Vol.2019-HPC-168 No.9
2019/3/6

河川流域のオープンデータと連携した実時間志向の簡易流体・拡散連成シミュレーション

田中 涼[1,a)]　鳥羽 奏一朗[1]　日吉 莉菜[1,†1]　福間 慎治[1,b)]　森 眞一郎[1,c)]

概要：本論文では、河口近傍から上流域に跨る河川の流体計算と, その結果を反映した簡易な移流拡散計算を行う連成シミュレータの実装について報告する. 実装に際しては, 格子ボルツマン法を用いた流体計算の入力境界境界条件として河川流域の関連する動的オープンデータをリアルタイムに反映させるとともに, 流体シミュレーション結果の流速情報とオープンデータとして得られる風速データを考慮した移流拡散計算を行った. 高速化のための並列化においては, シミュレーション対象となる河川形状によって変わるノード間通信パターンの変更に伴うプログラミングの煩雑さを軽減するための基本形状テンプレート法の導入, 河川形状を考慮した通信量削減などを実装するとともに,GPU を用いた高速化を行った。

1. はじめに

近年の計算機性能の向上は目覚ましいものであり, この豊富な計算資源を用いたメニーコアプログラミングによるインタラクティブな実時間シミュレーション開発が注目されている. また, 豊富な計算資源を背景に複数の物理現象を組み合わせたマルチフィジックスシミュレーションの研究開発も盛んに行われている. しかしながら, 事前にシナリオを設定しておくのではなく, 実行中のインタラクティブな境界条件の変更に対応した実時間志向シミュレーションの試みは未だ発展途上である.

我々は, このような実時間志向シミュレーションの一例として, 時々刻々変換する観測データに基づく Web 上の動的オープンデータをシミュレーションの入力データとして供給し, 実世界と連動して物理シミュレーションを行うサイバーフィジカル・シミュレーションシステム（Cyber Physical Simulation System:CPSS) に着目し, その一例として河川流域の動的オープンデータと連携した流体・拡散連成シミュレーションシステム [1] を開発した. 本報告では、このシステムの高速化のための並列化ならびに GPU を用いたアクセラレーションに関する報告を行う.

本システムは, ユーザが様々な形状の河川に対して容易にシミュレーションを実行できることを目指している. そのため, 河川の形状をいくつかのパターンに分類し, それぞれのパターンに対応する並列プログラムでの通信パターンをテンプレート化することで, 地図情報からの自動プログラム生成を目標としているが, 今回の実装では通信パターンの選択は人手で行っている.

[1] 福井大学
Fukui Uniersity
[†1] 現在，セイコーエプソン株式会社
[a)] rtanaka@sylph.fuis.u-fukui.ac.jp
[b)] fukuma@u-fukui.ac.jp
[c)] moris@u-fukui.ac.jp

2. 計算手法

本研究では、格子ボルツマン法を用いた流体シミュレーションと拡散シミュレーションを組み合わせた連成シミュレーションの作成を行っている. 本研究における連成シミュレーションは, 液体や気体の流れを調べるための流体シミュレーションと, ある溶液などが入力点より広がっていく様子を観察するための拡散シミュレーションから成っている.

2.1 流体計算

流体シミュレーションでは, 流体の計算に格子ボルツマン法を用いている. 格子ボルツマン法は, 一般的な流体運動を定式化する際に用いられる Navier-Stokes 方程式を単純化した数理モデルであり, 元となる Navier-Stokes 方程式の近似解を求めることができる. 格子ボルツマン法は, 原則として空間は格子によって一様に離散化されている. 時間についても離散化され, 時間刻み Δt ずつ時間が進む. 流体は, 格子上に存在する仮想粒子の集合として捉える. この粒子は衝突と呼ばれる過程によって一部の粒子は速度の方向を変え,Δt の時間で静止, または別の格子点へ移動する. この粒子の格子間の移動を並進と呼ぶ. また, 粒子は個別に注目するのではなく, 時刻 t, 位置ベクトル r とした時の格子点

状において粒子速度ベクトル e_i をもつ粒子密度分布を, 実数値を持つ分布関数 $f_i(r,t)$ で表す. 一連の粒子運動は,

$$f_i(r+e_i\Delta t, t+\Delta t) = f_i(r,t) + \Omega_i(f(r,t)) \qquad (1)$$

と表され, これを格子ボルツマン方程式と呼ぶ. この式 (1) の左辺は, 衝突終了後に粒子が $e_i\Delta t$ 離れた近接の格子点に移動する並進過程を示している.Ω_i は, 衝突により変化した粒子分布を表した衝突演算子である.Ω_i は, 局所的な分布関数にのみ依存している.

格子ボルツマン法の計算方法は, 便宜上, 衝突, 並進, 巨視化の 3 つの過程を繰り返すものであると捉えることができ, このうち他の格子点との情報のやりとりは並進の時のみであり, 衝突, 巨視化における計算はすべて各格子点内で行われる.Ω_i に関して格子ボルツマン法で広く用いられている単一緩和時間近似を用いた格子 BGK モデルを考える. この格子 BGK モデルは格子点内の速度方向をあらわす分布関数との干渉を考慮する必要がない. 式 (1) の左辺は並進を表すが e_i の種類には限りがあり, 使用する速度分布にも依るが, 多くは $e_i\Delta t$ が隣接格子までの距離のみを表している. このように, 格子ボルツマン法は局所的な計算手法であるため, 複雑境界の形状に対して効果的であり, 並列計算にも適しているといわれている． 2 次元流体計算における格子 BGK モデルとして D2Q9 モデルを採用している.D2Q9 モデルにおいては, 正方形格子で空間を分解し, 粒子は静止を含む 9 種類の速度を持つ. このとき粒子速度 $e_i(i=0,1,...8)$ は以下のように定義される.

$$e_i = \begin{cases} (0,0) & i=0 \\ c=(\cos[(i-1)\pi/2], \mathrm{sim}[(i-1)\pi/2]) & i=1-4 \\ \sqrt{2}c(\cos[(2i-9)\pi/4], \sin[(2i-9)\pi/4]) & i=5-8 \end{cases} \qquad (2)$$

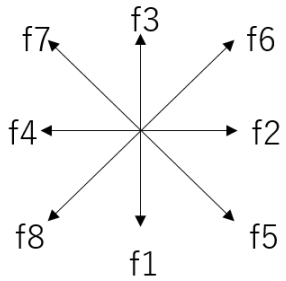

図 1 D2D9 モデル

2.2 拡散計算

拡散シミュレーションでは, 拡散計算として拡散方程式を用いている. 拡散方程式は拡散が生じている物質量の密度のゆらぎを記述する偏微分方程式である. 計算空間を等間隔で離散化した 2 次元正方格子を考える. この時各格子状にはある物理量 $C(x,y,t)$ が割り当てられているものとする. ここで,x,y,a は整数とする. 時刻 t から $t+\Delta t$ の間に, 各格子上の物理量が斜め格子を除いたいずれか隣の格子に確率的に移動する.D は拡散係数を表す.

通常の拡散計算では, この物理量の移動はどの方向に対しても均一である. この確率を p としたとき,(x,y) にあった物理量は D の確率で,$(x+a,y)$,$(x-a,y)$,$(x,y+a)$,$(x,y-a)$ のどれか 1 つに移動することになる. この時 $t+\Delta t$ における (x,y) の格子上の物理量 $C(x,y,t+\Delta t)$ の期待値 C' は

$$\begin{aligned} C'(x,y,t+\Delta t) = DC(x+a,y,t) + DC(x-a,y,t) \\ +DC(x,y+a,t) + DC(x,y-a,t) \end{aligned} \qquad (3)$$

と表せる. ここで,a,Δt が小さいとして式 (3) を Taylor 展開すると,

$$\frac{\partial C}{\partial t}\Delta t + o(\Delta t^2) = D(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2})a^2 + o(a^3) \qquad (4)$$

となる.$o(\Delta t^2)$,$o(a^3)$ を無視し, 式 (4) を整理すると,2 次元の拡散方程式

$$\frac{\partial C}{\partial t} = D(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})C, \quad D>0 \qquad (5)$$

が得られる.

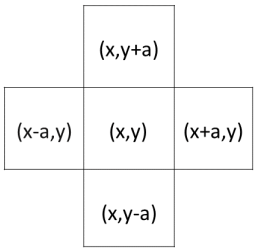

図 2 拡散方程式における格子

3. 連成シミュレーション

3.1 動的オープンデータを利用した流体シミュレーション

流体シミュレーションは川から海へ水が流れていく過程をシミュレートしており川の流入値が必須となる. 本研究では実環境に則したシミュレーションを目指しているため, 福井県が公開している河川の水位についてのオープンデータを流入値として用いることにした. 水位データは更新時刻にあわせて取得する. 流入速度の急激な変化によるシミュレーションの破綻を防ぐため, 取得したデータはそのまま流入値として反映するのではなく次のデータ取得が行われるまでの時間で緩やかに流入値を更新し, 流速を求める. 求めた値は, 移流拡散シミュレーションでの拡散係数に反映させる.

3.2 流体計算を考慮した移流拡散シミュレーション

本研究では, 流体計算結果と風の影響を考慮した移流拡散計算を行う. 本研究で使用する拡散シミュレーションでは固体と液体が混ざることで混相流が発生するようなものではなく, 比重が水よりも非常に小さく流体の流れに影響を及ぼさないまま拡散するものを想定している. 流体シミュレーションの計算結果を用いて拡散計算を行ったあとに流体シミュレーション側には拡散計算結果のデータを反映しない構成になっている. 拡散計算については 2.2 節で述べたが, 本研究では流速 u を考慮した移流項が必要となる. 式 (5) に移流項を加えた移流拡散方程式が

$$\frac{\partial C}{\partial t} + u(\frac{\partial}{\partial x} + \frac{\partial}{\partial y})C = D(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})C, \quad D > 0, \ (6)$$

となる. この拡散係数は風と流体の 2 要素によって求められる. ここでの風は,4 方向の向きを持っており Web のオープンデータ [2] から風速の情報を取得する. この風 (w_x, w_y) と流体シミュレーションから受け取った流体 (f_x, f_y) に重み $\alpha(0.0 \leq \alpha \leq 1.0)$ をそれぞれに加え合成することで流速 u を求める.

$$u = \alpha(w_x, w_y) + (1.0 - \alpha)(f_x, f_y) \qquad (7)$$

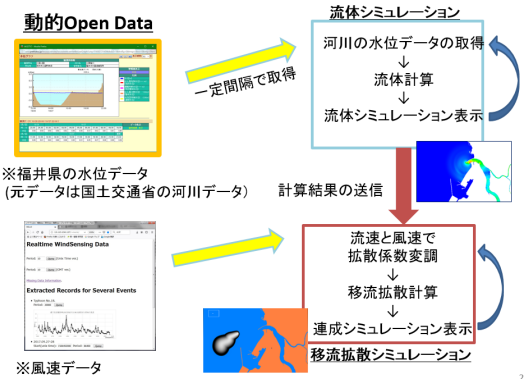

図 3 連成シミュレーションの流れ

4. 計算の高速化・並列化

本研究で開発した連成シミュレーションでは, 格子ボルツマン法や拡散計算など計算量が多く, 河川領域全体を計算するためには, 膨大な計算資源が必要となる. 本シミュレーションは実時間志向であるため,1 ノードで計算できる計算量には限りがある. そこで, 計算領域を複数ノードに分割したへ並列化によって計算負荷の分散をはかり,GPGPU の利用によって 1 ノードあたりの計算性能を向上させる. ノード間の通信には MPI,GPU の利用には OpenACC[6] を使用する.

4.1 計算の並列化

並列化は計算領域を複数に分け, それぞれの領域に計算機を割り当てる. 領域を割り当てられた各計算機は自身の領域を計算する. 流体計算の並進ステップや拡散計算では, 計算する格子の 1 つ隣の格子のデータを用いて計算する. そのため, 計算領域を分割した場合, 境界部分を計算するために自分自身のノードが持っているデータに加えて, 計算に必要な 1 つ隣のデータを保有するノードとの通信が必要となる. しかし後述する GPU 実装の場合は, 本研究で使用する GPU は GPU 間で直接通信ができないため CPU を介してデータの送受信を行う. 計算途中で境界データが必要になる度に 1 データずつ通信を行うのではなく,1 度に 1 ステップで必要な境界データをすべて通信することで, 通信の頻度を減らし, 冗長な通信のオーバーヘッドを削減する.

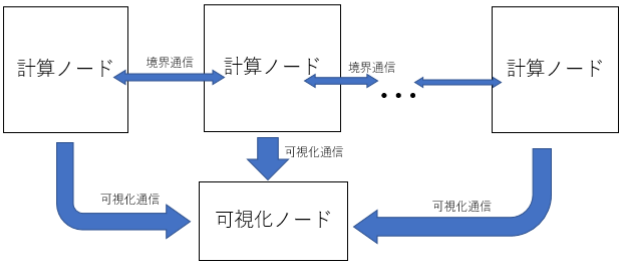

図 4 多ノードによる計算の並列化

4.2 河川形状を考慮した通信量削減

並列化した際に, ボトルネックとなるのは通信時間である. ここで, 図 5 のように領域を定めた場合を考える. 通常であれば境界領域の粒子密度関数 f は境界部の格子数分のデータ通信を行う. しかし, 並進ステップに必要なデータは流域部分の格子の粒子密度関数 f のみであり, 陸の格子の粒子密度関数は不必要である. この状況を活用して, 通信する範囲を流域部分のみに限定することで, 通信量を削減する.

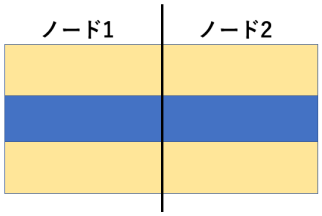

図 5 計算する領域

4.3 基本形状テンプレート法

並列化する際には, 大きな流域を複数に分割する. ここで各ノードでの通信位置について考える. 流入・流出方向は

1 方向の流れでもいくつか存在する. その他にも合流や分流など 2 方向・3 方向への流入・流出を合わせると多くの流入・流出パターンがあり,1 つ 1 つの通信を設定するのは非効率的である. そこで, 通信位置の自動設定化に向けて, 流入・流出方向の種類をパターン化し, それらの組み合わせにより, 流域形状に応じたプログラムの自動生成を行う. 流入・流出方向のパターン化には, 河口を起点とする 2 分木構成を考えればよいので,4 パターンの基本形を用意する (図 6).

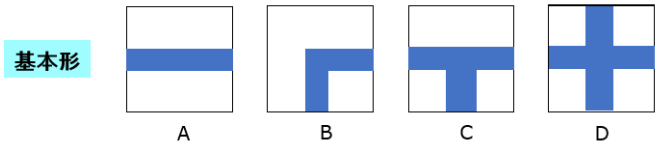

図 6 基本形 4 パターン

これらの基本形に対して座標変換を行うことで河川形状に適合したすべての流入・流出パターンを網羅できる. 例えば, 図 6 の B ように右から流入し, 下から流出するものを基本形とする. これを座標変換することによって, 下から左, 下から右など, 隣の方向への流入・流出方向を決定させる. このような座標変換パターンをテキストファイルから入力として与えることで, ユーザが簡単に川の形状を構成することができる. このパターン化を基本形状テンプレート法と呼ぶ.

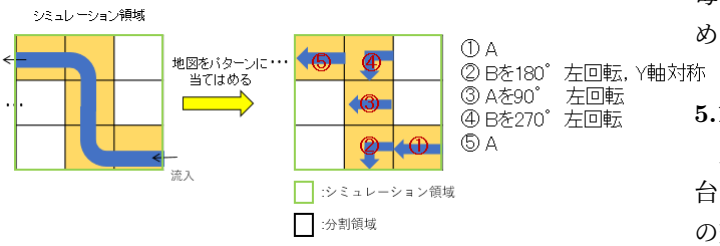

図 7 領域の座標変換例

また, シミュレーション領域の座標変換に伴い, 格子ボルツマン法の粒子密度関数 f も座標変換に合わせて変換させる. 例えば, 図 7 の領域③を計算する場合, 計算する時には流体の流れが左向きであるが, 実際は上向きである. 粒子密度関数を座標変換と同じように座標変換させて通信を行うことで正しく計算することができる. この変換は送信側と受信側どちらも行う必要があり, 図 8 は送信側の変換を図として表している. また,GPU は分岐命令を好まないため, 計算の性質上この変換は境界データ送受信のタイミングで CPU で行う.

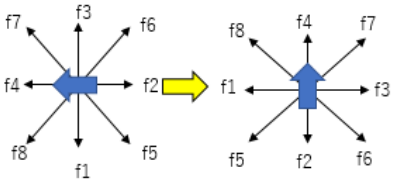

図 8 確率密度変数の変換

4.4 GPU の利用による計算の高速化

シミュレーションの流体計算に用いる格子ボルツマン法は, タイムステップごとにそれぞれの格子の計算は独立であり, 並列化が容易である。このことを利用して GPU の利用による高速化を行った.GPU を利用する際には,GPU で使用するためのデータを CPU からコピーしてくる必要がある. このデータコピーがボトルネックとなり,GPU の性能を最大限に引き出すことができない. これに対し OpenACC の Data 構文を用いて GPU 内のデータを再利用することで冗長なデータコピーの頻度を削減する.

5. 実機検証・考察

これまでの連成シミュレーションを実装し, 高速化・並列化に関する検証を行った. 並列化については, 並列化に伴って必要となった通信時間の測定と, 河川形状を考慮した通信量削減効果についての測定を CPU で実装して行った. 次に GPU 計算の実装を行い,1 ノードあたりの高速化の効果を確認した. また, 計算領域として, 福井県三国港付近の河川領域を利用した. 本シミュレーションの拡散計算は, 流体計算の巨視的な値を用いるため流体計算 N 回 (例:N=25) 毎に計算する. そのため, 計算の大部分は流体計算となるため, 時間の測定は流体計算を対象に行った.

5.1 並列化についての検証

並列化による高速化の効果として, 同じ領域を計算機 1 台で計算した場合と, 計算機 2 台で計算した場合の高速化の効果について検証した. ここでは並列化に伴い, 境界部分の確率密度関数の交換が必要となり,MPI による通信時間がかかってくる. この通信時間がシミュレーションの高速化の妨げとなる可能性があり, どの程度の時間がかかるのかを知っておく必要があったため, 通信時間に限定して時間を計測した. 使用した計算機 2 つの CPU は表 1 の通りである.

5.1.1 並列化による高速化の効果

CPU 版の計算量は, 計算する水域の大きさに依存するため, 領域の大きさではなく水域面積のバランスを取って計測を行う. 同等の水域面積を持った 800x1024 の領域 2 つを用意し, それらを計算機 1 台で計算する場合 (1600x1024) と, 計算機 2 台 (800x1024x2) を用いて計算する場合の 1 タイムステップあたりにかかる時間を計測した.

表 1 実装環境

	計算機 1	計算機 2
OS	CentOS 6.6	CentOS 6.4
プロセッサ	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz	Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
コア数	4	4
プロセッサ数	8	8
キャッシュメモリ	8192KB	8192KB
コンパイラ	gcc4.4.7	gcc4.4.7
MPI	MPICH2-1.1	MPICH2-1.1

表 2 並列化による高速化の効果

	1 タイムステップあたりの時間 [ms]
1 台	96.8
2 台	52.2

表 2 より, 計算機 2 台に並列化したことで約 1.9 倍の高速化が得られた. 2 倍でなかったのは, 並列化に伴って必要となった通信時間に起因すると考えられる.

5.1.2 ノード間の通信にかかる時間の測定

ここでは通信時間に限定して測定を行う. ここで, 通信するデータ量は float 型の大きさ 1024x9 の 2 次元配列である. 通信時間は相互結合網の構造や, 計算システムの規模にも依存するが, 通信するべき領域は境界部分のみであり, 隣の領域を計算しているノードに限られるため, 通信する計算機同士の距離は近く設定できると仮定して, 同じネットワーク内で, 距離の近い状況で測定した. 通信時間として, 通信する前後の座標変換を行う時間も含めた時間を計測した.

表 3 ノード間通信にかかる平均時間

	通信時間 [ms]
基本形状テンプレート法導入前	2.30
基本形状テンプレート法導入後	5.68

表 3 より基本形状テンプレート法を導入することで,3ms 程度時間がかかることがわかった.

5.1.3 河川形状を考慮した通信量削減の効果

川の形状に合わせて通信量を削減する実験を行った. 今回, この検証に関して流域分割のパターン化までの実装には至らなかったため, 基本形状テンプレート法は導入していない状況での計測となる. 今までの実験と同様に境界部分の解像度が 1024 の領域を使用して, 通信時間を測定した. 通常であれば境界部分は 1024 の解像度を持っており, データ量は 1024x9 である. 今回通信量を削減した結果, 川の領域のみを抽出することにより, データ量を 62x9 まで削減した. このときの通信時間を測定した結果が表 4 である.

表 4 河川形状を考慮した通信量削減の効果

通信量削減の有無	1step あたりの通信時間 [ms]
有	2.30
無	2.20

今回, 通信量削減の効果は 0.1ms であった. シミュレーションにかかる時間全体から見るととても小さく, 期待した効果は得られなかった. しかし, 今回の検証には基本形状テンプレート法の導入による座標変換の処理は入っておらず, この処理を行った場合, 通信のための計算時間も短縮されると考えられるため, この検証で確認できた, 時間短縮効果よりは大きくなると推測できる.

5.2 GPU 計算についての検証

CPU-GPU 間の最適化の効果の検証と,GPU 利用による高速化の効果,CPU-GPU 間のデータコピーにかかる時間について検証を行った. GPU の利用に使用した計算機は表 5 のとおりである.

表 5 使用する GPU

GPU プロセッサ	GeForce GTX 780 Ti
コンパイラ	pgcc(PGI)13.6-0
コア数	2880
メモリ	3072 MB

5.2.1 GPU 内データ再利用の効果

GPU 内データ再利用の効果について検証を行った.CPU-GPU 間の通信が必要なのは, シミュレーション開始時と, 並列化に伴う境界部分を行う時, 可視化ノードにデータを送信する時に限られる. それ以外では GPU 内にあるデータを再利用した. データを再利用する場合としない場合での流体計算 1 タイムステップあたりの時間を比較した.

表 6 データ再利用の効果

GPU 内データ再利用	25step 実行時間 [ms]
なし	100.8
あり	3.7

5.2.2 GPU の高速化効果

CPU のみを使って計算した場合と GPU を利用して計算した場合との比較を行った.CPU での計算は OpenMP を用いて 4 コアで計算させた場合である.1 ノードあたりの計算時間を比較した. 流体計算 1 タイムステップあたりの計算時間を測定した結果が表 7 である.

表 7 GPU よる高速化

計算ユニット	1step 実行時間 [ms]
CPU	96.8
GPU	3.7

GPU を利用することにより,CPU のみで計算した場合に比べて約 25 倍の高速化が得られた.

冗長な CPU-GPU 間の通信を削減し,GPU 内にあるデータを再利用することで約 27 倍の高速化が得られた.

5.2.3 CPU-GPU 間の通信時間の測定

CPU-GPU 間の通信にかかる時間を測定した.float 型の大きさ 1024x9,2048x9,4096x9 の 3 つの配列を用意し,CPU から GPU への転送と GPU から CPU への転送時間を測定した結果が表 8 である.

表 8 CPU-GPU 間の通信時間

DataSize	CPUtoGPU[ms]	GPUtoCPU[ms]
1024	0.124	0.088
2048	0.126	0.091
4096	0.134	0.095

表 8 より, データサイズ 1024x9 のとき,CPU-GPU 間にかかる時間は, 合計で 0.21ms ほどであった.2048x9・4096x9 とデータサイズを倍に増やしていくと, 数 ms だけ通信時間は伸びるが, 全体に対して 1%ほどの増加であった. このことから CPU-GPU 間の通信時間はデータサイズにはあまり依存せず, 通信頻度に依存して増加していく傾向にあると推測できる. したがって, 並列化の際に 1step に必要なデータをすべて通信するという手法を用いたのは有効であったのではないかと推測できる.

5.3 考察

表 7 より, 今回使用した CPU と GPU では約 25 倍の計算性能の差があることが分かった. また表 3 より, 通信時間の平均は約 5.8ms であり, これは GPU の計算時間よりも長い結果となった. このことから, 計算のノード間並列化と GPU 利用を併用して実装した場合, 計算よりも通信に時間がかかってしまい並列化による高速化は得られない結果となった.

しかし, 今後の展開としてシミュレーションの 3 次元化を目指している.3 次元化することによってシミュレーションの精度向上が見込めるが, 計算量が更に増大する.3 次元化によって計算量が増大すれば,GPU を利用した場合においても, 通信時間よりも計算時間の割合が高くなり, 並列化によって得られる恩恵は増加するであろうと推測できる. また,3 次元化した場合でも, 川は深さ方向にはつながっていかないため, 基本形状テンプレート法も 3 次元シミュレーションに適用可能である. 更に,3 次元化に伴って, ノード間の通信量も増加する. 川の形状に着目した通信量の削減の効果も通信量が増えれば増えるほど高まるのではないかと推測できる. このように, これまでで行った実装が 3 次元化に向けた有効な実装であったと言えるであろう.

6. おわりに

本研究では, 河川を流れる水の流体計算を行い, その結果を用いた拡散計算を行う連成シミュレーションを作成した.

広大な領域を計算するための必要な計算資源を確保するため, 複数ノードを用いた並列化と並列化に必要な通信パターンに合わせた領域の座標変換を行う基本形状テンプレート法を導入することにより, 本シミュレーションが高並列度にも対応できるようにした. また, 実時間指向を持ったシミュレーションとするため,GPU による高速化を行い,1 ノードあたりの計算性能を向上させることで, 実時間性を向上させた.

通信時間の測定を行い, ノード間通信にかかる時間と,CPU-GPU 間の通信にかかる時間を測定した. その結果, 現状では GPU の計算時間と基本形状テンプレート法を用いた場合の通信時間を比較すると, 通信時間のほうが長いという結果になった. しかし,1 台あたりの計算量を増やしていくことにより, 通信よりも計算のほうが時間がかかる状況になれば, 並列化による高速化の効果は高まっていくと推測できることに加え,GPU のメモリは限りがあるため, 並列化は必要であるといえる. また, 河川形状を考慮した通信量削減を行った. 実機で検証したところ, ほとんど効果は得られなかったが, この検証には基本形状テンプレート法を導入しておらず, 今後更なる検証が必要である.

領域分割による並列化, 基本形状テンプレート法の導入, 河川形状を考慮した通信量削減,GPU の利用による高速化等の実装を行ってきた. 今後の展望としてはシミュレーションの 3 次元化を目指しており, これまで行った実装は 3 次元化においても有効であるであろう.

参考文献

[1] 日吉莉菜:”動的オープンデータと連携した流体・拡散シミュレータの開発” 福井大学工学部情報・メディア工学科 卒業論文 (2017)
[2] ZHOU Rui, ZHANG Jiaochao, FUKUMA Shinji, MORI Shin-ichiro： A case study: Development of a time-critical IoT system for environment sensing (2017-09-11)
[3] 山口明徳: ”インタラクティブ流体シミュレーションにおける力覚提示の実装 ” 京都大学工学部情報学科 卒業論文 (2007)
[4] 山田航平: ”広域分散型協調シミュレーションのためのシミュレーションキャッシングフレームワークの実装と双方向一貫性制御方式の検討 ” 福井大学工学部情報・メディア工学科 卒業論文 (2016)
[5] 福井県 河川・砂防総合情報 http://ame.pref.fukui.jp(2019/2/5)
[6] OpenACCProgramming https://www.softek.co.jp(2019/2/5)