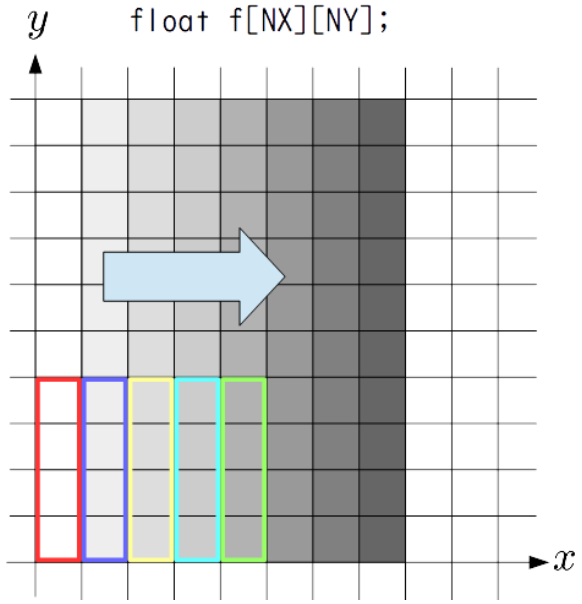


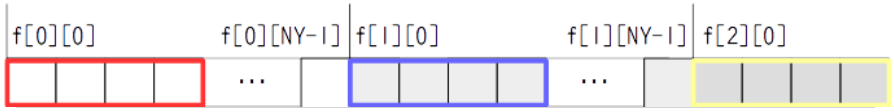
- 空間高次精度スキームの利用はメッシュ数を増やしたのと同等の効果
- 空間3次精度スキームと空間7次精度(SL-MPP7)スキームの計算コストはほぼ同じ

移流スキームにおける
SIMD命令の効率的な活用

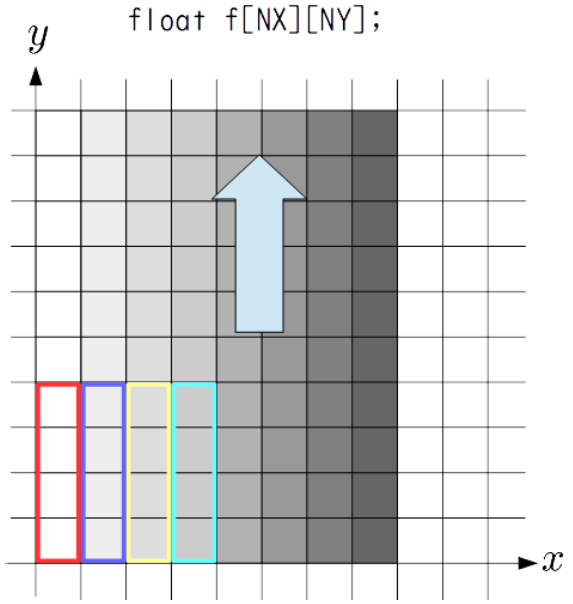
移流スキームのSIMD命令による高速化

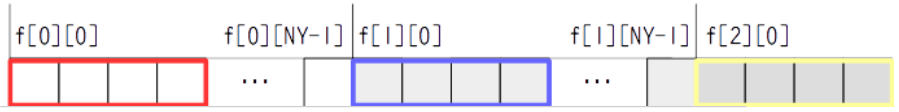
- x軸に沿った移流



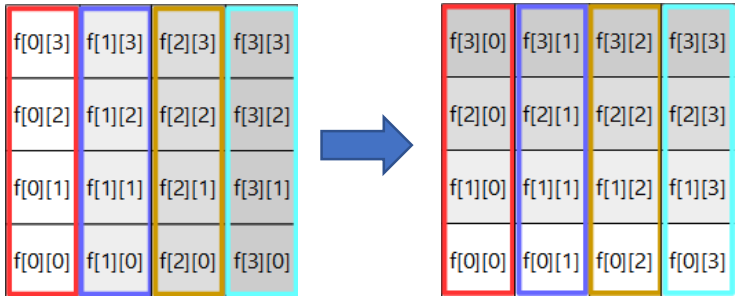


- y軸に沿った移流



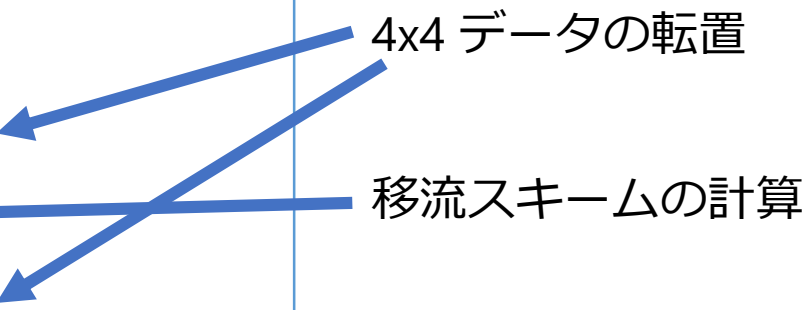


SIMDレジスタ上での
“in-place” なデータの転置



移流スキームのSIMD命令による高速化

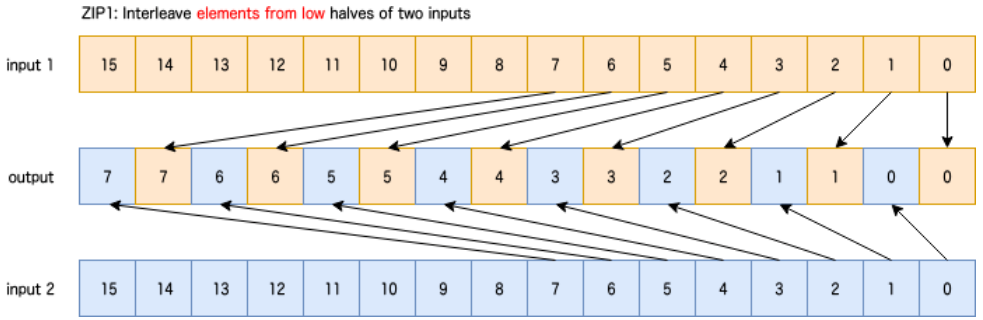
```
#define DF(ix,iy,iz) df[((iz)+NMESH_Z*((iy)+NMESH_Y*(ix)))]

for(int ix=0;ix<NMESH_X;ix++) {
   for(int iz=0;iz<NMESH_Z;iz+=vector_length) {
      svfloat32_t r0, r1, r2, r3, r4;

      r0=svld1(pred, &DF(ix,iy,iz));
      r1=svld1(pred, &DF(ix,iy+1,iz));
      r2=svld1(pred, &DF(ix,iy+2,iz));
      r3=svld1(pred, &DF(ix,iy+3,iz));

      svtranspose_4x4_f32(&r0, &r1, &r2, &r3);

      sweep_1d_sve(&r0, &r1, &r2, &r3);

      svtranspose_4x4_f32(&r0, &r1, &r2, &r3);

      svst1( pred, &DF(ix, iy, iz), r0 );
      svst1( pred, &DF(ix, iy+1, iz), r1 );
      svst1( pred, &DF(ix, iy+2, iz), r2 );
      svst1( pred, &DF(ix, iy+3, iz), r3 );

   }
}
```



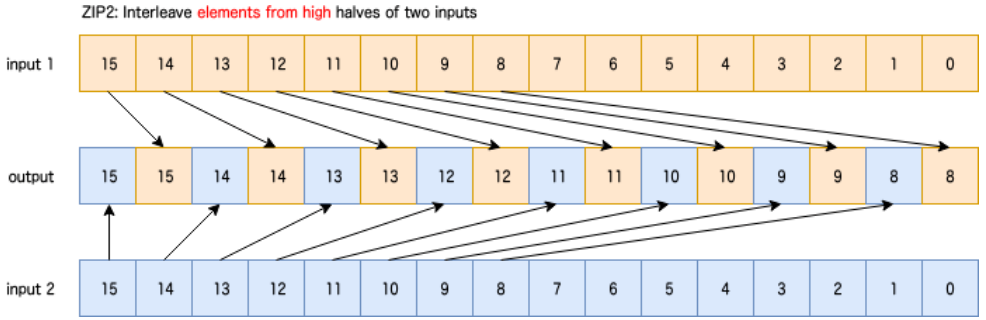
SIMD命令による転置 (SVE命令)

- svzip1 命令と svzip2 命令の組み合わせで実現
  - sv{type} svzip1_f32(sv{type} op1, sv{type} op2)

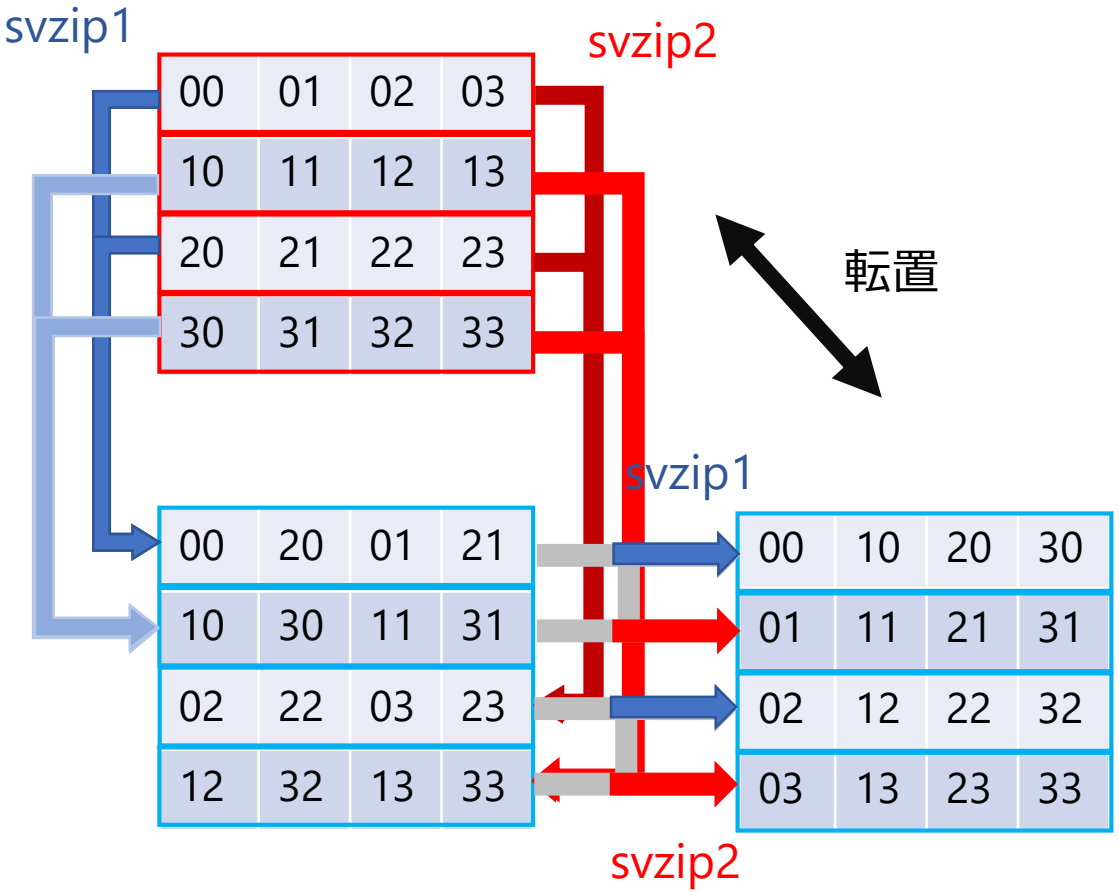
    Intel AVXなどの unpacklo 命令に相当



  - sv{type} svzip2_f32(sv{type} op1, sv{type} op2)



- 4x4データ(4つの4要素レジスタ) の転置の例



- 16x16 データの転置は64命令で実行可能

田中君のqiita記事より転載

富岳における最適化

CMGあたりのVlasovシミュレーションのパフォーマンス

direction	w/o SIMD inst.	w/ SIMD inst.	w/ transposition
v_x	4.84 Gflops	176.7 Gflops	N/A
v_y	7.14 Gflops	233.3 Gflops	N/A
v_z	7.44 Gflops	17.9 Gflops	224.2 Gflops
x	5.51 Gflops	150.0 Gflops	N/A
y	6.88 Gflops	154.1 Gflops	N/A
z	6.50 Gflops	149.2 Gflops	N/A

(cf. 単精度理論ピーク性能 : 1.5Tflops/CMG)

- Vlasovシミュレーションの演算性能は理論性能の12-15%を達成
- (参考)我々の移流スキーム：B/F＝0.23

  A64FXの単精度浮動小数点演算でのHBM2メモリに対するB/Fは0.18

N体シミュレーションにおける SIMD命令の活用

宇宙論的N体シミュレーション

- 宇宙論的N体シミュレーションでの重力計算
  - TreePM法
  - 重力の長距離力 (PM force) は密度場から周期的境界条件の下でPM法で計算
  - 短距離力 (PP force)をカットオフ半径内からの粒子・Treeセルからの寄与として計算

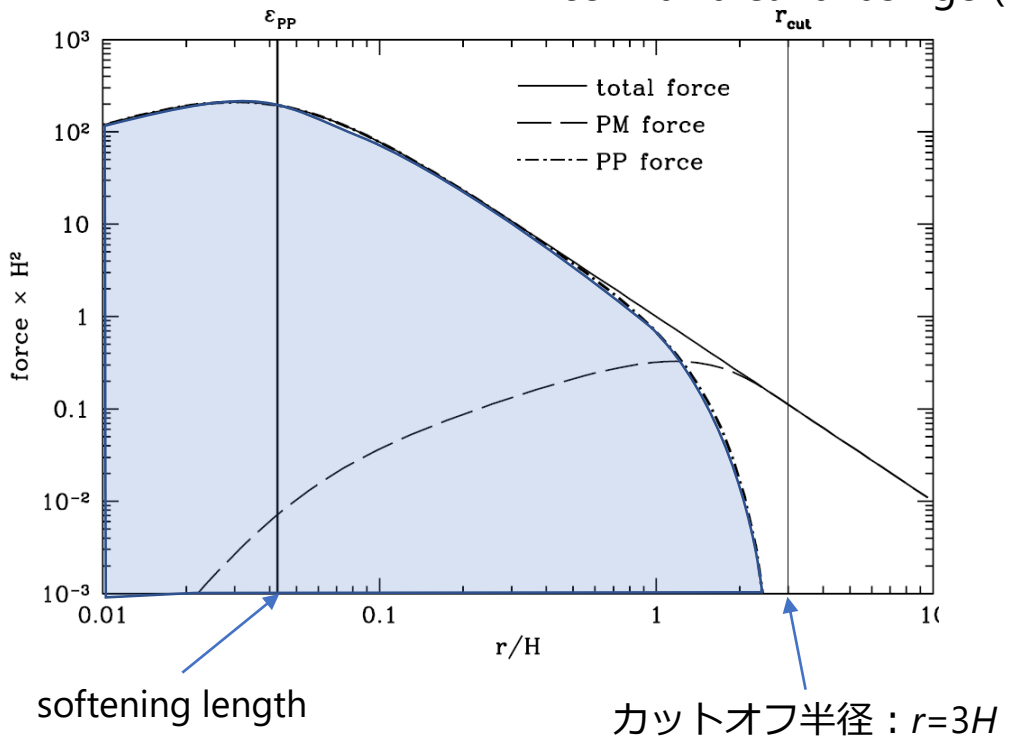
$$\frac{d\boldsymbol{v}_i}{dt} = \sum_{j=1}^{N} g(r_{ij}) \frac{m_j(\boldsymbol{x}_j - \boldsymbol{x}_i)}{(r_{ij}^2 + \epsilon_i^2)^{3/2}}$$

$g(r)$: カットオフ関数

  - softening lengthが一定の場合

$$\frac{d\boldsymbol{v}_i}{dt} = \sum_{j=1}^{N} m_j f(r_{ij})(\boldsymbol{x}_j - \boldsymbol{x}_i)$$

Yoshikawa & Fukushige (2005)



- 重力計算カーネル

N体シミュレーションの計算コストの大部分を占めるので、これの最適化がシミュレーション全体の性能を左右。

重力カーネルの最適化

- SIMD命令による高速化
  - 異なる粒子への重力計算をSIMD実行

    Intel SSE/SSE2 命令　(Nitadori et al. 2006)

    Phantom-GRAPE package

    https://bitbucket.org/kohji/phantom-grape/
  - AVX / AVX-2 命令セットにも対応

    (Tanikawa et al. 2012, 2013)
  - AVX-512命令セットにも対応

    (Yoshikawa et al. 2018)
  - 逆数平方根の高速演算命令も利用
- カットオフ関数の計算方法
  - 多項式計算

$$g_{\mathrm{P3M}}(R) = 1 + R^3\left(-\frac{8}{5} + R^2\left(\frac{8}{5} + R\left(-\frac{1}{2} + R\left(-\frac{12}{35} + R\frac{3}{20}\right)\right)\right)\right) - S^6\left(\frac{3}{35} + R\left(\frac{18}{35} + R\frac{1}{5}\right)\right) \qquad (0 \le R \le 2)$$

$$S \equiv \max(0, R-1)$$

  - 区間多項式近似 (PPA)

$$g_0^{\mathrm{nth}}(r) = \sum_{m=0}^{n} a_m^{(l)}\left(\eta - \frac{2l}{16}\right)^m$$

$$\eta = \frac{2r}{a} \quad l = \lfloor 8\eta \rfloor$$

  - テーブル参照

    $g(r)$ ではなく $f(r)$ をテーブル参照すれば逆数平方根の計算もサボれる。

テーブル参照による重力計算

- サンプリング点の工夫

  - r^2 で等間隔にサンプリング

    softening length よりも内側のサンプリングが粗くなる。

  - r^2 をAffine 変換した s のIEEE754表現の指数部Eビットと仮数部Fビットからテーブルのサンプリング点を決定

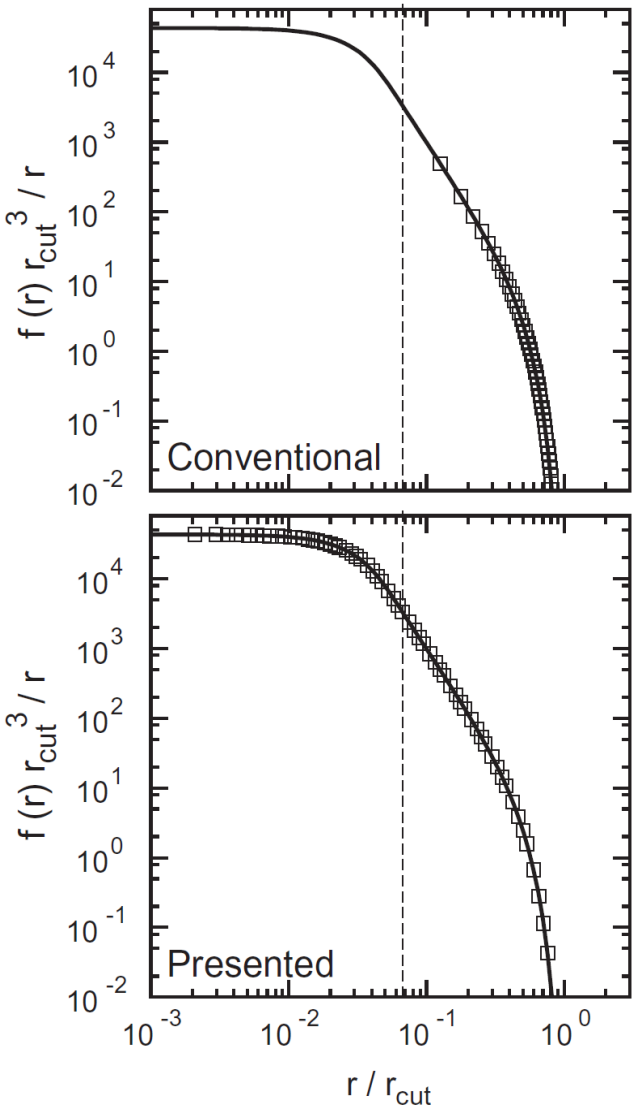
$$s = r^2(s_{\max} - 2)/r_{\mathrm{cut}}^2 + 2$$

E=4, F=6の場合の例

r	s	Exponent bits	Fraction bits	Index
0	2 ($s_{\min}$)	10000000	00000000000000000000000	0
$r_{\mathrm{cut}}/2$	3.2514×10^4	10001101	11111100000001100000000	895
r_{cut}	1.3005×10^5 ($s_{\max}$)	10001111	11111100000000000000000	1023

  - テーブルサイズは 2^{E+F}

Tanikawa et al. (2013)



テーブル参照による重力計算

- テーブル参照と線形補間による加速度の計算

$$f(r_{ij}) = f(r_k) + (s_{ij} - s_k)\frac{f(r_{k+1}) - f(r_k)}{s_{k+1} - s_k}$$

- サンプル点での値と傾きをテーブルにする。

  2次元配列 : force_table[TBL_SIZE][2];

```
for(int i=0,m32.f=2.0f;i<TBL_SIZE;i++,m32.u+=tick) {
   float r = sqrt((m32.f-2.0)/r2scale);
   force_table[i][0]= force_func(r);
}

for(int i=0,m32.f=2.0f;i<TBL_SIZE;i++,m32.u+=tick) {
   float s0=m32.f;
   m32.u += tick
   float s1=m32.f;
   force_table[i][1] =
      (force_table[i+1][0]-force_table[i][0])/(s1-s0);
}
```

- SIMD命令によるテーブル参照

```
int32_t *ptr = (int32_t *)force_table-(1<<(31-(23-FRC_BIT)));

svfloat32_t _dx = svsub_x(prd, _xj, _xi);
svfloat32_t _dz = svsub_x(prd, _yj, _yi);
svfloat32_t _dy = svsub_x(prd, _zj, _zi);

svfloat32_t _rsq = svmad_x(prd, _dx, _dx, 2.0f);
_rsq = svmad_x(prd, _dy, _dy);
_rsq = svmad_x(prd, _dz, _dz);
_rsq = svmin_z(prd, _rsq, _r2cut_xscale2);
                  ビットシフト
svuint32_t _sr = svlsr_x(prd, ¥
                           svreinterpret_u32(_rsq), 23-FRC_BIT);
svuint32_t _sl = svlsl_x(prd, _sr, 22-FRC_BIT);

svfloat32_t _ff = ¥      テーブル参照
      svreinterpret_f32(svld1_gather_index(prd, ptr, _sl);
svfloat32_t _df = ¥
      svreinterpret_f32(svld1_gather_index(prd, ptr+1, _sl);
```

https://bitbucket.org/kohji/workspace/snippets/BrX6Aj/gravity-kernel-code-for-sve-instructions#file-grav_S2_table_test.c

SVE命令を用いた高速化

- A64FXのコアあたりの演算性能
  - テーブル参照 ($f(r)$ をテーブル化)

    SVE命令無し：2.5×10^7 相互作用/sec

    SVE命令あり：1.2×10^9 相互作用/sec
  - 多項式計算 ($g(r)$ の計算)

    SVE命令無し：1.7×10^7 相互作用/sec

    SVE命令あり：6.3×10^8 相互作用/sec
  - 区間多項式近似 ($g(r)$ の計算)

    SVE命令無し：4.3×10^7 相互作用/sec

    SVE命令あり：8.1×10^8 相互作用/sec

- Intel AVX-512命令を用いたテーブル参照による実装との比較

  Intel Core i9 7980XE (base clock: 2.6GHz / 18 cores)
  - 2.6×10^9 interactions/sec/core @ 3.6 GHz (turbo boost)
  - A64FXのクロックは2.0 GHz
  - AVX-512命令で全コア利用時のクロック：1.8GHz

  ソケットあたりではA64FXの方が性能がでている。

- 富岳での宇宙大規模構造形成シミュレーション（GreeMコード）でも利用されている。

まとめ

- 宇宙大規模構造形成におけるダークマターとニュートリノの数値シミュレーション
- ダークマターのN体シミュレーションとニュートリノのVlasovシミュレーション
- Vlasovシミュレーション向けの新しい時間単段の空間高次精度移流スキーム
- 多次元移流シミュレーションでのSIMD命令の効率的利用方法
- 宇宙論的N体シミュレーションでのカットオフ入り多体重力計算におけるSIMD命令による高速化