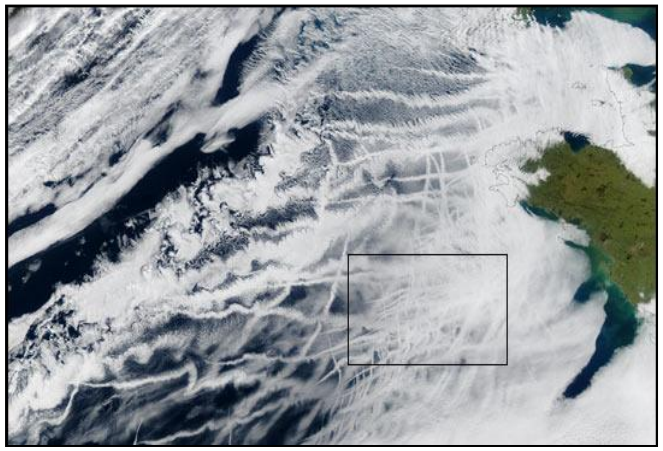
GPU化によってまかなえる雲微物理モデルの計算コスト増加

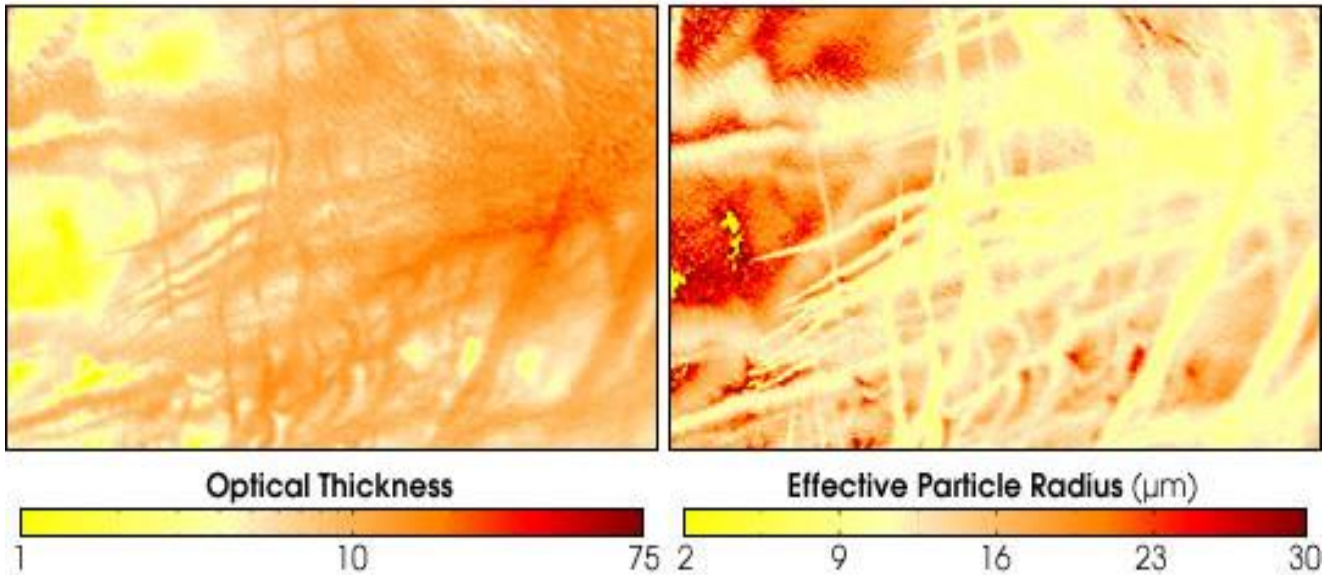
清木達也
小玉知央、中野満寿男、足立幸穂、八代尚

計算科学研究連絡会, 2023/05/09

衝突過程モデリングの重要性と課題



大西洋北部の航跡雲写真　(NASA)



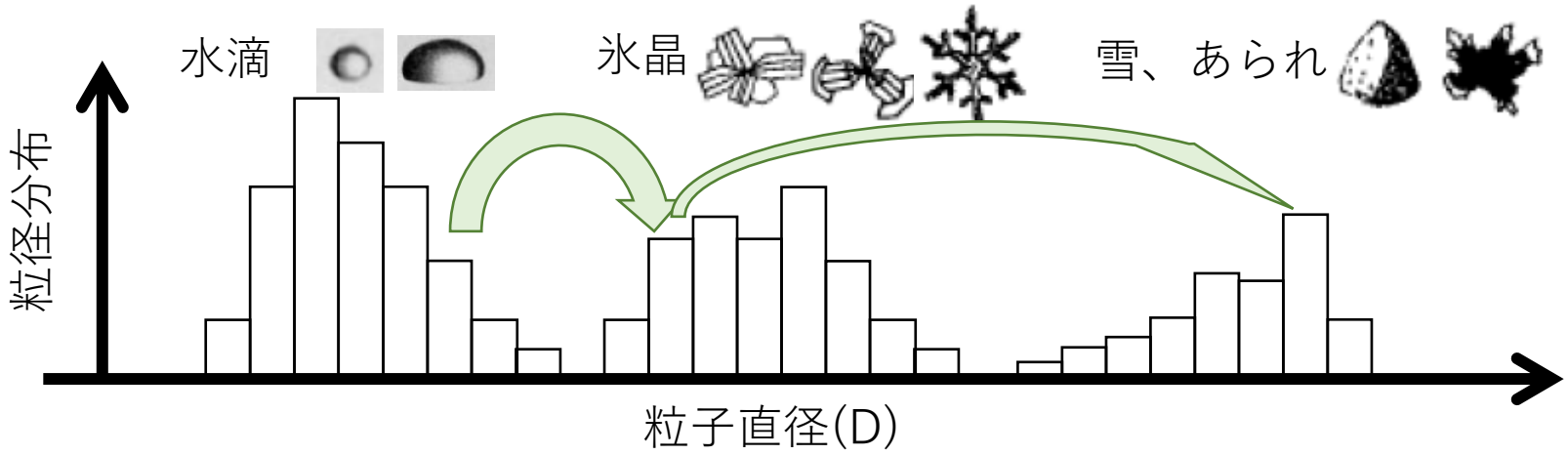
衝突成長：雲の寿命を決める最も速い粒子成長プロセス

- 人為起源エアロゾルによる雲改変
- 巻雲（長寿命、広域）から台風（短期、局所）まで

衝突成長式は全ての「持続時間」をコントロール

➔雲微物理モデルの性能を高めていくうえで高度化は避けられない

衝突過程モデリングの重要性と課題



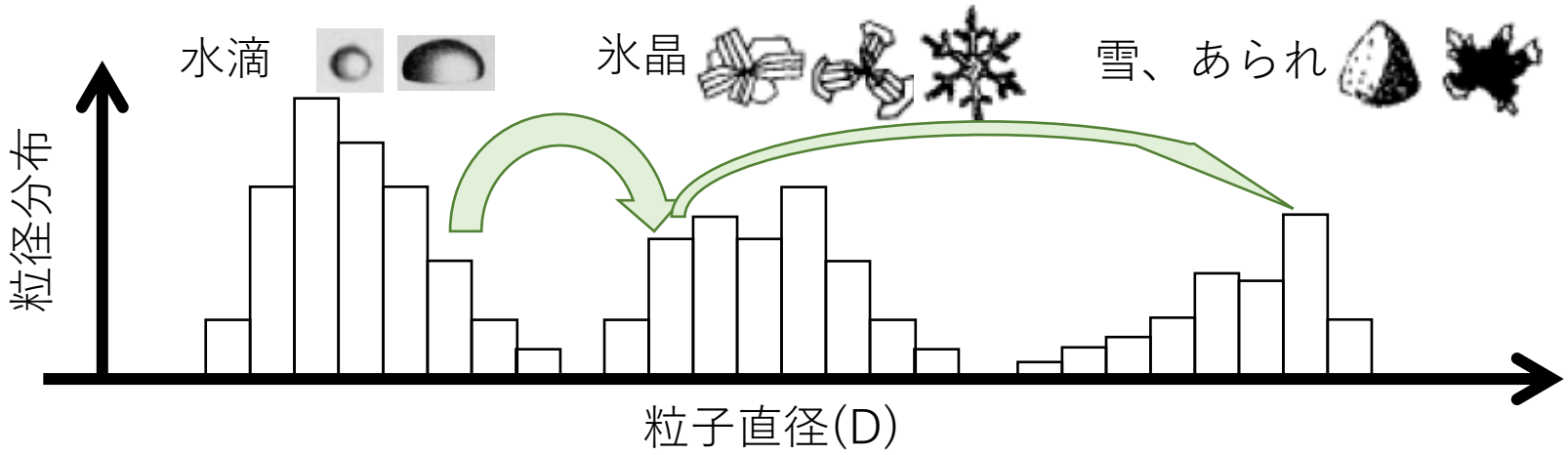
衝突成長の式：粒子種xと粒子種yが衝突して粒子種zができるので、粒子種x, yの質量が減っていく

ぶつかる粒子の直径の組み合わせ：直径の計算サイズL
ぶつかる粒子の種類の組み合わせ：粒子種の分類J

プログラム内の問題サイズ＝L*L*J*J　　（ほかのプロセスの問題サイズはせいぜいL*J）

衝突成長は科学的重要性が最も高い上に、計算コストも最も高い

衝突過程モデリングの重要性と課題



プログラム内の問題サイズ＝L*L*J*J

直径の計算サイズL

ビン法は30程度、バルク法でも2モーメント(2)、３モーメント(3)と粒径情報が増加

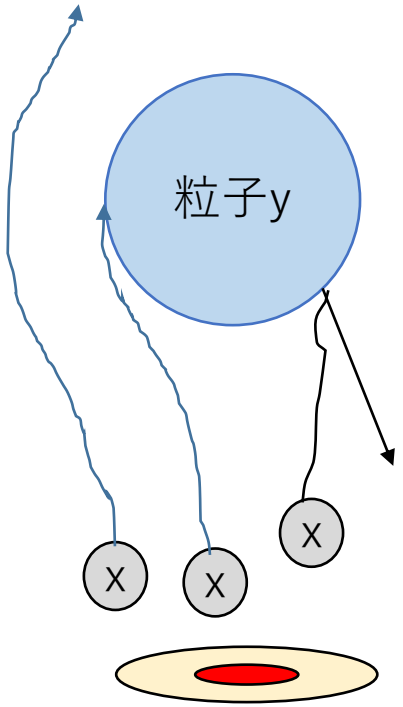
粒子種の分類$J=k^{**}N$ (kは２～１０程度)

属性N=バルク密度、アスペクト比、溶存エアロゾル、同位体比、融解率など

(多次元ビン法、超水適法、P3)

➔雲モデリングの発展に伴い計算コストはどんどん増加する

更なる高度化：捕集効率$E_{collection}$=衝突効率$E_{collision}$*固着効率E_{stick}



$$N_x(D) = f_x(D)\frac{\pi}{4}\left(D_y + D_x\right)^2 |v_y - v_x| dD$$　　粒子yにぶつかる粒子の数N_x

$f_x\, dD$	単位体積当たりの粒子数密度	**E_{stick}補正が必要**
$\frac{\pi}{4}\left(D_y + D_x\right)^2$,	衝突断面積	**$E_{collision}$補正が必要**
$\lvert v_y - v_x \rvert$,	通過距離	

衝突効率$E_{collision}$：断面積の範囲内にいても流れに沿ってぶつからない粒子を評価

➔粒子の大きさ依存性

固着効率E_{stick}：ぶつかっても弾かれる効果を評価

➔粒子の重さ、速度、表面の形状が大事

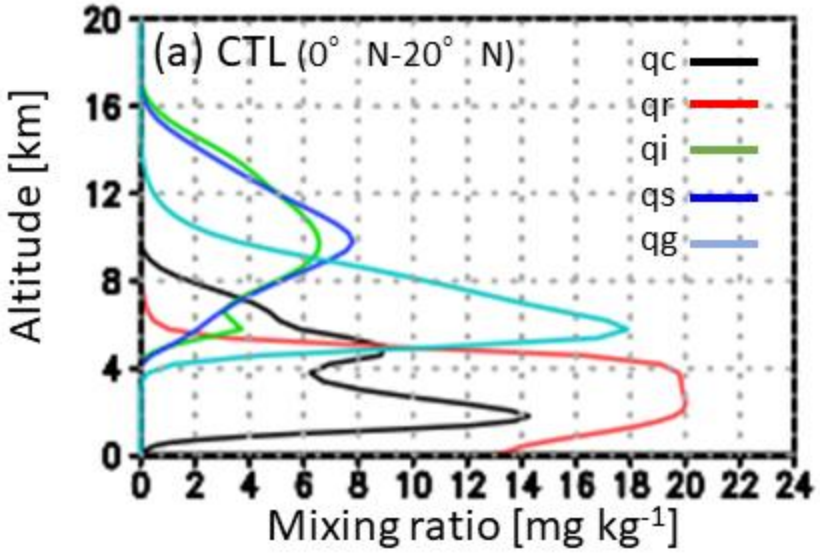
雲微物理	do loop内の行数	衝突の計算コスト	NICAM全体コスト（衝突の比率）
ダブルモーメントバルク法 (Seiki&Ohno,2022)	250行	1	1(26%)
衝突効率の精緻化	250+656行	2.4倍	1.30(46%)
衝突効率＋固着効率	250+656+53行	2.8倍	1.34倍(47%)

更なる高度化：捕集効率$E_{collection}$=衝突効率$E_{collision}$*固着効率E_{stick}

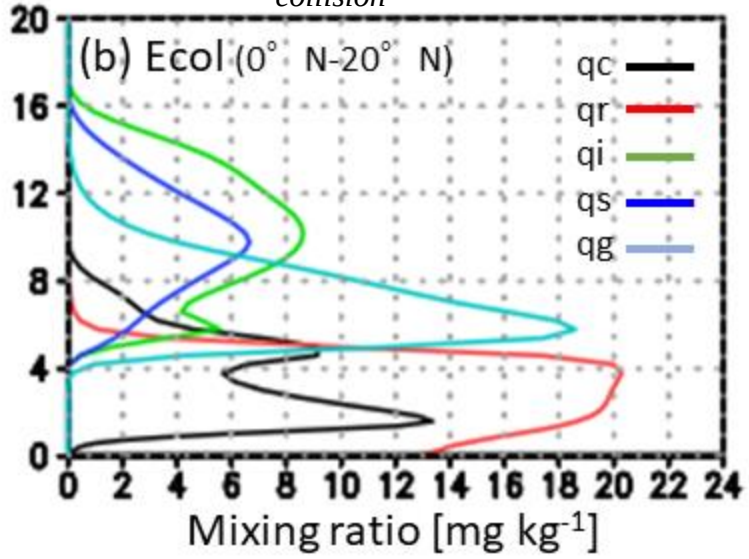
全球影響も計算コストに見合うだけ大きい

全球実験時の熱帯の質量混合比の鉛直分布比較

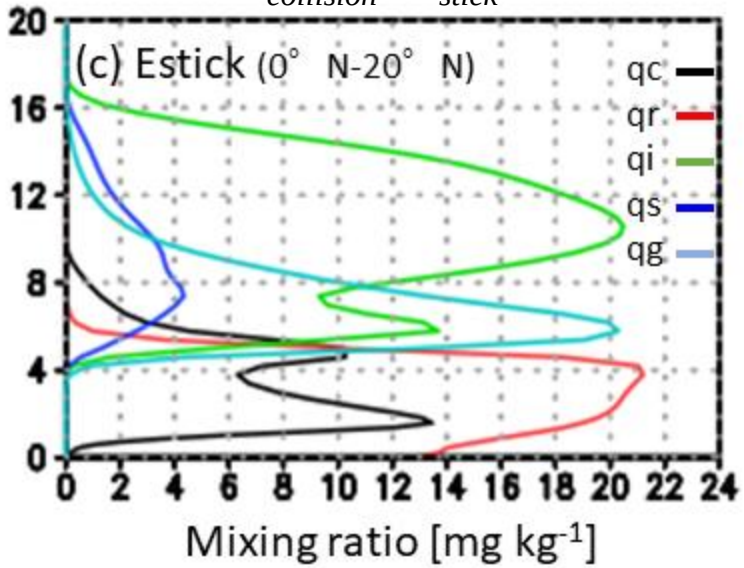
標準実験



$E_{collision}$導入



$E_{collision}$+E_{stick}導入



コスト削減の方針

- パラメタリゼーション（取組中➔コスト半減にする目途がたった）
- GPU加速器の使用

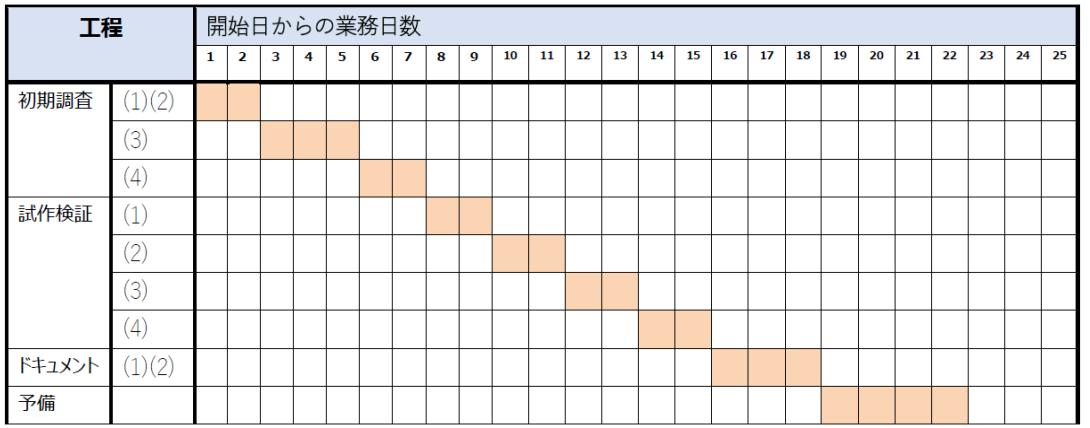
GPU化を企業に委託してノウハウを知ろう！

株式会社　アーク情報システム

1　初期調査
- (1) 調査環境でのコンパイルと実行（プロファイル）
- (2) サブルーチンmp_ndw6ソース調査 メモリ使用量、ループ構成
- (3) アルゴリズム改変の検討
- (4) 精度検証方法の検討

2　試作検証
- (1) OpenACCによるカーネル、データ転送の試作
- (2) （１）の精度および速度検証
- (3) 倍精度から単精度への変更
- (4) （３）の精度および速度検証

3　ドキュメント作成
- (1) 報告書作成
- (2) 試作結果からの考察まとめ



調査＋試作で1 subroutine、納期が２か月間くらい

GPU化発注に向けて用意したもの

1）シングルカラムモデルで特定の物理（＋周辺モジュール）のみを切り出し
- コンパイル1分半
- 実行1秒弱
- どこ(業者のPC)でも動くことを確認

2）ホットスポットの計測
- サブルーチンごとに計算コストを計って、業者にお願いする箇所を決定

3）実際に全球計算で利用する問題サイズにSCMを調整
- 全球14km78層実験の1プロセス辺りの計算コスト（ループ長、メモリ使用量）
  ijdim=4356, kdim=78
- 20 stepの実行で200秒

4）おおまかな計算内容の説明
- ループ内依存関係、計算精度の必要性(倍精度・単精度)

5）想定するGPUの指定
- NVIDIA TESLAシリーズのAmpere世代以降 (e.g., ES4がNVDIA A100)

（改変後ソースコードの著作権の譲渡は仕様書で確認した方が良い）

NICAMの雲微物理モデルNDW6をGPU化

衝突の式：雲微物理とエアロゾルに登場　（1000行位のサブルーチン）

$$\left(\frac{\partial N_x}{\partial t}\right)_{xy} \sim -\Sigma_i\Sigma_j f_x(x_i) f_y(y_j)\pi\left(D_{x,i}+D_{y,j}\right)^2 |v_{x,i}-v_{y,j}|,$$

粒子種の組み合わせ(全部で12ペア)に対して質量、数濃度の生成・消滅項を計算
ij, kの外側に粒径分布関数の数値積分の二重ループ(ijdim=4356, kdim=78, ngmax=4, ngmax=4)

[雲氷と雪の計算例]

```
Do ngx=1, ngmax
Do ngy=1, ngmax
      Do k=1, kdim
      Do ij=1, ijdim
            kernel_is          = 0.25d0*pi*(dai_glx+das_gly)*(dai_g
            PNIacNS2NS(ij,k)   = PNIacNS2NS(ij,k) - kernel_is
            PLIacLS2LS(ij,k)   = PLIacLS2LS(ij,k) - kernel_is*xi_gl
      End do
      End do
End do
End do
```

OpenACCを利用した主なコード改変の例

```
subroutine mixed_phase_collection_bin(…)
 :
 ! Integraion Start
!$acc kernels
!$acc loop independent
 do ngx=1, ngmax
!$acc loop independent
  do ngy=1, ngmax
!$acc loop independent
   do k=kmin,kmax
!$acc loop independent
    do ij=1,ijdim
     :
     ! collection equation
     ! 1.c-g(X=Cloud, Y=Graupel)
     kernel_cg = …
     !$acc atomic update
     PNGacNC2NG(ij,k)=PNGacNC2NG(ij,k)-kernel_cg*dNc_glx*dNg_gly
```

衝突過程のGPU版の実測

表 5.3 サブルーチン mixed_phase_collection_bin の詳細時間（単位：秒）

処理		original	openACC					
			No.1	No2	No.3	No.4	No.5	No.6
mixed_phase_collection_bin		77.060	1.545	1.325	1.507	1.208	1.123	1.136
mpc_initialize		0.592	0.257	0.259	0.255	0.256	0.261	0.259
mpc_datain	**転送(入)**		0.394	0.387	0.397	0.401	0.397	0.410
mpc_calc1st		0.034	0.001	0.001	0.001	0.001	0.001	0.001
mpc_calc2nd	**主計算**	75.946	0.602	0.387	0.563	0.250	0.163	0.165
mpc_calc3rd		0.487	0.004	0.004	0.004	0.004	0.004	0.004
mpc_dataout	**転送(出)**		0.287	0.287	0.287	0.290	0.291	0.291

6段階のGPU高速化技法を適応

オリジナルCPU版　：約80秒
一番原始的なGPU版：約1.5秒
最終的なGPU版　　：約1.1秒

- CPU-GPUデータ転送は0.7秒
- GPUの計算コストは0.16秒➔500倍速い

最も単純・簡単な実装でも十分速い！

（補足）倍精度➔単精度に変更した場合
- GPUの計算時間は26%off　（CPU版はきっちり50%off）
- CPU-GPU通信は50%off

➔GPUのコア一つ一つはそれほど速くないので、メモリバンド幅律速にはなってなさそう

GPUで高速化する原理

テスト環境： Intel-Xeon CPU E5-1650 v4 (3.6GHz, 6core, 12thread)+GPU Quadro GP100
1ノードについているGPUコア数：3584コア
➔依存関係のないdo loopのサイズ(4356*80*4*4=5575680)を単純にコア数で分業するだけ！

GPUに任せられるソースコードの範囲

１GPUメモリ16GBに入る分だけソースを詰めるべき！
⇔NDW6全体の使用メモリ1.1GB/CPU (14km解像度で鉛直78層)
➔NICAM全部も入る！
まずは物理モジュールごとに個別GPU化から始めると作業分担ができて楽

GPUのボトルネック＝CPUとGPUの通信速度＋変数出力

CPUとGPUのやりとりが必要なのはintent(in, out, inout)変数だけ.
モジュール内の作業変数はGPU内でallocateするので、CPU-GPU通信が発生しない
➔より上位モジュールでGPU化すると、予報変数以外通信がいらなくなるのでもっと軽くなる
＊衝突計算のサブルーチンはintent変数が３次元70個、合計170MBの通信で0.7秒
＊NICAMの出力部分の計算コスト：1.3%

GPU高速化のまとめ

衝突成長は高度化の余地が大きいが計算コストも指数的に高くなる

GPUは**依存のないdo loopをGPUコア数で分業**することで高速化
今回は3500コアを用いて500倍の高速化が図れた

GPUに任せたいdo loopに指示行を入れて、
GPUが管理すべき変数を指定するだけ(OpenMPとほぼ同じ)

鉛直方向に依存があるモジュール（乱流、重力落下）は水平ループのみ高速化
（NICAMだとijmax=4356）

サンプルコードを見てみたい方はどうぞ
https://fbox.jamstec.go.jp/public/TVIdQx4K_r5tDh5Rn5AZhyVbF5SEyg7pXViWNwm-o8yA(mod_mp_ndw6.f90内のサブルーチンmixed_phase_collection_bin)