メタプログラミングによるキュムラント格子ボルツマン法の実装

Implementation of Cumulant Lattice Boltzmann Method through Metaprogramming

○ 高柳 哲也, パナソニック, 大阪府守口市八雲中町 3-1-1, takayanagi.tetsuya@jp.panasonic.com
松本 貴也, パナソニック, 大阪府守口市八雲中町 3-1-1, matsumoto.takaya@jp.panasonic.com
三木 慎一郎, パナソニック, 大阪府守口市八雲中町 3-1-1, miki.shinichiro@jp.panasonic.com
Tetsuya Takayanagi, Panasonic Co, 3-1-1 Yagumo-nakamachi, Moriguchi, Osaka
Takaya Matsumoto, Panasonic Co, 3-1-1 Yagumo-nakamachi, Moriguchi, Osaka
Shinichiro Miki, Panasonic Co, 3-1-1 Yagumo-nakamachi, Moriguchi, Osaka

Our project has developed the new computational fluid dynamics (CFD) based on Lattice Boltzmann Method(LBM). Cumulant LBM ; CLBM[M. Geier (2015)], which satisfies the Galilean invariance, has numerical stability at high Reynolds Number. The drawback of this model is complex about implementation of this model by hand. Here, we propose the novel algorithm, “mathematical programming”, which leads mathematical relation between moment and cumulant or central moment and cumulant algebraically and generates CUDA C kernel code by programming. We confirmed CLBM didn’t induce any numerical instability around extremely high Reynolds number ($O(10^7)$). We validate this model with flow around a bluff body.

1. 序

近年の計算機性能の著しい進化により流体解析において乱流の数値シミュレーションが重要な役割を占めつつある.

例えば, エアコンから発生する気流の Reynolds 数(Re 数)は 10 万以上, 人の歩行による誘引気流は Re 数 25 ～ 50 万程度である事を考えるといずれのケースでも流れは乱流である. 乱流における渦は物質・熱の輸送を促進する効果があるため, IAQ(Indoor Air Quality)や空調機器設計を考える上で, 乱流を正しく評価可能な気流解析 SIM の重要性は今後ますます高まっていくと考えられる.

通常, Re 数の高い乱流領域では高精度の解析を行う為に LES(Large Eddy Simulation)が用いられる. LES は非定常乱流を解析する上で, 計算負荷と計算精度のバランスに比較的優れたモデルである. ただ, 一般的な商用 CFD ソフトでは Navier-Stokes 方程式(NS 方程式)を直接解くアルゴリズムであるため, 圧力ポアソン方程式など並列化が難しい部分を含む場合が多い. この為, 大規模計算などで LES を使用した場合, 計算負荷を並列化でカバーするにはまだ課題がある. Computational Fluid Dynamics; CFD での乱流解析のうち, 最も計算負荷を抑えられるのが乱流をモデル化した RANS(Reynolds Averaged Navier-Stokes)であり, 一般に計算負荷は LES の 1/10 倍 ～ 1/100 倍であるためよく用いられる. 乱流の定常状態解析など平均的な気流を知る上ではコストパフォーマンスが非常に良いが, 非定常性の強い乱流領域は苦手とする場合が多い.

近年, シンプルなアルゴリズムで複雑な気流の解析ができ, 並列計算との親和性が高いことから格子ボルツマン法(LBM)が, 新たな CFD 手法として注目を集めている.

2. 格子ボルツマン法

LBM では Fig. 1のように空間及び速度方向を均一な格子点で離散化した Boltzmann 方程式を解く事で, 間接的に NS 方程式を解く CFD 手法である. Fig. 1ではベクトル $\mathbf{c_i}$ で表される27個の方向に速度が離散化されているため 空間次元と合わせて, D3Q27モデルという. この速度方向のインデックスを i とした時に, その方向に進む仮想流体粒子の割合を表す分布関数を$f_i(\boldsymbol{x},t)$で表す. この分布関数が LBM では基本となる物理量であり, その時間発展は衝突と並進という2ステップの格子ボルツマン方程式で記述される.

まず, 衝突では分布関数を構成する仮想粒子が文字通り衝突することによって, 場所ごとに定義される局所平衡状態に近づいていく. 衝突による分布関数の変化を Ω_i, 衝突後の分布関数 $f_i^*(\boldsymbol{x},t)$で表すと, 衝突は次式のように表せる.

$$f_i^*(\boldsymbol{x},t) = f_i(\boldsymbol{x},t) + \Omega_i$$

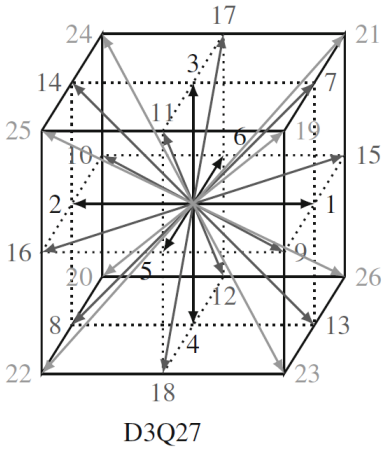

Fig. 1 LBM での速度モデル(D3Q27)

Ω_iには様々な衝突演算子が提案されており[1-7], この衝突演算子が系の数値安定性や計算精度に大きく寄与する. よく使われるのは次の BGK(Bhatnagar-Gross-Krook)近似である.

$$\Omega_i = -\frac{1}{\tau}(f_i(\boldsymbol{x},t) - f_i^{eq}(\rho,\boldsymbol{u}))$$

ここで, τは分布関数が局所平衡状態に緩和する緩和時間であり, BGK 近似は単一の緩和時間を用いるため, 単一緩和時間モデル(SRT; Single Relaxation Time)と呼ばれる. 局所平衡分布関数 $f_i^{eq}(\rho,\boldsymbol{u})$は次式のような離散化された Maxwell-Boltzmann 分布で

$$f_i^{eq}(\rho,\mathbf{u}) = w_i\rho\left(1 + \frac{3\mathbf{u}\cdot\mathbf{c_i}}{c^2} + \frac{9(\mathbf{u}\cdot\mathbf{c_i})^2}{2c^4} - \frac{3|\mathbf{u}^2|}{2c^2}\right)$$

ある.

次に衝突後の分布関数$f_i^*(\boldsymbol{x},t)$をそれぞれの速度方向 i へ1つ隣の格子点へ並進を行う.

$$f_i(\boldsymbol{x}+\boldsymbol{c_i}\Delta t, t+\Delta t) = f_i^*(\boldsymbol{x},t)$$

これ以外にも境界条件の適用などのステップはあるが,大まかに

はこの２つの単純なステップを繰り返すことで，流体の時間発展を計算することができる．この衝突と並進は完全に並列化可能であり，GPU(Graphics Processing Unit)などのアクセラレータと相性が良い．

3. キュムラント格子ボルツマン法

SRT は計算負荷が小さい為，よく用いられるが Re 数が少し高い領域では数値不安定性が生じる．そこで，乱流域(Re 数 $<O(10^3)$~$O(10^4)$)での計算の際には Humières ら[1]が提案した多緩和時間モデル(MRT; Multiple Relaxation Time)[1, 7]が用いられる事が多い．MRT は次式のように，分布関数をモーメント$m_{\alpha\beta\gamma}$に変換し，モーメントを衝突させることにより，SRT と比較して大幅な数値安定性を保証することに成功している．

$$\Omega_i = -\boldsymbol{M^{-1}SM}(f_i(\boldsymbol{x},t) - f_i^{eq}(\rho,\boldsymbol{u}))$$

ここで，$\mathbf{M}$，$\mathbf{M^{-1}}$は分布関数をモーメントへ変換/逆変換する行列である

しかし，MRT は Galilei 不変性を満たさないため，さらに Re 数の高い領域(Re 数 $>O(10^5)$)では数値不安定性を引き起こす．

この Galilei 不変性の保存性を満たす衝突モデルとして M. Geier は Cascade モデル[2]を提案した．Cascade モデルでは次式のように，モーメント$m_{\alpha\beta\gamma}$を Galilei 変換したセントラルモーメント$k_{\alpha\beta\gamma}$に変換してから衝突させることで，Galilei 不変性を保つ衝突モデル[3]である．

$$\Omega_i = -\boldsymbol{M^{-1}N^{-1}SNM}(f_i(\boldsymbol{x},t) - f_i^{eq}(\rho,\boldsymbol{u}))$$

ここで，$\mathbf{N}$，$\mathbf{N^{-1}}$はモーメントをセントラルモーメントに Galilei 変換/逆変換する行列である．このモデルは $\mathbf{N}$，$\mathbf{N^{-1}}$を単位行列とすると MRT に帰着する為，MRT を一般化したモデルであるといえる．この Galilei 不変性によって，MRT よりも更に数値安定性が上がることが M. Geier によって示された．

Cascade モデルは名前の通り，高次のセントラルモーメントが低次のセントラルモーメントに依存するというカスケード構造を取っている．この為，衝突時のセントラルモーメント間の独立性が高くないという問題がある．この問題点を解決する為に，高次のセントラルモーメントの平衡値を衝突後の低次のセントラルモーメントに因子分解を行う因子分解型カスケードモデル(Factorized Central moment LBM; FCLBM)[4]が提案されており，通常の Cascade モデルより更に数値安定性が高い．

そして，セントラルモーメント間の独立性が高い別の衝突モデルとして，キュムラント LBM(Cumulant LBM; CLBM)が M.Geier によって提案されている[5]．キュムラント$C_{\alpha\beta\gamma}$は次式のように定義され，分布関数を特徴付ける統計量である．

$$c_{\alpha\beta\gamma} = c^{-(\alpha+\beta+\gamma)}\frac{\partial^\alpha\partial^\beta\partial^\gamma}{\partial\Xi^\alpha\partial\Upsilon^\beta\partial Z^\gamma}\ln\left(F(\Xi,\Upsilon,Z)\right)\Big|_{\Xi=\Upsilon=Z=0}$$

ここで，$\boldsymbol{\Xi} = (\Xi,\Upsilon,Z)$は波数であり$F(\Xi,\Upsilon,Z)$は次式のように分布関数$f_i(\boldsymbol{x},t)$を両側ラプラス変換した物理量である．

$$F(\Xi,\Upsilon,Z) = \mathcal{L}[f_i(\boldsymbol{x},t)] = \int_{-\infty}^{\infty} f(\vec{\xi})e^{-\vec{\xi}\cdot\Xi}d\vec{\xi}$$

CLBM ではこのキュムラントを次式によって衝突させる．

$$c^*_{\alpha\beta\gamma} = c^{eq}_{\alpha\beta\gamma}\omega_{\alpha\beta\gamma} + (1-\omega_{\alpha\beta\gamma})c_{\alpha\beta\gamma}$$

ここで，$c^{eq}_{\alpha\beta\gamma}$はキュムラントの平衡値，$\omega_{\alpha\beta\gamma}$は緩和係数である．

CLBM では Galilei 不変性を満たし，またキュムラント間の統計的独立性が高い為，数値安定性が高い事が知られている．

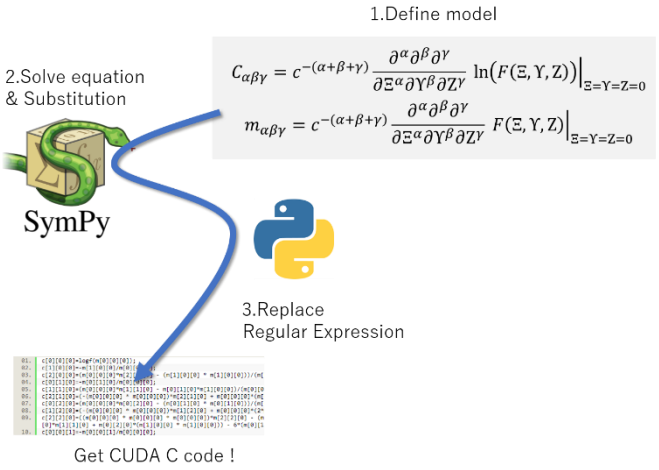

Fig. 2 メタプログラミングによるモデル実装の手続き

4. 本研究の目的

CLBM は乱流域における数値安定性が非常に高い事が知られているが，実際のモデル構築を行う際にモーメント，セントラルモーメント，キュムラント間の変換式導出とそのコーディングの難易度が高く，通常の手入力での実装を行うには煩雑な作業を必要とする．

そこで我々は CLBM のモデル構築に際して，コードの実装を手で行わず，ロジックによるプログラム生成，すなわちメタプログラミングにより CUDA C コードを自動生成する手法を新たに開発したので報告する．

CLBM を実装する際には，M. Geier らのオリジナル論文通り[5]モーメントをセントラルモーメントに変換し，セントラルモーメントからキュムラントに変換するモデル(以下，CLBM k2c)とモーメントからキュムラントに直接変換するモデル(以下，CLBM m2c)の 2 種類の実装が可能である．CLBM m2c ではモーメント⇔セントラルモーメント間の変換/逆変換が CLBM k2c より 1 回ずつ少ない為，実行速度が高速化する可能性がある．そこで，本研究では CLBM k2c と CLBM m2c の 2 つのモデルをメタプログラミングにより実装し，実行時間の比較を行った．

5. メタプログラミングによる CLBM m2c の実装

以下では，メタプログラミングを用いた CLBM m2c の実装方法について記す．Fig. 2 にメタプログラミングによるコード自動生成の手続きを示したが，以下で順を追って説明する．

1.モデルの定義

始めに変換を行いたいモデルの数式を定義する．CLBM m2c の場合はモーメントとキュムラントの定義式が必要である．このうち，モーメントは$F(\Xi,\Upsilon,Z)$と次式の関係がある．

$$m_{\alpha\beta\gamma} = c^{-(\alpha+\beta+\gamma)}\frac{\partial^\alpha\partial^\beta\partial^\gamma}{\partial\Xi^\alpha\partial\Upsilon^\beta\partial Z^\gamma}F(\Xi,\Upsilon,Z)|_{\Xi=\Upsilon=Z=0}$$

この為，$F(\Xi,\Upsilon,Z)$を通じてモーメント$m_{\alpha\beta\gamma}$とキュムラント$c_{\alpha\beta\gamma}$が関係している事が分かる．

2. Sympy による代数計算/連立方程式

先ほどの定義式からモーメントとキュムラント間の変換式を導くことができる．例えば，最も簡単な例として，c_{100}を定義に基づいて計算すると，

$$c_{100} = c^{-(1+0+0)}\frac{\partial^1\partial^0\partial^0}{\partial\Xi^1\partial\Upsilon^0\partial Z^0}\ln\left(F(\Xi,\Upsilon,Z)\right)\Big|_{\Xi=\Upsilon=Z=0}$$

$$= c^{-1}\frac{1}{F(0,0,0)}\frac{\partial}{\partial \Xi}F(\Xi,\Upsilon,\mathrm{Z})|_{\Xi=\Upsilon=\mathrm{Z}=0}$$

と計算できる.これに, モーメントの定義式と密度$\rho = F(0,0,0)$の関係式を用いると,

$$c_{100} = -\frac{1}{\rho}m_{100}$$

というように計算できる.

これを全ての次数のキュムラントについて行うことで, モーメントとキュムラントの関係式を導く事ができる. 実際にモーメント⇔キュムラントの変換式を導出する際には手計算するのではなく, Python の代数計算ライブラリである Sympy を用いて導く.

Sympy は変数や関数をシンボリックに定義し, 代数的に微分, 積分, 級数展開の計算を行ったり, 連立方程式を解いたりすることができる Python のライブラリである.

この Sympy を用いて, 関数 F を定義し, 先の定義に基づいてキュムラント$c_{\alpha\beta\gamma}$と F との関係式を計算する. モーメント$m_{\alpha\beta\gamma}$でも同様に, 定義に基づいて関数 F との関係式を計算する.

これらが計算できたらキュムラントの関係式にある F の偏微分項を全てモーメントで置き換えていく.この時には最高次のモーメントから最低次のモーメントの順で置き換えていく.この手続きにより,代数的にキュムラントのモーメント表現 (以下, cum_by_mo)を得ることができる.

次に導いた cum_by_mo を今度は, モーメントについての方程式と見なし, 連立方程式を解く.これにより, キュムラントからモーメントへの逆変換の表現(以下, mo_by_cum)を導く.これらの手続きの際には, 式を簡単化する為に,$C_{\alpha\beta\gamma} = \rho c_{\alpha\beta\gamma}$を定義しておく.

これらの手続きにより, 例えば, m_{210}, C_{210}の変換/逆変換は次式のように得られる.

$$C_{210} = -m_{210} + \frac{m_{010}m_{200}}{m_{000}} + \frac{2m_{100}m_{110}}{m_{000}} - \frac{2m_{100}m_{100}^2}{m_{000}^2}$$

$$m_{210} = -(C_{010}C_{100}^2 + C_{010}C_{200} + 2C_{100}C_{110} + C_{210})e^{C_{000}}$$

このように Sympy を利用することで,モーメント, キュムラント間の変換式導出を自分で手計算する必要性が無くなる.なお, 分布関数からモーメントへ変換する行列と逆行列の導出も同様の手続きを踏めば容易に導出できる.

3. Python でのコード自動生成

これらすべての変換式/逆変換式が得られたら, それらを正規表現や置換を用いて所望の形の CUDA C コードに変換する. Sympy での代数式は Python 内でシームレスに渡すことができる為, 容易に CUDA C コードを生成する事ができる. 実際に生成されたコードの抜粋を Fig. 3 に示した.

なお今回は, コード実装の容易化という点にのみ着目し, 高速化という観点からは一切のチューニングをしていない. この為, このコード自動生成時にチューニングを行えば, 本研究よりも高速化が見込めると考えられる.

このようにメタプログラミングを用いることで, モデルの複雑さに関係無く, コード生成を行える. メタプログラミングは, 物理量の数学的定義式のみに基づいている為, コード生成の過程で入力ミスが生じることは無い. この為, 生成されたバグ取りの必要が無いという事は大きなメリットと言える. 我々は, 今回構築した数学的な定義式に基づいたメタプログラミング技術を"数学的プログラミング"と名付けた.

この数学的プログラミングでは, 全ての CUDA C コードを自動生成するのではなく,モデル構築において最も煩雑であるモーメント⇔キュムラント間の変換部分について適用する.

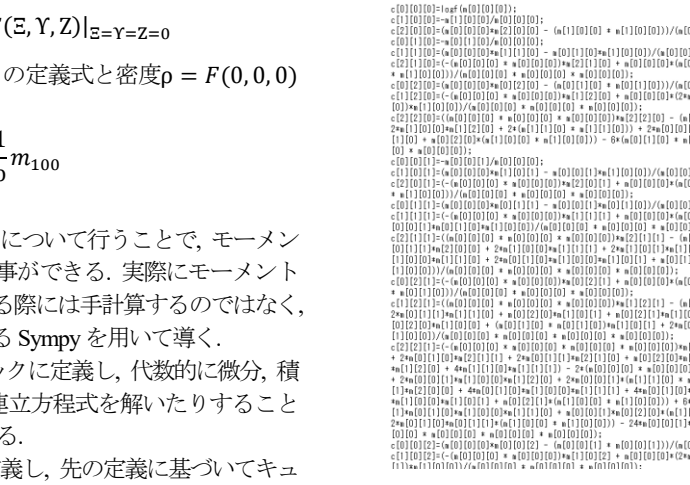

Fig. 3 メタプログラミングによる CUDA C のコード生成抜粋 (モーメントからキュムラントへの変換式)

この為, モーメント⇔キュムラント間の変換/逆変換部分以外の衝突部分などは従来通りプログラミングを行う.この衝突部分や CUDA C カーネル全体のコード生成も自動化することはできるが, 衝突部分などのプログラミングは容易な為,自動化するメリットは小さい.キュムラントの衝突は陽に示すと, 次式のように行われる.

$$C_{110}^* = (1 - \omega_1)C_{110}$$
$$C_{101}^* = (1 - \omega_1)C_{101}$$
$$C_{011}^* = (1 - \omega_1)C_{011}$$

$$C_{200}^* - C_{020}^* = (1 - \omega_1)(C_{200} - C_{020}) - 3\rho(1 - \frac{\omega_1}{2})(u^2D_xu - v^2D_yv)$$

$$C_{200}^* - C_{002}^* = (1 - \omega_1)(C_{200} - C_{002}) - 3\rho(1 - \frac{\omega_1}{2})(u^2D_xu - w^2D_zw)$$

$$C_{200}^* + C_{020}^* + C_{002}^* = \rho\omega_2 + (1 - \omega_2)(C_{200}^* + C_{020}^* + C_{002}^*) - 3\rho(1 - \frac{\omega_2}{2})(u^2D_xu + v^2D_yv + w^2D_zw)$$

ただし,

$$D_xu = -\frac{\omega_1}{2\rho}(2C_{200} - C_{020} - C_{002}) - \frac{\omega_2}{2\rho}(C_{200} + C_{020} + C_{002} - \rho)$$
$$D_yv = D_xu + \frac{3\omega_1}{2\rho}(C_{200} - C_{020})$$
$$D_zw = D_xu + \frac{3\omega_1}{2\rho}(C_{200} - C_{002})$$

である. 残りのキュムラントの衝突は以下のように実行する.

$$C_{120}^* + C_{102}^* = (1 - \omega_3)(C_{120} + C_{102})$$
$$C_{210}^* + C_{012}^* = (1 - \omega_3)(C_{210} + C_{012})$$
$$C_{201}^* + C_{021}^* = (1 - \omega_3)(C_{210} + C_{021})$$
$$C_{120}^* - C_{102}^* = (1 - \omega_4)(C_{120} - C_{102})$$
$$C_{210}^* - C_{012}^* = (1 - \omega_4)(C_{210} - C_{012})$$
$$C_{201}^* - C_{021}^* = (1 - \omega_4)(C_{210} - C_{021})$$

$$C_{111}^* = (1 - \omega_5)C_{111}$$

$$C_{220}^* - 2C_{202}^* + C_{022}^* = (1 - \omega_6)(C_{220} - 2C_{202} + C_{022})$$
$$C_{220}^* + C_{202}^* - 2C_{022}^* = (1 - \omega_6)(C_{220} + C_{202} - 2C_{022})$$
$$C_{220}^* + C_{202}^* + C_{022}^* = (1 - \omega_7)(C_{220} + C_{202} + C_{022})$$

$$C^*_{211} = (1 - \omega_8)C_{211}$$
$$C^*_{121} = (1 - \omega_8)C_{121}$$
$$C^*_{112} = (1 - \omega_8)C_{112}$$
$$C^*_{221} = (1 - \omega_9)C_{221}$$
$$C^*_{212} = (1 - \omega_9)C_{212}$$
$$C^*_{122} = (1 - \omega_9)C_{122}$$
$$C^*_{222} = (1 - \omega_{10})C_{222}$$

緩和係数のうち, ω_1が流体の粘性と関係し,

$$\omega_1 = \left(\frac{1}{2} + \frac{3\nu}{c^2\Delta t}\right)^{-1}$$

となる. 本研究では, ω_1以外の緩和係数は全て 1.0 を代入している.

6. メタプログラミングによる CLBM k2c の実装

同様の手続きを踏むことで M. Geier らのオリジナルの論文[5]で提案しているセントラルモーメントからキュムラントへの変換式を導くこともできる.セントラルモーメントは次式で定義される.

$$k_{\alpha\beta\gamma} = c^{-(\alpha+\beta+\gamma)} \frac{\partial^\alpha \partial^\beta \partial^\gamma}{\partial \Xi^\alpha \partial \Upsilon^\beta \partial Z^\gamma} \tilde{F}(\Xi, \Upsilon, Z)\Big|_{\Xi=\Upsilon=Z=0}$$

$$\tilde{F}(\Xi, \Upsilon, Z) = e^{-u\Xi - v\Upsilon - \omega Z} F(\Xi, \Upsilon, Z)$$

CLBM k2c ではまずモーメント⇔セントラルモーメント間の変換式から求める. 変換式はモーメント, セントラルモーメントの定義式から F を通じて計算し, セントラルモーメントのモーメント表現を得る(k_by_mo). この k_by_mo をモーメントについて連立方程式を解けば, 逆変換の表現を得ることができる(mo_by_k).

次に, CLBM m2c と同じ手続きを踏み, モーメントとキュムラント間の関係式(mo_by_cum)を求める.

最後に, k_by_mo の全てのモーメント項について, mo_by_cum を用いてキュムラントに置き換えていく事で,セントラルモーメントのキュムラント表現(k_by_cum)を得る.この関係式をキュムラントについて, 連立方程式を解けば, 逆変換の表現(cum_by_k)を得ることができる.

全ての表現が得られたらこれを CLBM m2c と同様に正規表現や置換を用いて, CUDA C コードの生成を行う.これら手続きにより, CLBM k2c も容易に実装できる.

6. CLBM の数値安定性検証

構築した CLBM の数値安定性を評価する為にキャビティフローで Re 数 10^3~10^7 の間で検証を行った. 格子解像度については 100 x 100 x 100 で固定し, 収束判定については 50000[step]以上発散せずに解けるかで判定している.

プログラム全体は Python で書かれており, メインの繰り返し計算部分を PyCUDA[7]で GPU 並列化し計算を行っている. GPU は Quadro RTX 8000 x 1 基を用いて計算を行っており,計算精度は単精度で計算を行った.

比較の為に D3Q19-MRT-LBM と D3Q27-FCLBM についても計算を行っている.データ出力には h5py から HDF フォーマットで出力している.ポスト処理時には, pyevtk を用いて VTK(Visualization Toolkit)フォーマットへ変換したものを Paraview で可視化している.

検証結果を Table 1 に示す. まず D3Q19-MRT-LBM では Re 数が 5000 まで安定に計算でき, Re 数 10000 では計算が発散した. 一方で, D3Q27-FCLBM および D3Q27-CLBM では, 今回検証した全ての Re 数で数値安定に解く事が出来た. MRT とこれら衝突モデルとの差異は Galilei 不変性であり, 乱流域では衝突時の Galilei 不変性を保つ事が本質的に重要であることを意味している.

Table 1 キャビティフロー(100 x 100 x 100)での衝突モデルごとの数値安定性

Reynolds数	D3Q19	D3Q27	
	MRT-LBM	FCLBM	CLBM
10^3	○	○	○
$5x10^3$	○	○	○
10^4	×	○	○
10^5	×	○	○
10^6	×	○	○
10^7	×	○	○

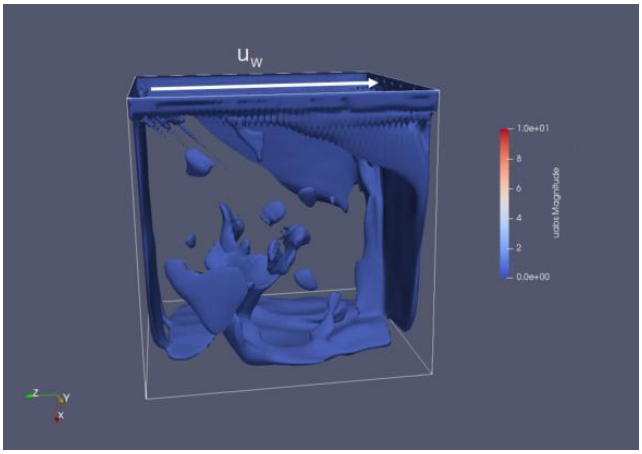

Fig. 4 Re 数 10^6 におけるキャビティフローの速度コンター(0.5[m/s]領域を抽出)

Re 数 10^6 における CLBM の流速分布コンター図(0.5[m/s]領域)のスナップショットを Fig. 4 に示す. 100 x 100 x 100 という限られた格子解像度の為, 移動壁近傍は離散化誤差の影響を受けている事が分かる.

7. CLBM の妥当性検証

次に CLBM の妥当性検証の為に, 円柱周りの流れの計算を行った. 比較検証用の実験データ取得の為に, 実験系の構築を行った. 構築した実験系のシステム図を Fig. 5 に示す.

実験系は, 幅 15[cm] , 高さ 15[cm]の流路の中心にΦ20 の円柱が設置されている系であり, 前段には整流の為に穴径の異なる整流板が多段で設置されている. 流速分布の算出には PIV(Particle Image Velocimetry)を用いた. 微粒子をエアロゾルジェネレータにより導入し, シートレーザー(532[nm])で可視化した. 側面から高速度カメラ(1000[FPS])で撮影している.

実験条件について,バルクの Re 数は 2976 であり,シミュレーションと比較を行う為に,円柱後方流れを数秒間の時間平均を行っ

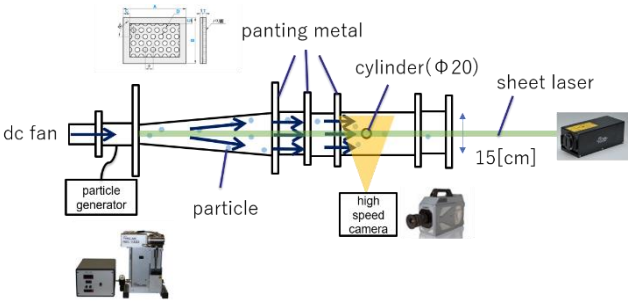

Fig. 5 円柱周りの流れ解析実験系

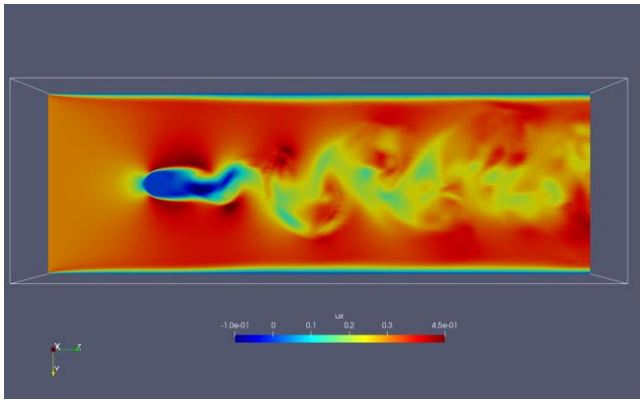

Fig. 6 CLBM での円柱周りの流れ u_x (Re 数 2976) のスナップショット

た. また実験は N 数 5 で行っている.

LBM でのシミュレーションは本実験系と同じ系で計算を行った.格子解像度は dx=0.75[mm]であり,格子点数は 200 x 200 x 600 である.シミュレーション時には実験と同じくカルマン渦の生成が定常状態となったステップから時間平均を行っている.

Fig. 6 に CLBM で計算した際の u_x の流速分布のスナップショットを示す.カルマン渦の形成と円柱後方の淀み領域が確認できる. Fig. 7 に円柱後方の淀み点から流れ方向への時間平均された流速分布を示す. ここで, u_x は流速の x 成分[m/s], U_0 は流入速度[m/s], x は円柱の中心からの距離[cm], D は円柱の直径[cm]である.また実験は N 数 5 の最大・最小幅をプロットしている. Fig. 7 を見てわかる通り, 実験結果の流速のディップ位置や曲線概形を概ね再現しており, 構築したモデルは妥当であると判断した.

8. 実行時間の比較 : CLBM m2c vs CLBM k2c

最後に, M. Geier らのオリジナルのモデル(CLBM k2c)とモーメントを直接,キュムラントに変換するモデル(CLBM m2c)において実行速度の比較を行った. 比較に際しては, 300 x 300 x 300 格子でのキャビティフローの計算を行い, D3Q19-MRTLBM, D3Q27-SRT-LBM, D3Q27-MRT-LBM, D3Q27-FCLBM, D3Q27-CLBM のモデルにおいて,1[step]あたりの衝突・並進部分の CUDA C カーネル実行時間[ms]を計測した. Table 2 に比較結果を示す. 実行時間については複数ステップで計測を行い, その平均値を算出している.

ここでは D3Q27-SRT の実行時間をリファレンスとして,他の衝突モデルと比較する.D3Q27-MRT では約 33%の計算負荷が増えているが, 一般的には 30%程度といわれており, 想定通りの結果である. FCLBM については,約 46%程度の計算負荷増加となった.今回検証した範囲では, CLBM と同じ数値安定性を示している事を考えると, FCLBM は計算負荷と数値安定性のバランスが良く, 実用的なモデルであると言える.

CLBM については, 概ね 80 ~ 90%程度の計算負荷増加となったが, 2 つのモデルで計算負荷が異なる結果となった.

CLBM m2c ではオリジナルの論文と比較して,変換・逆変換の演算回数が 1 回ずつ少ない為,実行速度向上が期待されたが,実際には CLBM k2c の方が計算負荷は小さいという結果となった.

これは, CLBM m2c ではモーメントをキュムラントに直接変換する際の変換式が, セントラルモーメントからキュムラントに変換するよりも複雑である事を示している. 今回の結果からは, 1 度ずつ変換式が増えてもセントラルモーメントを経由してキュムラントへ変換するモデルの方がトータルの計算コストとしては抑えられるという事を意味している.

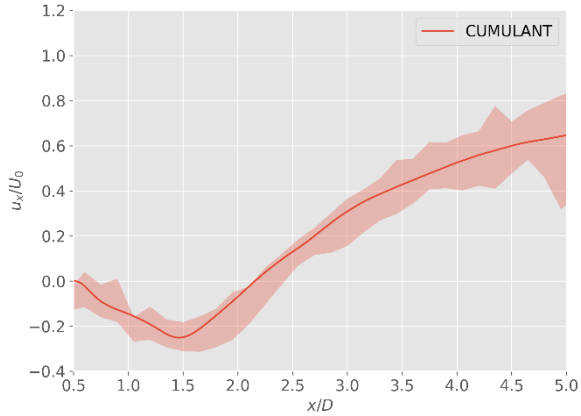

Fig. 7 円柱後方の淀み領域の u_x の流速分布
実線 : CLBM, 塗りつぶし: 実験値

Table 2 各衝突モデルの実行時間の比較
Execution Time: 衝突・並進の CUDA C 実行時間 単位 [ms]
Ratio: D3Q27-SRT-LBM をリファレンスとした計算時間の比

	D3Q19	D3Q27				
Collision Model	MRT	SRT	MRT	FCLBM	Cumulant m2c	Cumulant k2c
Execution Time	49.58	68.63	91.40	100.22	132.45	123.72
Ratio	0.72	1	1.33	1.46	1.93	1.80

9. 結論

本研究では M.Geier らの提案した CLBM について, 実装時に必要となる各種の変換及び逆変換の表式導出とコーディングの複雑性というハードルを取り除くために, メタプログラミングによるモデル実装を行った.具体的には, モーメントやセントラルモーメント, キュムラントの数学的な定義式のみから, Sympy を用いて代数的に変換式を導出し,それらを Python 内で正規表現や置換を用いて,CUDA C カーネルを自動生成するという新たなメタプログラミング手法(”数学的プログラミング”)の提案を行った.

この数学的プログラミングを用いて, CLBM の構築を行い, 3 次元キャビティフローで Re 数 $10^3 \sim 10^7$ まで数値安定性の検証を行い, 全ての Re 数で数値安定に計算できることを明らかにした.

また,モデルの妥当性検証の為, 円柱周りの流れについて, 実験(PIV)および CLBM を比較する事で検証を行った.円柱後方の淀み領域の主流方向の流速分布の比較を行い, CLBM は実験結果を再現している事が検証できた.

また, M. Geier がオリジナルの論文で提案しているモーメント⇒セントラルモーメント⇒キュムラントへ順に変換をするモデル(CLBM k2c)とモーメントからキュムラントへ直接変換するモデル(CLBM m2c)の 2 つのモデルの実装を行い, 実行時間の比較を行った. その結果, 実行時間としては変換回数が 1 回ずつ増えるものの, CLBM k2c の方が CLBM m2c よりも速く, 計算負荷の観点からはオリジナルの論文通りに実装した方が良い事が明らかとなった.

10. 今後

現在, 当部署では開発を進めている LBM を IAQ 分野へ応用する為に, FCLBM や CLBM と微粒子の強連成が可能なシミュレーションプラットフォームの構築を行っている. このプラットフォームにより, 室内空間における PM2.5 やウイルスといった微粒子の過渡的な拡散現象の解析を目指している.

例えば, Fig. 8 は FCLBM と微粒子を連成し, Four-way モデルで 6

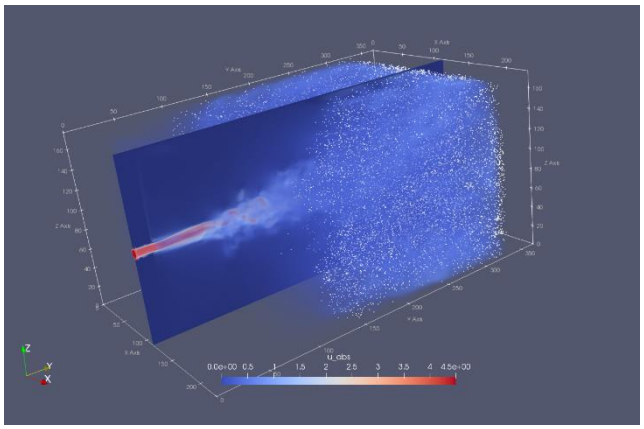

Fig. 8 FCLBM-微粒子連成モデルでの室内空間の微粒子拡散の解析例 白：微粒子(粒径 5[μm])

畳程度の空間内の微粒子の拡散を解析した例であり，従来の CFD でよく用いられる Reynolds Averaged Navier-Stokes；RANS とは大きく結果が異なっている事が明らかになりつつある．今後は，本プラットフォームを拡張し，熱流体との連成を予定しており，社内向け汎用 CFD ツールとして開発を継続する．

参考文献

(1) D. d'Humières et al., "Multiple-relaxation-time lattice Boltzmann models in three dimensions", Phil. Trans. R. Soc. A 360, (2002), 437-451
(2) M. Geier, A. Greiner, J.G. Korvink, "Cascaded digital lattice Boltzmann atomata for high Reynolds number flow", Phys. Rev. E 73, 066705 (2006).
(3) L. Fei, K.H. Luo, Q. Li, ″Three-dimensional cascaded lattice Boltzmann method: Improved implementation and consistent forcing scheme″, Phys. Rev. E. 97, (2018)053309
(4) M. Geier, A. Greiner, J.G. Korvink, "A factorized central moment lattice Boltzmann method", Eur. Phys. J. Spec. Top. 171 (1) 55-61 (2009)
(5) M. Geier, M. Schönherr, A. Pasquali, M. Krafczyk, "The cumulant lattice Boltzmann equation in three dimensions: theory and validation", Comp. Math. Appl. 70 (4) (2015) 507-547.
(6) Y. P. Sitompul, T. Aoki ″A filtered cumulant lattice Boltzmann method for violent two-phase flow″, J. Comp. Phys. 390 (2019) 93-120
(7) K. Suga, Y. Kuwata, K. Takashima, R. Chikasue, "A D3Q27 multiple-relaxation-time lattice Boltzmann method for turbulent flows", Comp. Math. Appl. 69 (6) (2015) 518-529.
(8) A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, A. Fasih, "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation", Parallel Comp. 38(3), (2012), 157-174.