高性能計算 CAE における多相流シミュレーション

池端昭夫 [1†] 黒崎裕太 [1] 佐々木一真 [1]

[1]TOTO 株式会社

Multiphase fluid simulation method in high performance CAE

Akio IKEBATA[*†] Yuta KUROSAKI[*] Kazuma SASAKI[*]

[*]TOTO LTD.

Abstract

A CAE framework which realizes mass-scale multiphase fluid simulation in practical commercial products by means of finite volume method on unstructured grids has been opened. In order to utilize the multiphase fluid simulations in real-case applications, calculation robustness and efficient high performance computing must be satisfied in the framework. TVD on finite volume method has been applied for the purpose of stabilizing advection calculation on polyhedral unstructured grids. Multi-process and multi-threaded calculations on GPU server and supercomputer “Fugaku” has been conducted using Non-Conformal boundary Domain Decomposition Method on unstructured grids. This framework has been applied to multiphase fluid simulations of the shower, which is able to evaluate various qualities of spouts.

Keywords: Multiphase flow, Fugaku, HPC, TVD, PLIC, Unstructured grid, Domain decomposition method

1. はじめに

CAE における流体解析では，構造解析等と同様に市販ソフトウェアが企業で広く用いられている．著者らは衛生陶器をはじめとする水まわり製品の開発・販売を行っており，空気と水が混在する多相流のシミュレーションが設計の多くの場面で実施されている．多相流は，空気のみもしくは水のみのシミュレーションと異なり，複雑かつ時系列に大きく変動する気液界面を正しく捉えることが大きな課題であり，市販ソフトウェアを用いたシミュレーションでは，計算精度や計算速度の面で不十分な状況が出現した．

この課題への対処を出発点として，詳細が公開されている一般計算手法のコード習作による精度検証や改良の試み，さらにはスパコン利用によるハイパフォーマンスコンピューティング（以下 HPC）への適用による計算時間短縮効果検証を行ってきた．その結果，市販ソフトウェアに頼らずとも開発・保守の社内リソースを最小限に抑えて一般計算手法に基づく多相流 CAE が構築可能であることを本稿で示す．この手法は「学術雑誌などでアルゴリズムが詳細に公開されており，すべて自分でコード構築が可能な計算手法」の組み合わせに基づく CAE 環境整備の考え方の提案であり，OpenFOAM などのオープンソースコードを直接利用する方法とはまた別の「オープン CAE」であると考える．オープンソースの利用は近年拡大しているが，本課題の「多相流＋HPC」という明確な目標において，様々な機能を持つ巨大なオープンソースの全体構造を理解して各部作り込んでいくのは容易ではないと考えられる．むしろ，オープンな計算手法に基づいて小さなコード作成，例えば移流計算プログラムから作り始めて徐々に機能を追加していく方が，最初の取り組みの敷居を下げることが可能と考える．

オープンな計算手法を利用するにあたって，企業での CAE 利用における大きな要請点はソルバーの「計算安定性」である．業務を計画的に進めるためには計算時間の見積もり，計算完了の日時の予測が必須であり，いくら高精度でも計算が不安定でいつ計算が破綻するか予測不可能な手法を用いるのは不適である．とくに水と空気の二相流体解析では，密度比が約 1,000 倍もあり，毎回安定的に計算を実行することは容易ではない．そこで精度と安定性のバランスが良く広く用いられている TVD 法をベースとした手法を利用する．

次に，複雑構造を持つ様々な製品開発に適用するため，流路形状に追随した計算メッシュを利用できる「非構造格子法」に基づく計算手法を利用する．汎用性が高い「多面体非構造格子」の TVD 法は詳細が公開されており，簡単なコード作成で移流計算テストから始めることが可能である．本稿では第二章で詳細な計算手法を示す．また気液界面追跡の方法として，非構造格子における PLIC 法の実装方法を明らかにする．第三章

†*akio.ikebata@jp.toto.com*

では，これらの手法の妥当性についてベンチマーク計算により検証する．

HPC の主要技術としては，古くはベクトル計算，その後スカラー並列計算に主力が変わり，近年では GPU 並列サーバー利用が大きな流れとなっている．まずスケーラビリティが課題となる並列計算手法については，メモリ分散並列を複雑計算モデルに汎用的に適用可能な「非境界整合領域分割法」について第四章で述べる．次にプロセス単位の実行最適化の観点より，GPU サーバーおよびスパコン富岳でのコーディング方法について示す．第五章ではこれらの手法を用いることでプロセッサ演算性能の向上と良好な並列化効率が得られ，1 億メッシュを超える大規模な多相流体シミュレーションが富岳上で実用的な時間で実施できるようになったことを，シャワー吐水シミュレーションを通して示す．

2. 非構造格子多相流シミュレーション計算手法の概要

任意多面体格子での基本的な非定常流体計算手法を示す．各有限体積格子における三次元流速ベクトルを$\boldsymbol{u}=(u,v,w)$，密度をρ，圧力をp，粘性係数をμ，体積力に置き換えた表面張力を$\boldsymbol{S}$，重力加速度ベクトルを$\boldsymbol{g}$とおくと，ナビエストークス流体方程式より，非定常流体解析において各項を分割し順番に陽的に計算するフラクショナルステップ法では，1 タイムステップにおいて次の 3 式を順番に解く．

$$\frac{\boldsymbol{u}^{*}-\boldsymbol{u}^{n}}{\Delta t}=-\boldsymbol{u}^{n}\cdot\nabla\boldsymbol{u}^{n} \tag{1}$$

$$\frac{\boldsymbol{u}^{**}-\boldsymbol{u}^{*}}{\Delta t}=\frac{1}{\rho}\nabla\cdot(\mu\nabla\boldsymbol{u}^{*})+\frac{1}{\rho}\boldsymbol{S}+\boldsymbol{g} \tag{2}$$

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^{**}}{\Delta t}=-\frac{1}{\rho}\nabla p^{n+1} \tag{3}$$

2.1. 移流項の計算

式(1)～式(3)で計算精度と安定性の観点で最も問題となるのは式(1)の移流項である．求める物理量$\phi\equiv(u,v,w)$として式(1)を以下のように変形する．

$$\frac{\phi^{*}-\phi^{n}}{\Delta t}=-\nabla\cdot(\phi^{n}\boldsymbol{u}^{n})+\phi^{n}\nabla\cdot\boldsymbol{u}^{n} \tag{4}$$

右辺第一項は，図 1 に示す各面を通して法線方向から格子に入ってくる（あるいは出ていく）物理量の総和を示し，右辺第二項は，その間に流れによって体積変化が起こり格子内の物理量が変化することを表している．一般に非圧縮性の性質を示す低マッハ数の緩やかな流れにおいては，右辺第二項の$\nabla\cdot\boldsymbol{u}$が示す体積変化はゼロに近いとおくことができるが，削除しなくても良い．右辺第一項では面を通した物理量の移動により両側隣接格子の物理量変化が決まるため保存型の定式化である．本手法は有限体積法とよばれる．この移動量を一般的にフラックスとよび，フラックスの定式化が流体計算の精度と安定性を左右する．式(4)では面fのフラックスは$\phi_f\boldsymbol{u}_f$の面法線方向成分であり，$\phi_f\boldsymbol{u}_f$と面法線方向ベクトル$\boldsymbol{n}_f$との内積で計算される．面値ϕ_fの計算方法について，隣接格子の二つの格子の物理量ϕを足して 2 で割る中心差分法では，状況により計算が不安定になることがある．隣接格子およびさらにその周囲の格子の値も利用して高次精度空間補間し，さらに計算安定化のために人工粘性係数を入れた微分係数項を加える方法も考えられるが，人工粘性係数の問題ごとの適切な設定は容易ではない．面値ϕ_fの安定的な計算手法としては TVD 法[1]がある．図 1 の多面体格子に TVD 法を適用する場合，面の法線方向を仮想的な一次元軸を設定し，下流側の隣接格子ϕ_D，上流側隣接格子ϕ_C，さらにその上流の値ϕ_Uの計 3 つのϕを用いて計算する．

$$\phi_f=\phi_C+\frac{1}{2}\psi(r_f)(\phi_D-\phi_C),\quad s.t.\ r_f\equiv\frac{\phi_C-\phi_U}{\phi_D-\phi_C} \tag{5}$$

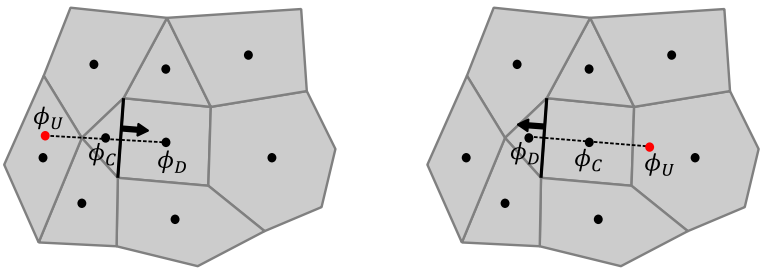

Fig. 1 TVD method for finite volume method on polyhedral unstructured grids

関数$\psi(r_f)$は様々な式が提案されているが，計算安定性と精度の観点では Van Leer の式が推奨される．

$$\psi(r_f) \equiv \frac{r_f + |r_f|}{1 + |r_f|} \tag{6}$$

ちなみに$\psi(r_f) \equiv 0$一定の場合は一次風上差分に帰着し，$\psi(r_f) \equiv 1$は中心差分，$\psi(r_f) \equiv 2$は風下差分に相当する．式(6)の Van Leer 式は$\psi(r_f)$が 0 から 2 に滑らかに変化する性質が，計算安定性および精度上好ましい．

非構造格子 TVD 法のもう一つの論点としては，仮想点Uにおけるϕ_Uの空間補間方法である．いま非構造格子Cの空間勾配$\nabla\phi_C$が既知とすると，

$$\phi_U = \phi_D + 2\nabla\phi_C \cdot (\boldsymbol{x}_C - \boldsymbol{x}_D) \tag{7}$$

により計算できる．この定式化では等間隔直交格子で$\nabla\phi_C$を中心差分で展開した場合，式(7)は$\boldsymbol{x}_U$におけるϕ_Uに一致するため妥当である．非構造格子の場合，$\nabla\phi_C$は周囲の隣接格子を用いた最小二乗法で計算できる．

Appendix の Code1 および Code2 に，式(4)の有限体積法および式(5)の TVD 法のコーディング例を示す．有限体積法では，体積変化項は外し，非圧縮性仮定としている．有限体積法計算の前に行う各面の移流フラックス計算では，TVD 法で求めた面値ϕ_fと流速$\boldsymbol{u}$の面法線方向成分との積を保存しておく．

なお筆者らは TVD 法の精度をさらに改善する方法として，面の法線方向の一次元軸に保存型 CIP 法の二次補間関数を用いる VSIAM3[2]について公開しており，計算精度は TVD 法の同等以上の結果を得ている[2].

2.2. 非移流項の計算

式(2)の粘性項については, フラックスの値が$\mu\nabla\boldsymbol{\phi}$の面法線方向成分となり，移流項同様に有限体積法の定式化で計算できる．表面張力項の$\boldsymbol{S}$の計算については，一般的な CSF モデル[3]を使用する．CSF モデルにおける中心差分式の部分を，最小二乗法による勾配計算に置き換えればよい．また式(3)については，体積変化と圧力変化との関係式 $\partial p/\partial t = -\rho {C_s}^2 \nabla \cdot \boldsymbol{u}$ に式(3)を代入して圧力pに関するポアソン方程式が得られる．

$$\nabla \cdot \left(\frac{\Delta t}{\rho}\nabla p^{n+1}\right) = \frac{p^{n+1} - p^*}{\rho {C_s}^2 \Delta t} + \nabla \cdot \boldsymbol{u}^{**} \tag{8}$$

この方法は CIP-CUP 法とよばれる[4]．非圧縮性に近い緩やかな流れにおいては，タイムステップΔtの間の圧力の波動を捉えることは非現実的であるため，圧力に対しては陽解法ではなく陰解法で計算されなければならない．$\boldsymbol{u}^{**}$, p^*は式(8)のポアソン方程式を解く直前の値であり，p^*は式(1)と同様の方法で移流計算される．式(8)の右辺第一項はp^{n+1}に関する係数が含まれており，圧力方程式行列の対角項に足し合わされるため，行列は優対角化され収束性が向上する．

連立方程式解法は企業の製品設計利用におけるロバスト性を考慮し，PCG 法[5]を推奨する．PCG 法は，他の方法での問題ごとのパラメータ調整（緩和係数等）が必要ないところが利点となる．PCG 法のアルゴリズムを図 2 に示す．前処理行列$M^{-1} \equiv \{m_{ij}\}$をどのようにするかが一つの論点であるが，簡便で並列計算に適用しやすい$\{m_{ij}\} \equiv \{1/a_{ii}\}$を用いるのが実用的である．$a_{ii}$は$\boldsymbol{A}$の対角項であり，$\{m_{ij}\}$の非対角項は全てゼロである．もちろん$M^{-1}$にはマルチグリッド法[5]を用いることも可能である．

Setting initial value for $\boldsymbol{x}^0$
Residual $\boldsymbol{r}^0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^0$;
$\gamma = \boldsymbol{b}^T\boldsymbol{b}$;
for (l = 0; l < *Max_iteration*; l++) {
  $\varepsilon = \boldsymbol{r}^{l^T}\boldsymbol{r}^l$;
  if (ε/γ < *Tolerance*) break;
  $\boldsymbol{z}^l = \boldsymbol{M}^{-1}\boldsymbol{r}^l$;
  $\rho^l = \boldsymbol{r}^{l^T}\boldsymbol{z}^l$;
  $\beta = (l == 0)\ ?\ 0 : \rho^l/\rho^{l-1}$;
  $\boldsymbol{p}^l = \boldsymbol{z}^l + \beta\boldsymbol{p}^{l-1}$;
  $\boldsymbol{q}^l = \mathbf{A}\boldsymbol{p}^l$;
  $\alpha = \rho^l/\left(\boldsymbol{p}^{l^T}\boldsymbol{q}^l\right)$;
  $\boldsymbol{x}^{l+1} = \boldsymbol{x}^l + \alpha\boldsymbol{p}^l$;
  $\boldsymbol{r}^{l+1} = \boldsymbol{r}^l - \alpha\boldsymbol{q}^l$;
}

Fig. 2 Preconditioned Conjugate Gradient method for Poisson equation’s solver

2.3. 密度の計算

多相流シミュレーションにおける単相流とは異なる課題として，気液の密度差の大きさが挙げられる．水と空気の場合 1000 倍近い差があり，水と空気の界面を正確に捉えつつ計算の安定性を確保する必要がある．非構造格子のオイラー法では，気液界面のメッシュを動かして追跡していくラグランジェ法とは異なり，固定メッシュで各メッシュの流体率（液体率）により陰的に気液界面を表す VOF 法が一般的な方法である．その中でも，図 3 のように各格子体積内では気液界面は区分的に平面とみなし，流体率と法線ベクトルから平面方程式を算出して流量フラックスを計算する PLIC（Piecewise Linear Interface Calculation）型の VOF 法（PLIC-VOF 法）が実用的な計算手法である．

再構築する平面の単位法線ベクトルを$\boldsymbol{n} = (n_x, n_y, n_z)$，切片を$q$とおくと，平面方程式は$n_x x + n_y y + n_z z + q = 0$で表される．$\boldsymbol{n}$は周囲の流体率$F$を用いて最小二乗法で求めることができるので既知とすると，自格子の流体率F_iに適合するようにqを求める問題に帰着する．Rider らの方法[6]は，自格子の体積をV_i，平面方程式で切り取られる体積をV_rとおくと，方程式

$$f(q) = V_r/V_i - F_i = 0 \tag{9}$$

が成り立つようにqを求める．V_rを陽的に数式展開することは非構造格子では困難なため，qを少しずつ変えながら反復計算により収束させていく．Rider ら[6]は Brent 法[7]を用いることにより比較的少ない反復数で収束させることができることを示している．Brent 法は多くのコード例が公開されている．

V_rの計算は任意多面体の体積を求める幾何計算であり，原理は単純である．反復計算において$\boldsymbol{n}$を固定のもとqを仮定したとき，平面と格子との各辺での交点は容易に計算できるので，多面体V_rの各頂点の座標値を決定することができる．参考までに平面と格子の任意面との交差面の頂点座標を求めるコーディング例を Appendix Code3 に示す．

多面体の面の数をN_p，各面の面積をs_j，ある点（切り取る平面上の任意の点でよい）から各面までの高さをh_jとおくと，体積は次式で求められる．なおh_jは多面体の面の法線方向に沿った距離であり，多面体平面と点との距離の公式から簡単に計算され，正負の符号を取ることに注意する．

$$V_r = \frac{1}{3}\sum_{j=1}^{N_p} s_j h_j \tag{10}$$

各面のs_jは，面を構成する辺の数をN_e，各辺の長さをl_k，面上のある点（頂点のいずれかでよい）から各辺までの高さをh'_kとおくと次式で求められる．

$$s_j = \frac{1}{2}\sum_{k=1}^{N_e} l_k h'_k \tag{11}$$

式(9)の再構築が収束したら，その平面方程式が各面を切り取る面積の割合がそのまま流体率の面値に該当しフラックスを計算することで有限体積法により以下の流体率に関する移流方程式を解くことができる．

$$\frac{F^{n+1} - F^n}{\Delta t} = -\boldsymbol{u}^n \cdot \nabla F^n \tag{12}$$

なお筆者らは PLIC 法のような収束反復計算を必要としない STAA 法[2]を公開している．計算精度について次章で評価する．

各非構造格子において，式(12)で流体率を計算することにより密度は

$$\rho = (1 - F)\rho_G + F\rho_L \tag{13}$$

で直ちに求めることができる．G，Lは気相，液相を示す．粘性係数についても同様に計算できる．

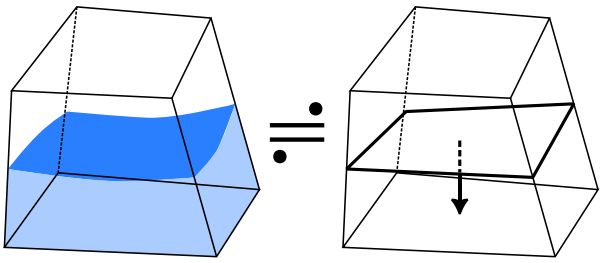

Fig. 3 Approximation of interface on PLIC

3. 計算手法の精度検証

任意多面体非構造格子の PLIC に関しては前節のように丁寧にアルゴリズムを整理することによりコーディングは実装可能ではあるものの，これまではそのシミュレーション精度評価が報告されている事例は見受けられない．今回，三角形不均一格子による二次元移流計算テスト[8]により，式(14)の計算精度検証を行った．図 4 にベンチマークモデルを示す．二次元計算領域の中央上側に切り欠きの入った円を配し，円の内側の格子の流体率を 1，外側を 0 と設定する．格子は図 4 の右側の拡大図のとおり三角形の形状を不規則に変形させたものとした．領域中心を回転軸とした反時計回りの剛体回転速度場を設定し，一回転後の計算結果を評価する．流体率Fは気液界面近傍で空間的に値が急激に変化するため，∇Fの単純な最小二乗法の計算では，精度よく$\boldsymbol{n}$を求めることができない．PLIC については，標準的な Youngs の方法[9]を用いた．この方法は，まず節点（格子の頂点）の周りの隣接格子のFから最小二乗法で勾配を計算し，次に格子の中心点において節点の勾配を平均することにより∇Fを求める．一方 STAA 法は[2]同様にレベルセット関数Gを構築して，隣接格子のGから最小二乗法で$\boldsymbol{n}$を求める．Appendix の Code4 に，PLIC の Youngs の方法で用いる最小二乗法計算のコーディング例を示す．このコードでは 4×4 行列解法としてクラメルの公式を用いている．

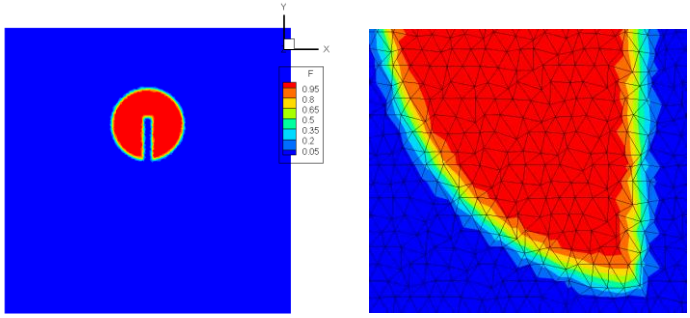

Fig. 4 Rudman's benchmark test [6] for the evaluation of accuracy on interface calculation methods

一回転後の計算結果を図 5 に示す．比較として TVD 法の結果を併記する．一般的な移流方程式解法である TVD 法では，気液界面に数値拡散が生じ，流体率の移流計算には不適合であることが自明である．一方 PLIC 法と STAA 法との比較では，気液界面の数値拡散は PLIC が僅かに上回るもののほぼ同等である一方，切り欠き円の形状の保存性では角部などで僅かに STAA 法が優れる．計算結果の L_1 エラー

$$L_1 = \frac{\sum_i |F_i - F_i^{init}|}{\sum_i |F_i^{init}|} \tag{14}$$

の比較では，TVD が 2.70×10^{-1}，PLIC が 1.85×10^{-2}，STAA が 2.48×10^{-2} とわずかに PLIC が上回るもののほぼ同等の結果となった．いずれにせよ PLIC 法，STAA 法ともに精度，安定性の点で非構造格子での適用性は良好である．一方 TVD 法は図 5 の結果より，数値拡散が大きくFの移流には適さないものの，式(4)の移流計算における人工粘性係数を必要としない安定的な解法であることが示されている．

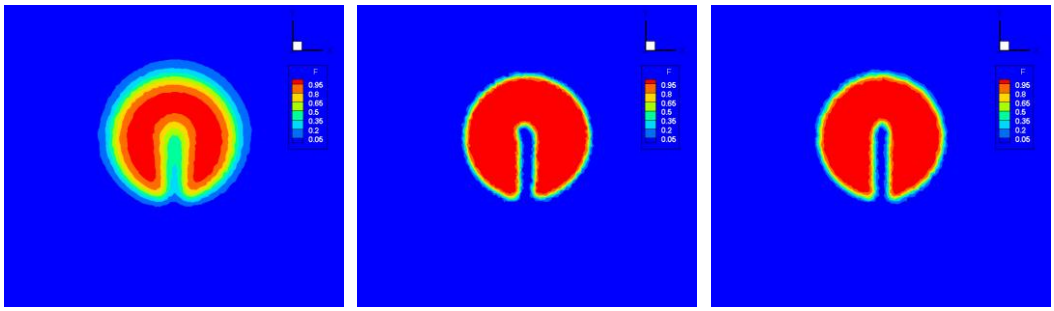

Fig. 5 The results of Rudman's benchmark test [6] after one evolution (left : TVD, middle : PLIC, right : STAA[2])

4. 高性能計算（High Performance Computing, HPC）への適用

多相流シミュレーションを企業 CAE として製品設計に活用するためには，複雑な製品モデルにおいて十分なメッシュ解像度で高速に計算することが必要である．1 億メッシュ以上の大規模計算モデルに対応するため，メモリ分散並列と CPU 演算における最適化を実装した．非構造格子の領域分割法では METIS などのオー

プンソースコードを用いることが可能だが，各種パラメータの調整にノウハウが必要な場合がある．企業で簡易に利用でき，かつ複雑形状製品への適用が可能なアルゴリズムの実装方法を説明する．

4.1. 非境界整合領域分割法（Non-Conformal boundary Domain Decomposition Method, NC-DDM）

分割された計算領域の間では，流体計算に必要な隣接格子の情報を MPI によりデータ交換する．図 6 に示すとおり，隣接領域に送信すべき格子群（紫色）の値を配列にパッキングし，MPI_Isend()により非同期送信する．さらに隣接領域から受信すべき格子群（緑色）の値の配列を MPI_Irecv()により非同期受信し，自領域の仮想格子に展開する．これを全ての領域が，全ての隣接領域に対して行う．したがって非同期送信および非同期受信のシステムコールが一斉に起動されるが，各プロセスは最後に MPI_Waitall()をコールすることで，システム側が一斉通信の処理を完了するまで wait がかかる．送信すべき格子群は，相手領域の全ての節点に対して共有しうる自領域の格子のリストとなる．同様に受信すべき格子群は，自領域の全ての節点に対して共有しうる相手領域の格子のリストとなる．これらの格子リストは，計算開始前に作成して配列にあらかじめ保存しておくことにより，計算開始後の通信ではその配列を参照してデータ送受の処理を簡略化できる．

具体的な実行手順を示す．まずプロセスは図 6 の紫色，緑色で示される境界格子について先に計算処理を行い，全て計算が完了したら MPI_Isend()，MPI_Irecv()を一斉に発行する．非同期なのでただちにプロセスに制御が戻ってくるため，システム側が通信処理をしているのと並行で，次にプログラム側は境界以外の格子の計算を行う．各領域の計算格子数が境界格子数よりも十分大きければ，この「境界以外の格子の計算」は「境界格子の通信」の処理時間を完全に隠蔽することができる．境界以外の格子の計算が完了したら MPI_Waitall()を発効する．隠蔽が成功していたら通信は既に完了しているため，直ちにプロセスに制御が返ってくる．通信が終わっていない場合は，通信が完了するまでブロックされ，計算効率の低下が発生する．したがって領域分割では適切な領域数で均等に領域分割することが重要となる．提案している非境界整合領域分割法（Non-Conformal boundary Domain Decomposition Method, NC-DDM）[10][2]のアルゴリズムを示す．

1. X，Y および Z 軸方向に分割された仮想の粗い直交格子=バケット（図 7 左）を用意する．
2. 各非構造格子がどのバケットに属するか，バケットごとに格子番号リストを作成する（図 7 右）．
3. 非構造格子数の合計が各分割領域において最も均等に近くなるように，全領域のバケット群に対して Z 軸方向について Mz 個に分割する．
4. 同様に，Mz 個に分割された各領域のバケット群に対して，Y 軸方向について My 個に分割する．
5. 同様に，My×Mz 個に分割された各領域のバケット群に対して，X 軸方向について Mx 個に分割する．
6. Mx×My×Mz 個に分割された各領域のバケット群において，格子番号リストを結合する．

バケットのサイズは，占有メモリが大きくなりすぎず，かつ領域分割が均等に近くなるよう，平均格子間距離の 1～2 倍程度とする．3，4 および 5 の手順で各軸方向に順次分割するため，バケットの分割領域は境界不整合となる．しかし非構造格子ではいかなる場合も領域同士の通信は図 6 のように不規則となるため，バケットの分割領域が整合，不整合は意味を持たない．先述したとおり，領域どうしの通信格子のリストを事前に選び出し，配列に保存しておくことで，ソルバーでの領域間通信を簡素に実現することができる．

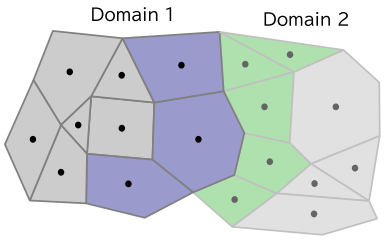

Fig. 6 The method of data exchange on unstructured grids with Message Passing Interface (MPI)
(VIOLET : Cell values to be send to domain 2, GREEN : Cell values to be received from domain 2)

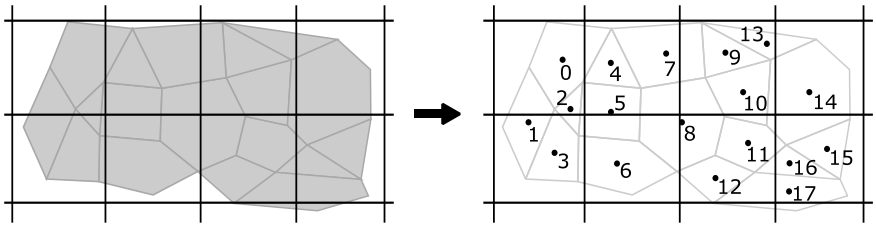

Fig. 7 Setting the bucket for unstructured grids on Non-Conformal Domain Decomposition Method (NC-DDM)

4.2. GPU，スパコン富岳への対応

任意多面体非構造格子多相流の計算においては，$\nabla\cdot\boldsymbol{v}$のようなフラックスの積分計算がプログラムの多くの部分を占める．各非構造格子の面の数は格子によって異なるが，おおよそ 10 以内である．GPU でのカーネル関数のコーディングを図 8 左に示す．N が格子数，M が各格子の面の最大値，p が隣接格子番号を表す．7 行目を“`for (i = 0; i < N; i++) {`“ と変えて 4，5 行目を削除すると，通常の CPU コードになる．GPU は数百以上の非常に多くのスレッドをデバイスで同時実行できるため，カーネル関数の呼び出し側で blockDim.x および gridDim.x を例えば各々256 と設定することで，図 8 左のコードは CPU よりも大幅に短い時間で計算を完了させることができる．

一方スパコン富岳で性能を発揮するためには，A64FX のコア数が GPU よりも大幅に少ないため，最外側のループ i に関するスレッド並列だけでなく，最内側ループのベクトル化とソフトウェアパイプライニングを適用させ，A64FX の演算器を効率的に利用する必要がある．通常の CPU コードでの最内側 j のループのベクトル長 M では不十分と考えられる．そこで，図 8 右に示す三重ループ構造への変更を行う．最外側ループを NS 刻みの増分とし，ループ長 NS の ii に関するループを最内側に挿入することにより，最内側ループのベクトル長を大きくできる．外側ループは OpenMP のディレクティブを挿入することでスレッド並列実行が可能となる．また最内側の ii に関するループには富士通コンパイラのディレクティブを挿入する．試行結果，NS=64 で良い結果が得られ，それ以上では性能はほぼ向上しないことが分かった．最内側ループはアンローリング展開をせず，またソフトウェアパイプライニングを促すようにディレクティブ指示することでさらに良い結果を得られることが分かった．Appendix の Code5 および Code6 に，式(4)の有限体積法計算における GPU コードおよび A64FX コードを示す．

また非構造格子では中心差分式の代わりとして最小二乗法が物理量の勾配計算に頻繁に用いられるため，全体の計算時間における最小二乗法の割合がある程度を占める．最小二乗法についても図 8 のループ構造になっているため，同様の最適化コーディングを適用できる．さらに式(8)のポアソン連立方程式の求解も計算負荷が大きく，その中で $\boldsymbol{q}^l=\mathbf{A}\boldsymbol{p}^l$ の行列とベクトルとの積の計算が大半を占めるが，こちらも有限体積法計算同様の二重ループ構造となるため，図 8 の最適化が適用可能である．

なお PLIC 法や STAA 法においてはアルゴリズムが多少複雑となるため，図 8 右の内側ループベクトル化ができない部分が存在する．ただしいずれも流体率 F が $0<F<1$ の格子のみ再構築を行えばよいため，最外側ループを全格子ではなく再構築対象格子のみにループ長を大きく縮減でき，結果として全体計算時間における PLIC および STAA のインパクトを抑えることができる．

```
int i, ns, j, p;
double sum;

i = threadIdx.x + blockDim.x * blockIdx.x;
ns = blockDim.x * gridDim.x;

for (; i < N; i += ns) {
   sum = 0.0;
   for (j = 0; j < M; j++) {
      p = neighbor(i,j);
      …
      sum += v[p] * ...
   }
   val[i] += sum * ...
}
```

```
int i, ii, i0, j, p;
double sum, sump[NS];

#pragma omp parallel for private(ii,j,i,p,sump)
for (i0 = 0; i0 < N-N%NS; i0 += NS) {
   for (ii = 0; ii < NS; ii++) sump[ii] = 0.0;
   for (j = 0; j < M; j++) {
      #pragma loop nounroll
      #pragma loop swp
      for (ii = 0; ii < NS; ii++) {
         i = i0 + ii;
         p = neighbor(i,j);
         …
         sump[ii] += v[p] * ...
      }
   }
   #pragma loop nounroll
   #pragma loop swp
   for (ii = 0; ii < NS; ii++) {
      i = i0 + ii;
     val[i] += sump[ii] * ...
   }
}
// 残りの部分，ループ内は従来コードと同じ
for (i = N-N%NS; i < N; i++) {
   ...
}
```

Fig. 8 A typical coding for unstructured grids (left : for GPU-CUDA, right : for A64FX)

5. 設計適用検証

本論文の CAE モデルの適用例として，図 9 に示す海外富裕層向け高級シャワー現行製品モデルにおいて多相流計算を実施する．このシャワーは浴室上部に固定設置される．ミリ単位の吐水口径や流路幅を含む微細かつ複雑な流路構造を持ち，300mm 径の大型な製品であることから，シャワー流路流れおよび吐水を同時に計算するために，図 10 の大容量の非構造格子流路モデルを用いる．格子数は 15,549 万である．

本モデルでの A64FX における 4.2 節のコード最適化適用前後でのホットスポットにおける計算性能比較結果を図 11 に示す．いくつかのサブルーチンでは高速化の効果が得られないものがあるものの，大半のサブルーチンで半減以下になっており，全プログラムでは最適化前の 41%の計算時間まで低減できた．改良後の 1CPU の計算速度は，富岳リリース当時の GPU を用いた図 8 左でのコードの 1GPU の計算速度に若干およばない程度であり，富岳の多数の CPU を用いることで，混相流シミュレーションにおいて一般的 GPU サーバーでは利用不可能な計算処理性能が得られる．このモデルの富岳における強スケールの並列化効率を図 12 に示す．A64FX の 1CPU は CMG (Core Memory Group)とよばれる 12 演算コアと内部キャッシュで構成された処理単位 4 つで構成されており，各 CMG が 8GB の HBM2 メモリに 256GB/sec で直結された NUMA 構成である．本モデルでは 96CMG 未満の計算はメモリ不足により実行できないため，96～240CMG の計算結果を示す．図 13 には 96CMG の場合の境界非整合領域分割のようすを示す．複雑な流路領域においても，1CMG に 1 プロセス 12 スレッドを割り当てた 240 並列 (60CPU，2,880 スレッド)で 96 並列の約 2.1 倍，1 プロセスの約 197 倍（推定値）の強スケーラビリティが得られており，本モデルの富岳における実用的な並列化効率が示されている．

Fig. 9 Overhead shower for the test simulation

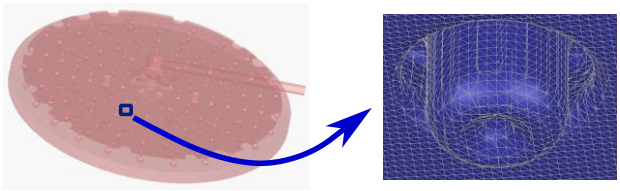

Fig.10 The test simulation model for unstructured grids

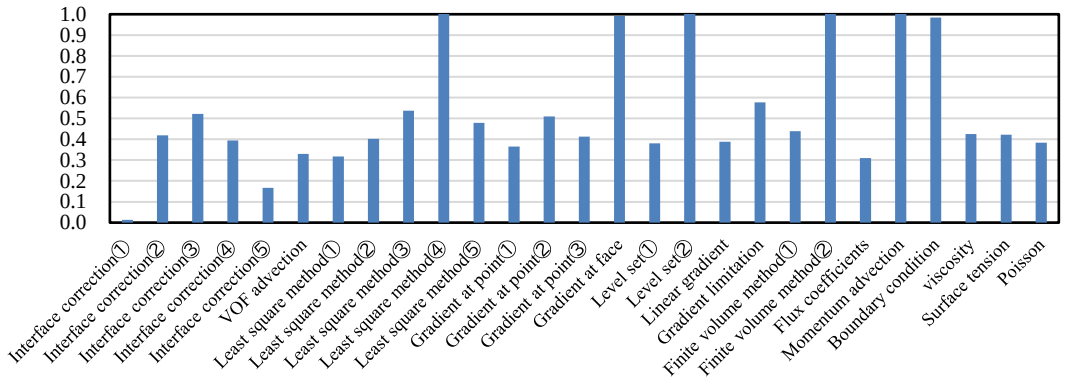

Fig. 11 The result of (Elapsed time of A64FX-optimised code) / (elapsed time of original code)

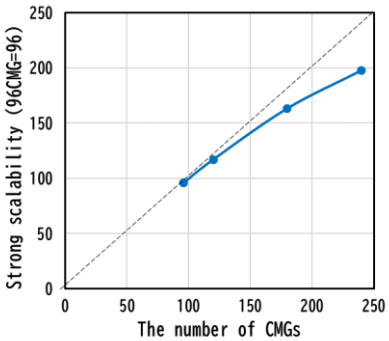

Fig. 12 The strong scalability of the simulation

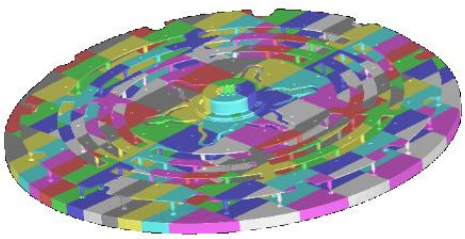

Fig.13 The result of NCDDM by 96 domains

図 14 にシミュレーション結果を示す．閉領域の流路に水が浸入することで流路の空気を取り込み，複雑な気液界面や気泡を形成しながら流路を充填していくようすや，充填後に押し出すように吐水口からの吐水が始まるようす，閉止後に流路に充填した水が少しずつ落下していくようすが捉えられている．この一連の吐水→止水プロセスの実時間 1.5 秒間のシミュレーションに要した富岳の計算時間は 240 並列で約 150 時間であり，設計適用に利用可能な範囲に収まった．スパコン富岳の膨大な計算資源を利用することにより，多数のケーストライを並行して計算でき，流路設計業務の期間短縮に貢献可能といえる．

高級シャワーは製品品質が求められるため，閉止後の水垂れを抑制することは重要な設計課題の一つである．図 15 に計算モデルで設定する傾斜角度を示す．図 15 の場合，向かって右側の吐水口からの水垂れが発生しやすくなる．設定傾斜角度における水垂れの有無について実機と比較検証を行った．表 1 に示すように，定量的には実機では傾斜角 5° での水垂れ発生に対し計算結果では傾斜角 6° での発生となっており，わずかな誤差に収まっているといえる．実機試験では，傾斜角 4～5° 近傍では水垂れの有無は安定しない結果であったことを考慮すると．本シミュレーション結果はおおむね妥当といえる．

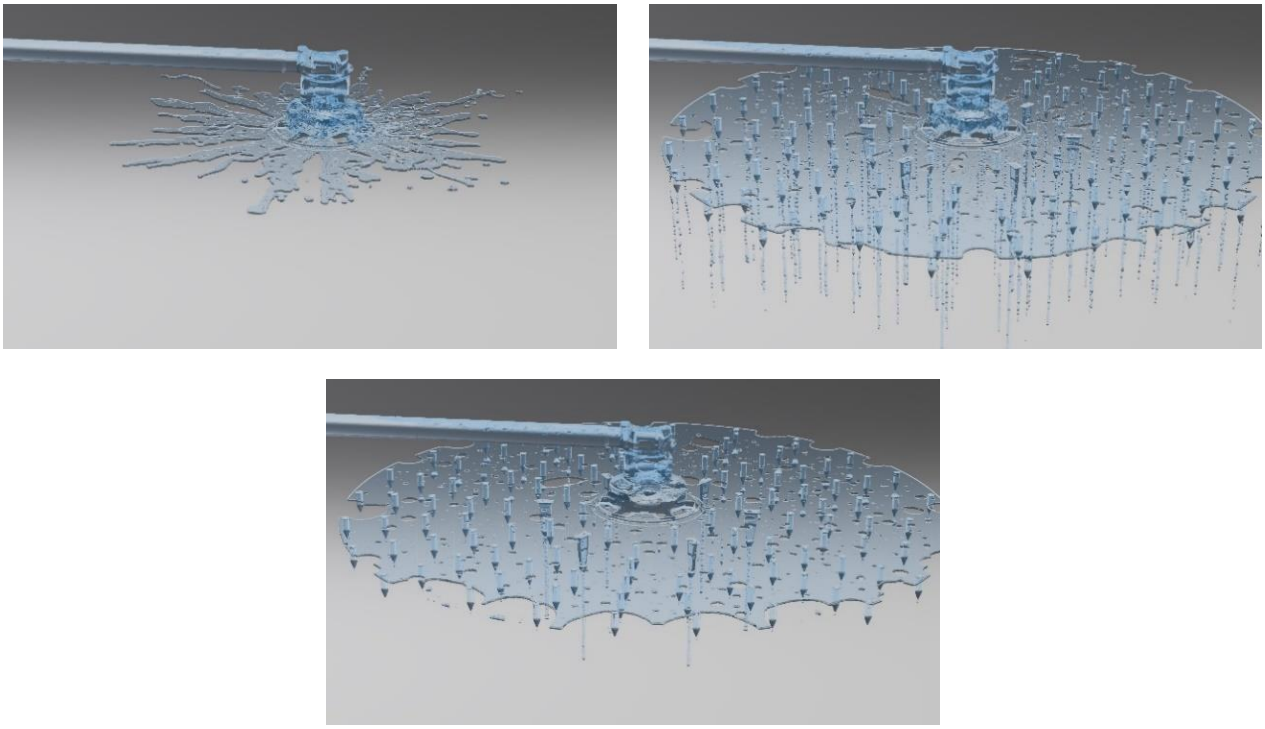

Fig. 14 The result of the simulation
(upper-left : water-filling step, upper-right : spouting step, lower : shut-off and leaking step)

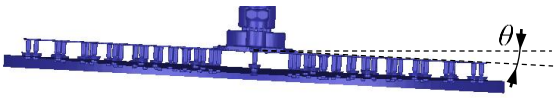

Fig. 15 The model for evaluating leakage.

Table 1 Comparison between experiment and calculation

Inclined angle θ	Leakage	
	Experiment	Calculation
0°	○	○
5°	×	○
6°	×	×
8.5°	×	×

○：Non-existence of leakage，×：Existence of leakage

6. 結論

非構造格子の気液二相流シミュレーションを核とした水まわり製品流体解析 CAE のフレームワークを提案した．TVD 拘束などのロバスト性向上のための定式化や富岳の計算高速化・並列化により，1 億メッシュ以上の複雑流路の製品の計算を実施できるようになった．シャワー止水後の水垂れなどの吐水品質設計への適用が可能となったことを示した．

今回公開したフレームワークはいずれも既に詳細の定式化までオープンとなっている計算手法の組み合わせで構築されており，C 言語のプログラム開発ができる技術者ならば容易にコード生成できるレベルの複雑度および難易度と考えられる．企業 CAE の取り組みにおいて，従来の海外製を中心とした商用コードの利用ではない，自社運用に最適化された流体 CAE 環境構築に利用可能である．コードを自社内でハンドリングすることにより，これまで GPU や富岳といった当時最新の HPC システムにすぐに適用できたことが大きな利点となった．将来の新しい HPC 技術にも本フレームワークをいち早く適合させることが期待できる．また計算手法については，例えば運動方程式の移流項については TVD 法と保存型 CIP 法を関数パッケージとして用意しておき，問題に応じて関数の呼び出し側を変更することで両者の使い分けが可能である．さらに TVD 法は Van Leer だけでなく様々なリミッタの式に修正して試行することも容易である．例えば二次風上法に変更するならば，$\psi(r_f) \equiv (r_f + 3)/4$とすればよい．同様に気液界面方程式においては PLIC 法と STAA 法を使い分けることも簡単にできる．表面張力計算についても，CSF モデルを他のモデル，例えばレベルセット法に基づく方法など，対象とする問題に応じて柔軟に変更できる．それらの試行錯誤は，自社内でのノウハウの蓄積となり他社差別化にもつながると考えられる．いずれにせよ柱となるフレームワークを確定することが前提であり，長期間にわたる CAE ツールとしての利用に耐えうるように基本土台を構築する必要がある．

なお本フレームワークで用いる非構造格子は OpenFOAM をはじめとして様々な非商用・商用の流体解析ソフトウェアで格子生成可能である．一例として商用ソフトウェア scFLOW を用いた非構造格子作成について手順の概要を Appendix B に示す．

今後は，提案した CAE フレームワークを様々な新製品開発に適用し，製品品質向上への寄与を目指すとともに，得られたデータをもとにさらなる計算精度，計算性能の向上を目指す．

謝辞

本研究の第四章および第五章は，HPCI システム利用研究課題（課題番号：hp210013）を通じて，理化学研究所のスーパーコンピュータ「富岳」の計算資源の提供を受け，2021 年 3 月から 2022 年 3 月にかけて実施した．ここに記して謝意を表する．

参考文献

[1] M. S. Darwigh and F. Moukalled，Tvd schemes for unstructured grids，Int. J. Heat Mass Transfer，Vol.46，No.4，p.599，2003.
[2] 池端昭夫, 清水友哉, 肖鋒，保存型 CIP 法による多相流の効率的な非構造格子数値計算モデル，日本計算工学会論文集，p.20180001，2018.
[3] J. U. Brackbill，D. B. Kothe and C. A. Zemach，A continuum method for modeling surface tension，J. Comput. Phys.，Vol.100，p.335，1992.
[4] T. Yabe and P. Y. Wang，Unifed numerical procedure for compressible and incompressible fluid，J. Phys. Soc.，Vol.60，p.2105，1991.
[5] R. Barrett，F. T. Chan，J. Donato，M. Berry and J. Demmel，反復法 Templates（応用数値計算ライブラリ），朝倉書店，1996.
[6] W. J. Rider and D. B. Kothe，Reconstructing volume tracking，J. Comput. Phys.，Vol.141，p.112，1998.
[7] R. P. Brent，Algorithms for minimization without derivatives，Dover Books on Mathematics，2013.
[8] M. Rudman，Volume-tracking methods for interfacial flow calculations，Int. J. Numer. Methods Fluids，Vol.24，p.671，1997.
[9] D. L. Youngs，An interface tracking method for a 3d eulerian hydrodynamics code，Technical Report，44/92/35，AWRE，1984.
[10] 池端昭夫，肖鋒，大規模並列 GPU 計算の衛生陶器多相流シミュレーションへの適用，情報処理学会誌コンピューティングシステム，Vol.9，No.4，p.1，2016.

付録 A 任意多面体非構造格子のコーディング例

本稿で記述した主な定式化の説明の補完として，コーディング例を Code 1～6 に示す.

Code 1 Finite volume method for unstructured grids

```
//
// 有限体積法のコード例
//
// Np                : 格子数
// nface[i]          : 格子番号iの面の数
// Pface_number(i,j) : 格子番号iのj番目の面番号+1，符号で表裏を表す
// flux[f]           : 面番号fのフラックス，移流項の場合uφ，事前に計算しておく
// area[f]           : 面番号fの面積
// volume[i]         : 格子番号iの体積
//

int    i, j, f0, f, s;
double sum;

for (i = 0; i < Np; i++) {
     sum = 0.0;
     for (j = 0; j < nface[i]; j++) {
          f0 = Pface_number(i,j);
          f = abs(f0) - 1;
          s = (f0 < 0) ? -1 : 1;
          sum += flux[f] * area[f] * s;
     }
     phi[i] += delta_T * sum / volume[i];
}
//
```

Code 2 TVD scheme for unstructured grids

```
//
// TVD法のコード例
//
// Nf                        : 面の数
// Fcell_number(i,j)         : 面番号iに接するj番目の格子番号
// uf[i], vf[i], wf[i]       : 面番号iの流速ベクトル(u,v,w)
// nvx[i], nvy[i], nvz[i]    : 面番号iの法線ベクトル
// phix[C], phiy[C], phiz[C] : 格子番号Cのφの勾配ベクトル，事前に最小二乗法で計算しておく
// x[C], y[C], z[C]          : 格子番号CのX,Y,Z座標
//

int    i, j1, j2, C, D;
double dot, phi_C, phi_D, phi_U, rf, psi;

for (i = 0; i < Nf; i++) {
     j1 = Fcell_number(i,0); j2 = Fcell_number(i,1);
     dot = uf[i] * nvx[i] + vf[i] * nvy[i] + wf[i] * nvz[i];
     if (dot > 0.0) {
        C = j1; D = j2;
     } else {
        C = j2; D = j1;
     }
     phi_C = phi[C]; phi_D = phi[D];
     phi_U = phi_D + 2.0 * (phix[C] * (x[C] - x[D]) + phiy[C] * (y[C] - y[D])
                            +phiz[C] * (z[C] - z[D]));
     rf = (phi_C - phi_U) / (phi_D - phi_C);
     psi = (rf + fabs(rf)) / (1.0 + fabs(rf));
     phif[i] = phi_C + 0.5 * psi * (phi_D - phi_C);
}
//
```


Code 3 Generation of the face which has been cut by the interface-plane for PLIC reconstruction

```
56  //
57  // PLIC再構築のための任意面を気液界面で切断したときの面生成のコード例
58  //
59  // i                        : 現在処理している面番号
60  // nedge[i]                 : 面番号iの辺の数
61  // Fnode_number(i,j)        : 面番号iのj番目の節点番号
62  // nx, ny, nz               : 切り取る平面の単位法線ベクトル
63  // q                        : 切り取る平面の切片q
64  // codn[j*3]                : 節点番号jの三次元座標ベクトル
65  // edge_plane_cross_pt(..) : 2点を結ぶ線分と平面との交点を生成する関数
66  // point_list[]             : 生成された平面の頂点座標リスト
67  //
68
69  int    j, j0, j1, j2, k, M, in_liquid;
70  double *p1, *p2, *pm, r;
71
72  M = nedge[i];
73  j0 = Fnode_number(i,0);
74  k = 0;
75
76  r = nx * codn[j0*3] + ny * codn[j0*3+1] + nz * codn[j0*3+2] + q;
77  in_liquid = 0;
78  if (r >= 0.0) in_liquid = 1;
79
80  for (j = 0; j < M; j++) {
81       j1 = Fnode_number(i,j);
82       j2 = Fnode_number(i,(j+1)%M);
83       p1 = &(codn[j1*3]);
84       p2 = &(codn[j2*3]);
85       pm = edge_plane_cross_pt(p1,p2,nx,ny,nz,q);
86       if (pm != NULL) {
87                 point_list[k++] = pm;
88                 in_liquid = 1 - in_liquid;
89       }
90       if (in_liquid) point_list[k++] = p2;
91  }
92  //
```


Code 4 Calculation of gradient at vertices with Least Square Method for unstructured grids

```
//
// 最小二乗法のコード例
//
// Nn                  : 節点数
// ncell[i]            : 節点番号iに接続している格子数
// Ncell_number(i,j)   : 節点番号iに接続しているj番目の格子番号
// x[p], y[p], z[p]    : 格子番号pのX,Y,Z座標
// xn[i], yn[i], zn[i] : 節点番号iのX,Y,Z座標
// f[p]                : 格子番号pの物理量
// fx[i], fy[i], fz[i] : 節点番号iの物理量の勾配ベクトル
// determinant4(..)    : 4x4行列の行列式計算のインライン関数
//

int    i, j, p;
double a11, a12, a13, a14;
double a22, a23, a34;
double a33, a34;
double a44;
double b1, b2, b3, b4;
double xx, yy, zz, pp, detA, fx0, fy0, fz0;

for (i = 0; i < Nn; i++) {
     a11 = 0.0; a12 = 0.0; a13 = 0.0; a14 = 0.0;
     a22 = 0.0; a23 = 0.0; a24 = 0.0;
     a33 = 0.0; a34 = 0.0;
     a44 = 0.0;
     b1 = 0.0; b2 = 0.0; b3 = 0.0; b4 = 0.0;
     for (j = 0; j < ncell[i]; j++) {
          p = Ncell_number(i,j);
          xx = x[p] - xn[i];
          yy = y[p] - yn[i];
          zz = z[p] - zn[i];
          pp = f[p];
          a11 += xx * xx;
          a12 += xx * yy;
          a13 += xx * zz;
          a14 += xx;
          a22 += yy * yy;
          a23 += yy * zz;
          a24 += yy;
          a33 += zz * zz;
          a34 += zz;
          a44 += 1.0;
          b1 += xx * pp;
          b2 += yy * pp;
          b3 += zz * pp;
          b4 += pp;
     }
     detA = determinant4(a11,a12,a13,a14,a12,a22,a23,a24,a13,a23,a33,a34,a14,a24,a34,a44);
     fx0 =  determinant4( b1,a12,a13,a14, b2,a22,a23,a24, b3,a23,a33,a34, b4,a24,a34,a44) / detA;
     fy0 =  determinant4(a11, b1,a13,a14,a12, b2,a23,a24,a13, b3,a33,a34,a14, b4,a34,a44) / detA;
     fz0 =  determinant4(a11,a12, b1,a14,a12,a22, b2,a24,a13,a23, b3,a34,a14,a24, b4,a44) / detA;
     fx[i] = fx0;
     fy[i] = fy0;
     fz[i] = fz0;
}
//
```

Code 5 Finite volume method for unstructured grids (GPU-CUDA code)

```
//
// 有限体積法のコード例，GPU(CUDA)バージョン
//
// Np                : 格子数
// nface[i]          : 格子番号iの面の数
// Pface_number(i,j) : 格子番号iのj番目の面番号+1，符号で表裏を表す
// flux[f]           : 面番号fのフラックス，移流項の場合uφ，事前に計算しておく
// area[f]           : 面番号fの面積
// volume[i]         : 格子番号iの体積
//

int    i, j, ns, f0, f, s;
double sum;

i = threadIdx.x + blockDim.x * blockIdx.x;
ns = blockDim.x * gridDim.x;
for (; i < Np; i += ns) {
     sum = 0.0;
     for (j = 0; j < nface[i]; j++) {
          f0 = Pface_number(i,j);
          f = abs(f0) - 1;
          s = (f0 < 0) ? -1 : 1;
          sum += flux[f] * area[f] * s;
     }
     phi[i] += delta_T * sum / volume[i];
}
//
```

Code 6 Finite volume method for unstructured grids (A64FX code)

```
//
// 有限体積法のコード例，A64FX(富岳)バージョン
//
// Np                : 格子数
// MFACE             : 格子番号iの面の数の最大値
// Pface_number(i,j) : 格子番号iのj番目の面番号+1，符号で表裏を表す
// flux[f]           : 面番号fのフラックス，移流項の場合uφ，事前に計算しておく
// area[f]           : 面番号fの面積
// volume[i]         : 格子番号iの体積
//

#define   NS   64

int    i0, i, j, ii, f0, f, s, Np2;
double sump[NS], sum;

Np2 = Np - Np % NS;

#pragma omp parallel for private(i,j,ii,sump,f0,f,s)
for (i0 = 0; i0 < Np2; i0 += NS) {
     for (ii = 0; ii < NS; ii++) sump[ii] = 0.0;
     for (j = 0; j < MFACE; j++) {
          #pragma loop nounroll
          #pragma loop swp
          for (ii = 0; ii < NS; ii++) {
               i = i0 + ii;
               f0 = Pface_number(i,j);
               f = abs(f0) - 1;
               s = (f0 < 0) ? -1 : 1;
               sump[ii] += flux[f] * area[f] * s;
          }
     }


```

```
211       #pragma loop nounroll
212       #pragma loop swp
213       for (ii = 0; ii < NS; ii++) {
214            i = i0 + ii;
215            phi[i] += delta_T * sump[ii] / volume[i];
216       }
217  }
218
219  // 残りの部分，ループ内は従来コードと同じ
220  for (i = Np2; i < Np; i++) {
221       sum = 0.0;
222       for (j = 0; j < nface[i]; j++) {
223            f0 = Pface_number(i,j);
224            f = abs(f0) - 1;
225            s = (f0 < 0) ? -1 : 1;
226            sum += flux[f] * area[f] * s;
227       }
228       phi[i] += delta_T * sum / volume[i];
229
230  }
231  //
```

付録 B scFLOW による非構造格子生成およびソルバーへの入力の手順概要

本稿で提案した非構造格子流体計算ソルバーにおいて，一例として市販ソフトウェア scFLOW による格子生成からソルバー読込みに至る手順のガイドを示す．

手順 1．非構造格子生成

非構造格子を，scFLOW のプリプロセッサ scFLOWpre で生成する．scFLOWpre は，scFLOW を起動した際に表示されるメニューの該当のアイコンをクリックすることで開始できる．クイックスタートとして，同じメニューにある「ユーザーズカイド」をクリックし，「操作編」→「チュートリアル」の順で適当なチュートリアルを選択し，表示される操作手順に従ってプリプロセッサ使用方法を学習しながら実施することができる．その中で非構造格子生成については，「八分木設定／メッシュ設定」のところで具体的な操作手順が示されている．基本的な流れとしては，まず八分木から四面体格子が自動生成され，その後四面体格子どうしの結合により多面体格子が自動生成される．四面体格子から多面体格子への変換により，計算領域全体のメッシュ数を削減することができる．

メッシュ作成が終了したら，scFLOWpre でファイルを書き出す．いくつかのファイルが出力されるが，その中で .gph ファイルが GPH ファイルとよばれるもので，非構造格子データが入っている．そのファイルフォーマットは，「ユーザーズガイド」の「ソルバー編」→「ファイル」で公開されている．

手順 2．ソルバーへの読込み

本稿で提案したフレームワークの流体解析ソルバーに，GPH ファイルを読み込む前処理部を追加する．GPH ファイルフォーマットに従って `fread()` などのバイナリーリードシステムコールで読み込むことができる．通常，バイトオーダーは BIG_ENDIAN である．ただし scFLOWpre が FORTRAN を前提としているため，C や C++のプログラムから読み込む場合は，FORTRAN 形式バイナリーに沿って，各レコードの先頭と後端に配置される，間のデータのバイト数を示す 32 ビット整数値（＝4 バイト）を読み飛ばす必要がある．

Appendix A で示したとおり，非構造格子では `Pface_number(i,j)` や `Ncell_number(i,j)` に相当する，隣接格子や隣接面などとの接続情報（コネクティビティとよばれる）を設定する必要がある．これらのコネクティビティは通常二次元配列に保存されるが，GPH ファイルに用意されていないものについては，他のコネクティビティ情報から，自前でコーディングする必要がある．