土木学会論文集 A2(応用力学), Vol. 69, No. 2 (応用力学論文集 Vol. 16), I_95-I_105, 2013.

動的領域分割および複数GPUを用いたMPS粒子法の高速化

Accerelation of the moving paricle semi-implicit method through multi-GPU parallel computing with dynamic domain decomposition

曽田康秀*・渡邊明英**・小島 崇***
Yasuhide SOTA , Akihide WATANABE and Takashi KOJIMA

*理博　株式会社　東京建設コンサルタント（〒 170-0004 東京都豊島区北大塚 1 丁目 15 番 6 号 ）
**正会員　工博　株式会社　東京建設コンサルタント（ 〒 170-0004 東京都豊島区北大塚 1 丁目 15 番 6 号）
***正会員　工博　株式会社　東京建設コンサルタント（ 〒 170-0004 東京都豊島区北大塚 1 丁目 15 番 6 号）

In this paper, we propose the dynamic domain decomposition for particle methods, in which the full particles domain is decomposed into several sub-particle domains according to their initial positions. In the dynamic domain decomposition, the exchange of the particle data between neighboring sub-particle domains are effectively reduced, since each sub-domain moves along the particle motions. In addition, when the portion of the boundary particle domain to the full particle domain is small, the accerelation of numerical simulations can be achieved through the parallel computing by using the libraries such as MPIs. In this paper, we apply the method of dynamic domain decomposition to the multi-GPU parallel computing and investigate the efficiency of the method for the numerical accelerations. In fact, we will demonstrate that in dam break simulations, where the balk of water is moved from one side of the wall to another, the acceleration is achieved by reducing the ratio between the volume of the boundary particle domain and that of the full particle domain.

Key Words : *Particle Methods, multi-GPU, domain decomposition*

1. はじめに

2011 年 3 月の東北地方太平洋沖地震では, 地盤沈下やダムの決壊などさまざまな被害がもたらされた. それに伴い今後の巨大地震に備え, 地盤変化を考慮したより大規模で精密な 3 次元流体計算が必要になると予想される [1)]. 越塚らによって提案されたＭＰＳ粒子法[2),3)]は, 格子法と違って境界格子を設定する必要がないことから, 津波の陸地構造物への作用など境界が複雑な場合の流体計算に対して有効であることが指摘されている. 一方で, 粒子法では球形の等方粒子を等間隔で配置するため空間方向によって解像度を変えることができず, 格子法に比べて計算コストが掛かるという問題がある.

CPU による大規模数値計算の高速化手法として, 計算領域を空間分割しそれぞれの領域をＭＰＩ[4)]などを用いて複数 PC(複数ノード）で並列計算する手法が用いられている. この手法はＭＰＳ粒子法を用いた 3 次元津波解析に適用されており, １ｋｍ× 600 ｍ×高さ 30 ｍの範囲に１m間隔で粒子を置いた場合, ノード数４, コア数４のＣＰＵ計算機環境で, 実現象 60 秒に対して２３７時間の計算時間を要している[5)]. したがって, 粒子法を用いた 3 次元津波解析を津波予測などの実務に応用するには, さらなる高速化が必要である.

近年, ビデオカードによる高速画像処理技術を応用した GPGPU(General-purpose computing on graphics processing units) 計算は, 複数コアによる超並列計算およびメモリ帯域の大きさの点から, CPU 計算に変わる高速な数値計算の手法として注目されている[6),7)]. 特に 1 ノードでの数値計算において,CPU での共有メモリ型並列計算でデータ量が Cash 容量を上回りメモリ律速になる場合には,GPU 計算は CPU 計算に比べて優位になる[8)]. 半陰 MPS 粒子法の GPGPU 数値計算では, 圧力ポアソン方程式解法の部分がボトルネックになるが, これまでメモリアクセス量の低減化やテクスチャメモリの利用による高速化が報告されている [9),10)]. また NVIDIA 社の CUDA コンパイラでは,CUDA 4.0 で, ホスト側の一つのスレッド上で, 複数のデバイスを切り替えて使用できるようになった. したがって,1 ノード複数デバイスによる領域分割では, ホスト側で MPI などを使って複数のプロセスを立ち上げる必要がなくなった. このことは, 簡単なプログラムの変更により, １ノード複数 GPU 計算が可能になることを意味する. 従って複数デバイスを用いた領域分割法は, 今後 GPGPU 計算における高速化に対して, 有効な手段になると考えられる.

半陰 MPS 粒子法の高速化でも, 分割領域のそれぞれに一つのビデオカードデバイスを割り当てて並列計算を行う複数ＧＰＵ計算が適用されている[11)]. しかしながら, 空間固定された静的領域分割では, 流体全体の変化が激しい場合にデータ転送における通信コストが大きくなる. 一方, 流体移動が激しい場合に, 流体の動きに合わせて分割領域を変化させる動的領域分割も提

案されている[12]. しかしながら, 個々の分割領域内の粒子数に隔たりが生じた時点で領域を動かす方法では, 計算時間のかなりの部分で粒子数の不均一化が生じてしまい, 計算ロスが生じると考えられる.

本研究では, 以上の点を踏まえて, ダム崩壊のような流体全体が移動するような場合でも有効な, 粒子群の動きに合わせた動的な領域分割法を提案する. またそれを 1 ノードでの粒子法の複数ＧＰＵ計算に適用し, 全計算領域に対する境界断面の割合が小さい大規模流体数値計算において, 半陰 MPS 粒子法での圧力方程式解法の高速化に有効であることを示す.

2. 領域分割法の粒子法への適用

越塚らにより提案された半陰 MPS 粒子法は, これまでにさまざまな高精度化の改良が行われているが, 以下では, 越塚らのオリジナルなスキームを用い, 半陰 MPS 粒子法の高速化について考察する.

一般にオイラー法での領域分割法では, 計算領域を幾つかの小領域に分割した後, それぞれの領域で物理量データのメモリ確保を行い, それらをそれぞれ指定されたデバイスに格納する. 1 ステップでの計算は, それぞれのデバイス上で独立に行うが, 境界での物理量の計算は, 各デバイス上の物理量のみでなく境界で接した他デバイス上の物理量を必要とする. これらの手続きは粒子法における領域分割でも同様である[11]. 具体的な半陰 MPS 粒子法 1 ステップの流れ**図-1** を見ると, デバイス間で, データのやり取りを必要とするのは,

I 粒子の位置情報による, 各領域上での境界粒子の選定

II 隣接領域からの物理量データ転送.

III 圧力ポアソン方程式解法における行列ベクトル積計算

の 3 箇所である. 以下では, これらのデータ転送手続きについてまとめておく.

2.1 粒子の位置情報による境界粒子の選定

一般的な領域分割法では, 各領域において境界領域を選定し, 境界領域間で物理量の転送を行う必要がある.

境界粒子の選定は, フローチャート**図-1** に記述されている近傍粒子登録の箇所で行われる. 近傍粒子登録は粒子位置が更新されるごとに必要なので, ステップごとに 2 度行う必要がある. このそれぞれの登録の際に, 後述する方法で境界粒子を選定する. この際, 選定された境界粒子の物理量データを相手領域へのコピーしておく. 境界粒子選定およびデバイス間データ転送の具体的な手続きは以下のようになる.

i 領域 A のデータ領域をデバイス A 上のメモリ領域に, 領域 B のデータ領域をデバイス B 上のメモリ領域に確保する . ただしそれぞれのメモリ領域は, 境界で接する相手方のデータ領域を格納するため, 余分に確保しておく (**図-2(a)**).

ii A,B それぞれの領域で境界を選定し, そのメモリ領域を確保する (**図-2(ｂ)**).

iii A,B それぞれの領域で境界メモリ上のデータを相手デバイス上のメモリ領域へコピーする (**図-2(ｃ)**).

以下では, 各メモリ領域内の境界領域に存在する粒子を境界粒子, 境界粒子を相手デバイスにコピーしたものをゴースト粒子と呼ぶ.

2.2 隣接領域からの物理量データ転送

I の境界粒子の選定およびその物理量の相手デバイスへのデータ転送手続きが完了した後, 境界付近には境界粒子と相手デバイスからのゴースト粒子が共存している. 例えば, デバイス A のメモリ領域には,A の境界粒子に加えて領域 B のゴースト粒子が存在している. したがって, デバイス A では, 境界粒子 A の近傍粒子の完全な情報が得られる. 一方, 境界粒子 B の近傍には領域 B の非境界粒子も存在するが, デバイス A にはそれらの粒子が存在しない. したがって, デバイス A における領域 B のゴースト粒子の近傍粒子情報は不完全であり, 物理量の正しい値は得られない.

デバイス A および B で, 近傍粒子の情報から物理量をもとめた後, さらにそれらの物理量を用いて境界粒子 A の物理量をもとめるためには, 上述した物理量のデータ転送手続き iii を行い, デバイス B から正確な値を持つ境界粒子 B の物理量をコピー取得する必要がある. 近傍粒子登録以外で II の隣接デバイスからの物理量データ取得が必要なのは,

a Explicite stage(**図-1**) 内の, 粒子の位置と速度の計算

（粒子数密度および粘性項の計算のため）

b Final stage(**図-1**) 内の, 粒子の圧力計算

（近傍最小圧力計算のため）

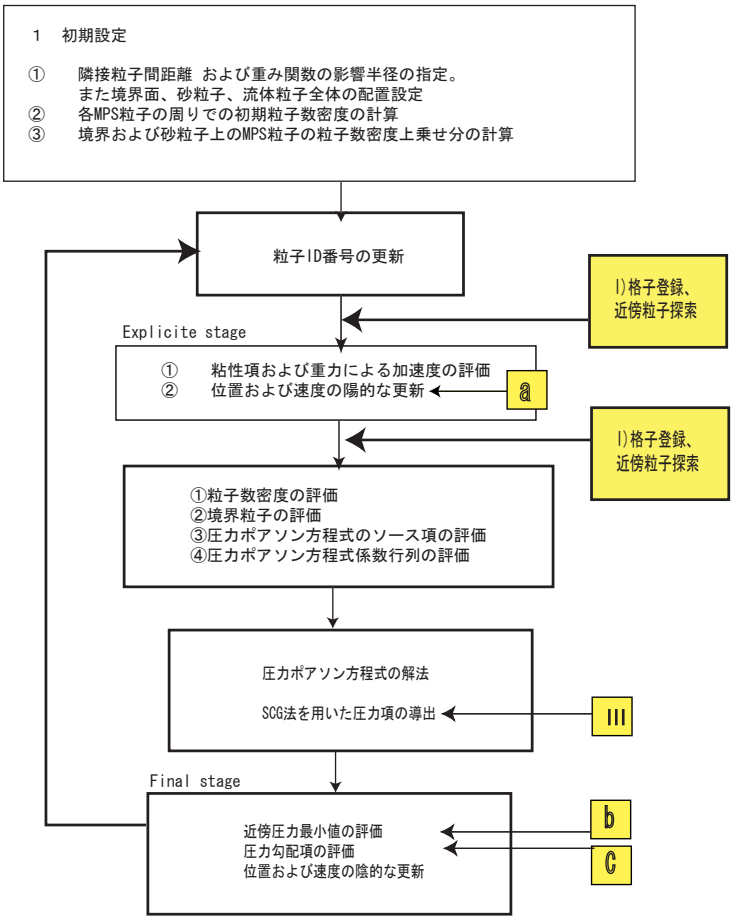
c Final stage(**図-1**) 内の, 粒子の近傍最小圧力計算

（圧力勾配項計算のため）

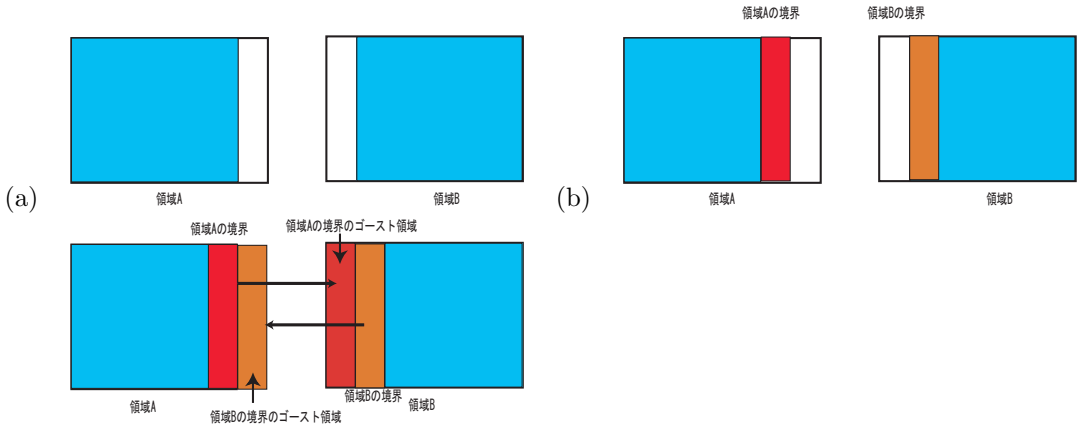
の 3 箇所である.

2.3 圧力ポアソン方程式解法における行列ベクトル積計算

MPS 粒子法における代数方程式 $W\vec{x} = \vec{b}$ の高速解法としては, これまで ICCG 法が標準的に用いられて



図–1　半陰 MPS 粒子法における計算過程のフローチャート. 黄色部分で, 領域間データ転送を行う.



(c)

図–2　計算領域の 2 領域への分割とゴースト境界領域の生成.(a) 領域 A での粒子 A のメモリ領域（青左）および領域 B での粒子 B のメモリ領域（青右） j の生成 (b) 領域 A および B での境界の選定およびそのメモリ領域（赤および茶）の確保.　(c) 領域 A および B の境界の相手デバイスへのコピー（相手デバイス上でのゴースト領域の生成）.

いる[2]. しかしながら,ICCG 法では, 前処理の Cholesky 分解（または修正 Cholesky 分解）の手続きおよび, 演算の一部に前進後退代入を含んでいるため, 直接並列化を行うことができない. これに対し, 速水, 原田によって提案された SCG（Scaled CG）法[16]では, 行列 W の対

角項を $D = \mathrm{diag}\,[W_{11}, W_{22}, \ldots, W_{NN}]$ として, 変換式

$$\begin{aligned} W' &\equiv D^{-1/2} W D^{-1/2} \\ \vec{x}' &\equiv D^{1/2}\vec{x} \\ \vec{b}' &\equiv D^{-1/2}\vec{b} \end{aligned} \tag{1}$$

を前処理として用いるため, 直接並列処理が可能となる. 以下では, 越塚らによって提案されたオリジナルな形の圧力ポアソン方程式

$$\begin{aligned} &\frac{d}{n_0}\sum_{j\neq i}^{N}\left[\frac{P_j - P_i}{|\vec{r}_j - \vec{r}_i|^2}\left(\vec{r}_j - \vec{r}_i\right)\omega\left(|\vec{r}_j - \vec{r}_i|, h\right)\right] \\ &= -\frac{\rho_0}{\Delta t^2}\left(\frac{n\left(\vec{r}_i\right) - n_0}{n_0}\right) \end{aligned} \tag{2}$$

$(i = 1, 2, \ldots)$ に対して,SCG 法を用いる. ただし ω は重み関数,$\vec{r}_i$ は i 番目粒子の位置ベクトル,d は空間次元,h は影響半径,n_0 は平均粒子数密度を表す. この圧力方程式に対して, 行列 W の i, j 成分は

$$W_{ij} \equiv \frac{d}{n_0}\frac{\left(\vec{r}_j - \vec{r}_i\right)}{|\vec{r}_j - \vec{r}_i|^2}\omega\left(|\vec{r}_j - \vec{r}_i|, h\right) \tag{3}$$

と表される. SCG 法の利用に際して, 複数デバイス使用の場合は, 以下で示すように, データ転送に注意が必要である.

SCG 法の過程の中で, 行列 W とベクトル $\vec{\xi}_k$ の積, $W\vec{\xi}_k$ を計算する必要がある. 圧力ポアソン方程式では, 行列 (3) の非対角成分には, 粒子 i とその近傍粒子 j との相対座標ベクトルが関係してくる. デバイス A,B における $\vec{\xi}_k$ をそれぞれ $\vec{\xi}_A^{(k)}$, $\vec{\xi}_B^{(k)}$ とすると, デバイス A 上の元境界粒子 i に対する行列ベクトル積は以下のように元粒子成分との内積とゴースト粒子成分との内積とに分解できる.

$$\sum_{j\neq i}^{N} W_{ij}\xi_{A,j}^{(k)} = \sum_{j\in devA} W_{ij}\xi_{A,j}^{(k)} + \sum_{j\in GhostB} W_{ij}\xi_{A,j}^{(k)} \tag{4}$$

式 (4) のうち, 右辺第 2 項は不完全であるが, 行列 W の Ghost 粒子成分に関しては, あらかじめ代数解法を行う前に, デバイス間転送で元粒子成分に置換しておくことによって完全にできる. 一方, 式 (4) の右辺第 2 項のゴースト成分 $\vec{\xi}_A^{(k)}$ は, 代数方程式解法の繰り返しごとに不完全になるので, 繰り返しの度に元粒子の値に置換する必要がある.

3. 粒子法への動的領域分割法の適用

以上の手続きは, 格子法での固定領域分割でも粒子を格子点に置き換えれば同様である. 一方, 通常の格子法領域分割では境界点が固定されるのに対し, 動的領域分割法では, 各領域の境界に属する粒子は各ステップごとに変動する. したがって, ステップごとにどの粒子が境界に属するのかを判定する手続きが必要である. 空間に固定された領域分割では, 隣接する境界領域の近傍に粒子が属するかによって, 境界粒子が決定される[11]. 一方, 我々が提案する動的領域分割法では境界領域を決める必要がない. 単に他デバイスに属する粒子が近傍にいるか否かによって境界粒子の判定がなされる. 以下では, 境界粒子の判定を近傍粒子探索に用いる格子点を用いて行う.2 領域分割での具体的な手続きは以下のとおりである.

i 粒子群を初期配置にあわせて, **図-3** のように, 粒子群 A と粒子群 B に分割し, それぞれの物理量をそれぞれ別々のデバイス A, デバイス B のメモリ上に確保する. ただし, オイラー法領域分割と同様, 相手粒子群からの境界粒子をコピーするための領域を余分に確保しておく.

ii 一般に, 粒子法では計算領域を影響半径を一辺の長さとする格子点の集合 に分割して各粒子の格子座標をもとめ, 隣接格子上の粒子を探索する. この格子分割は, 境界粒子探索においても有効である. 以下では, 粒子群 A および粒子群 B に属する粒子の存在する格子座標から, 境界粒子の存在する格子座標をもとめる（**図-3(a),(b)**).

iii 格子点集合から, 粒子群 A の粒子の格子番号を含む格子点の集合 L_A と粒子群 B の粒子の格子番号を含む格子点の集合 L_B をもとめる.

iv 格子点集合 $L_k \quad (k = A, B)$ （**図-3(c),(d)** のオレンジ部分）のそれぞれに対し, L_k 内格子点に隣接する格子点を加えた格子点集合 $\widehat{L}_k \quad (k = A, B)$ （ L_k の閉包）をもとめる.（**図-3(c),(d)** で, $\widehat{L}_k$ は（オレンジ部分に黄色部分を加えた領域） . $\widehat{L}_A$ に属する粒子群 B の粒子を, 粒子群 A の境界粒子, $\widehat{L}_B$ に属する粒子群 A の粒子を, 粒子群 B の境界粒子と定義する. オイラー手法と同様, 境界粒子群に対するメモリ領域を, 境界を構成する 2 つのデバイスの両方に確保する. 次に, 各デバイスの元領域から, 同デバイス上の境界粒子群へのメモリーコピーを行い, それをさらに異デバイス上へメモリコピーし, ゴースト粒子を生成する.

粒子法では, 粒子群 A と粒子群 B の境界は, 各ステップ後に変化していく. したがって, 上の境界粒子の選定は, 各ステップで粒子位置が変化するごとに行う必要がある. 以上の手続きは, 領域が 3 つ以上の場合でも同様である. また上の定義から, 2 つの粒子群が混ざり合う場合も含めて, 一般的に各粒子群に対する境界粒子群を一意的に定義することが可能である. したがって, 以下で示すように, ダム崩壊のような流体全体が激しく運動する場合にも, 境界粒子を定義することは可能である.

計算コードでは, 近傍粒子探索としてまず各粒子の位置情報から格子番号登録を行い, 次に各粒子に対して, 近傍格子のサーチを行い, 近傍格子内で影響半径内にあ

る粒子の粒子番号を粒子数掛ける最大近傍粒子数のサイズの配列に登録する． 以下の GPGPU 計算では, 各スレッドを各粒子の粒子番号に割り当てて, 並列計算を実行する.

4. 動的領域分割法の複数 GPU 計算の適用と粒子法の高速化

NVIDIA 社のビデオカードを用いた GPGPU 計算では,CUDA4.0 から単一ＰＣに装填された複数のビデオカードの間で PCI スロットを介したデータアクセスが可能になった. したがって, 上で示した領域分割法における各領域をそれぞれ異なるデバイスに割り当てて, 並列計算を行うことが可能になる. 以下では, この動的領域分割を,1 ノードでの複数 GPU を用いた MPS 粒子法に適用し, その有効性について検証する.

複数 GPU を用いた領域分割法では, 各分割領域に属する物理量をそれぞれ異なるデバイス上のグローバルメモリに格納し, そのデバイス上で計算する. 一般に CUDA コンパイラでデフォルトのホストデバイス間メモリ転送を行う際には, デバイスホスト間で同期が取られる. したがって複数デバイスによる GPU 計算では,1 デバイスごとに同期を取ると, 転送時間の効率が悪くなる. ＣＵＤＡコンパイラによる複数デバイス計算では, メモリ転送の際に, 非同期の関数を用いることも可能である[6),7)]. またデバイスが３つ以上では,Left-Right アプローチ[13)]を採用すると効率がよくなる. このアプローチでは,A,B,C ３つのデバイス間非同期転送をの順にデータ転送を行う代わりに, （right 転送）を行った後に (left 転送) を行う. 以下の３ＧＰＵ並列計算では, この順序でメモリ転送を行う． これ以外にもＧＰＧＰＵ計算では, アーキテクチャの違いから, ＣＰＵ計算とは異なる計算コードの高速化が必要になる.

4.1 境界粒子のリセットによる複数ＧＰＵ計算の高速化

一般に, 動的領域分割では, 境界の粒子が混合されていく. 本研究での動的領域分割法では, 境界粒子の判定は, 近傍格子に他領域上の粒子が存在するか否かによって判定される. 従って, 粒子どうしの混合が進むほど, 境界粒子の数が増大し, 境界における粒子転送のコストが増大していく. これを回避するためには, 一定時間が経過した後, 定期的に, 粒子の配置をリセットする必要がある. 粒子のリセットについては, 以下の手続きを踏む.

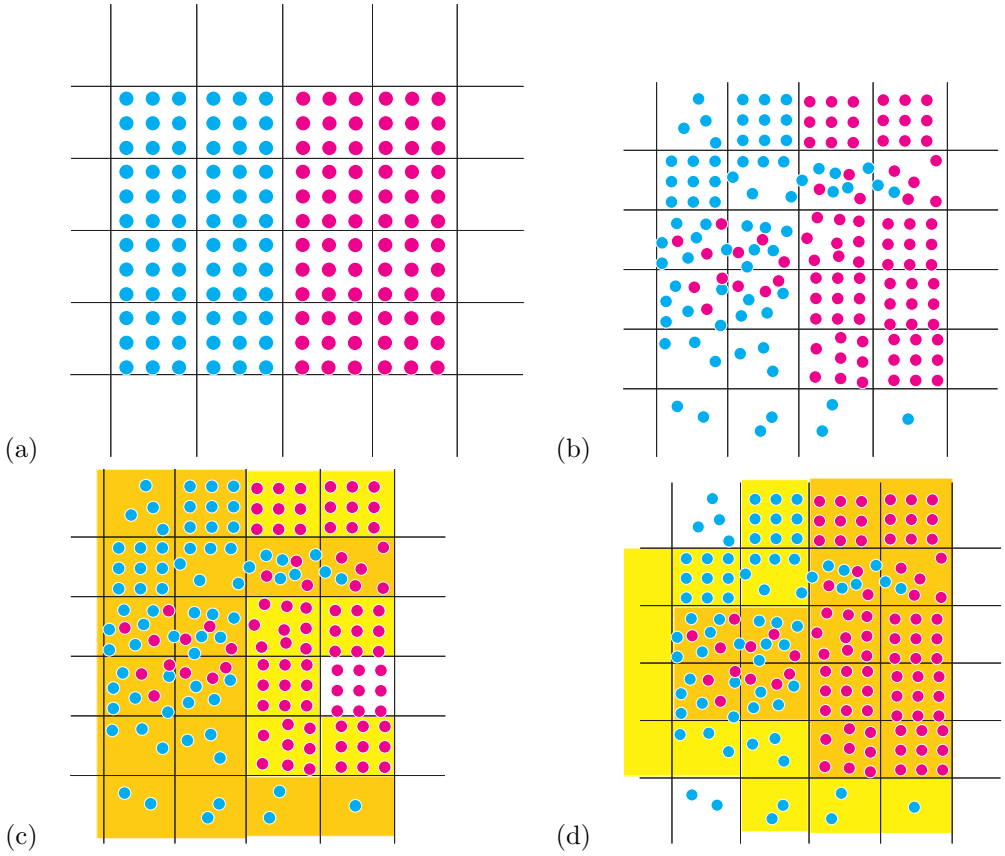
1. 時刻 t で領域 A および領域 B に属する境界粒子の個数 $N_{(A,B)}$ および $N_{(B,A)}$ をそれぞれもとめる
2. 各時刻での境界領域の境界面の法線ベクトルを $\vec{n}_b$ として, 各境界粒子の位置ベクトル $\vec{r}$ の $\vec{n}_b$ 方向成分 $(\vec{n}_b \cdot \vec{r})$ をもとめる.
3. 各粒子の $(\vec{n}_b \cdot \vec{r})$ 成分をソートし, 小さい順から, $N_{(A,B)}$ 個までを, 粒子群 A に, $N_{(B,A)}$ 個を粒子群の空きメモリに登録する.

以上の手続きでは, リセットに際して境界面に垂直な方向 $\vec{n}_b$ をもとめる必要がある. 以下のダム崩壊実験のように流体の運動が一方向の場合には, $\vec{n}_b$ は初期の境界面に垂直な方向に取ることができる. 一方, より一般的な場合は, 領域 A,B の境界領域のそれぞれの重心を結ぶベクトルなどで, $\vec{n}_b$ を定義する必要がある. この方向が決まれば, この方向成分で粒子をソートして, 粒子を領域 A に近い側と遠い側に 2 分して境界粒子を振り分ければよい. 以上の粒子登録リセットの手続きを行う頻度は, 流体混合の激しさに依存する. 以下のダム崩壊の例では, ダムが崩壊して水塊が壁にぶつかるまでは, 境界粒子数はほぼ一定であるが, ぶつかった後は流体混合が起こり, 境界粒子数が急激に増加する． したがって壁衝突後は, 一定時間間隔で境界粒子をリセットする必要がある. 境界粒子数を各時刻で測定し, その変化率が閾値を越えるか否かで粒子リセットの実行を動的に判定することも可能である.

5. 数値計算テスト

以下では, 半陰 MPS 粒子法における複数 GPU 並列計算の有効性を確かめるために, ＭＰＳ粒子法計算の典型であるダム崩壊問題によるベンチマークテストを行う. 以下の数値計算では,MPS 粒子法は越塚氏のオリジナルなものを用い, 壁の層は, 圧力を計算する層を表面に１層, その背後に粒子数密度を保つための粒子層を４層配置した. ダム崩壊問題では, 水柱が崩壊するため, 水粒子全体の重心位置は, 計算の間大きく変化する. また壁に水塊が当たった直後の水面形状は複雑になる. よって, 空間に固定された領域によるオイラー的領域分割では, 領域ごとに粒子数の隔たりが生じ, 計算効率が悪くなることが予想される. 一方, 動的領域分割では,2 領域の境界は重心の移動に伴い変化するため, ダム崩壊のような計算にも有効であると期待される. 実際, 我々は境界断面での接触領域の全流体領域に占める割合が小さい場合には, 複数ＧＰＵ計算で計算の高速化が実現できることを示す. 以下では,Linux マシーンのＰＣＩスロットに NVIDIA 社の Kepler 世代のビデオカード Gefore GTX 660 Ti(以下 GTX660) 3 枚を搭載し, 最大３ＧＰＵまでの並列複数 GPU 計算を **表-1** のような計算環境で行った.

まず複数 GPU 計算を行う前に, １ノードでの共有メモリ型 CPU 並列計算と１ GPU 計算の計算性能を比較しておく. 例として MPS 粒子法陰解法１ GPU 計算で計算コストの大部分を占めた圧力ポアソン方程式解法での行列ベクトル積演算部分を比較する. 以下では Intel Core i7-3986X(3.30Ghz)(以下 Core i7）による



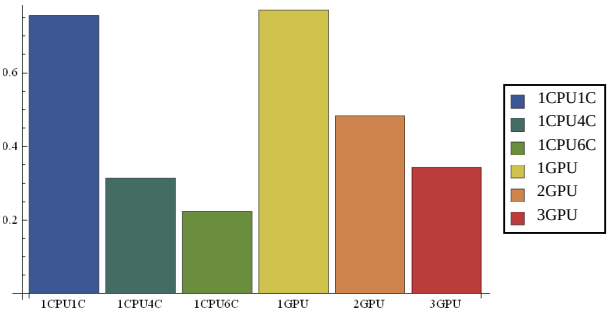
図–3 動的領域分割における格子座標分割および境界領域.(a) 初期分布での領域分割例（青：領域 A 内粒子, ピンク：領域 B 内粒子）,(b) 時間発展後の領域分割例（青：領域 A 内粒子, ピンク：領域 B 内粒子）,(c) 時間発展後の領域 A 内粒子を含む格子点集合 L_A(オレンジ）およびその閉包 (オレンジ＋黄色）.(d) 時間発展後の領域 B 内粒子を含む格子点集合 L_B(オレンジ）およびその閉包 (オレンジ＋黄色）. ただし実線が影響半径サイズでの格子分割を表している.

openMP 並列計算と NVIDIA 社の Kepler 世代のビデオカード Gefore GTX 660 を用いた複数 GPU 計算での比較を行った. ただし GPGPU 計算では, 粒子数 N_{par} に対し, スレッド数 512, ブロック数 $(int)(N_{par}/512)$ に固定し, テクスチャメモリやシェアードメモリの利用などのチューニングは行わず単純なグローバルメモリ上でのデータアクセスを行った. これらの計算環境で CPU 側は openMP よる並列計算を,GPU 側は複数 GPU による並列計算を行い, ダム崩壊初期分布に対する圧力ポアソン方程式ベクトル行列積計算 ((4) 式）での計算速度を比較した. **図-4** に見られるように,openMP 並列計算では,$N_{par} = 50000$ で複数コアによる計算速度の向上が見られる. メモリ律速の状態では, 複数コアによる加速率はデータ量によりほぼ一定となる. また, 1 GPU での計算速度は,Corei7 1 CPU 1 コアに対して 0.8 倍程度になった. 一般的には, メモリ律速の状態では,GPU のほうがメモリ帯域の点で 1 CPU 計算よりも速くなると予想されるが, 格子法と違い,(4) 式での行列, ベクトル積の各成分の粒子番号は固定されず, メモリアクセスに遅延が起こることが考えられる. 実際,((4) 式での行列成分またはベクトル成分を定数に置き換えると, CPU で計算速度が変わらないのに対し, GPU では数倍の計算速度向上が見られる. **図-4** で複数 GPU による加速は見られるが, 複数コアによる加速率ほどではない. したがって,GPU 計算が高速化の上で CPU 計算に対して優位性を保つには, より高速なメモリアクセスの手段を用いて 1 GPU での高速化を図る必要がある.

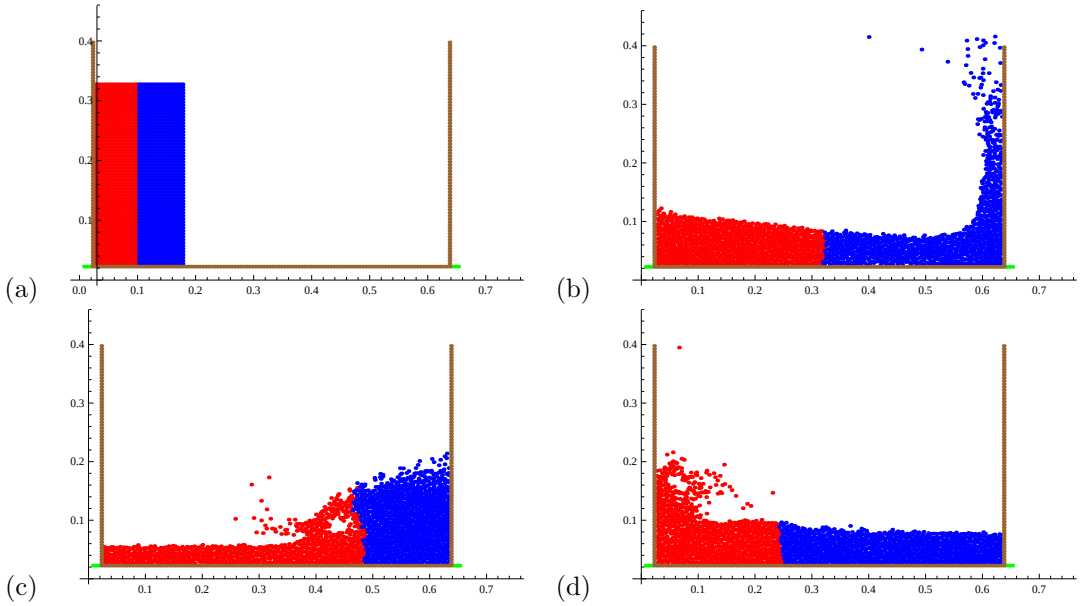
複数 GPU 計算では, 境界領域体積の全粒子領域体積に対する割合 ξ が小さいほど, 計算効率は良くなると考えられる. 以下の水柱崩壊数値実験の場合, x 方向の水柱の幅を L_x, x 軸に垂直な断面積を S とすると, 全粒子体積は L_xS となる. 一方境界領域体積は影響半径 h_p を用いて大雑把に h_pS のオーダーとなり, $\xi \sim h_p/L_x$ である. したがって, 全粒子体積を固定し初期粒子間距離 b を小さくして解像度を上げるか, b を固定し全粒子体積を大きくするほど, 複数 GPU での加速率は大きくなると考えられる.

5.1 ダム崩壊問題における動的領域分割の検証

次に, 流体が激しい運動する場合での動的領域分割の適用例として, ダム崩壊の数値計算を行う. ダム崩壊に関しては, 実験によって水柱崩壊での各時刻での先端位置のデータが得られており[17], 実験との比較で結果の正当性を検証することが可能である. 以下では, まず



図–4 ダム崩壊初期分布に対する圧力ポアソン方程式ベクトル行列積計算 ((4) 式) での CPU および GPU, 計算速度の比較. 縦軸は粒子数 $N_{par} = 50000$ に対する 100 ループでの累積計算時間 $[s]$. CPU 計算では,Intel Core i7-3986X(3.30Ghz) での openMP 並列計算を,GPU 計算では,Gefore GTX 660 Ti を用いた複数 GPU 計算を行った. 棒グラフは左から,1CPU1 コア,1CPU4 コア,1CPU6 コア, 1 GPU, 2 GPU, 3 GPU での計算時間 $[s]$ を表してる.



図–5 2 G P U計算の 2 次元ダム崩壊への適用例.(a)$t = 0[s]$,(b)$t = 0.4[s]$,(c)$t = 0.8[s]$,(d)$t = 1.2[s]$ でのスナップショット. 赤と青は各時刻でデバイス 1 とデバイス 2 に格納されている粒子をそれぞれ表している. 初期時刻では, 水柱の左側 (赤領域) と右側 (青領域) に粒子が分割されて 2 デバイスに格納されており, 時間発展するにつれて初期に分割された粒子は境界部分を除いて初期のデバイス領域に格納されたまま移動していく.

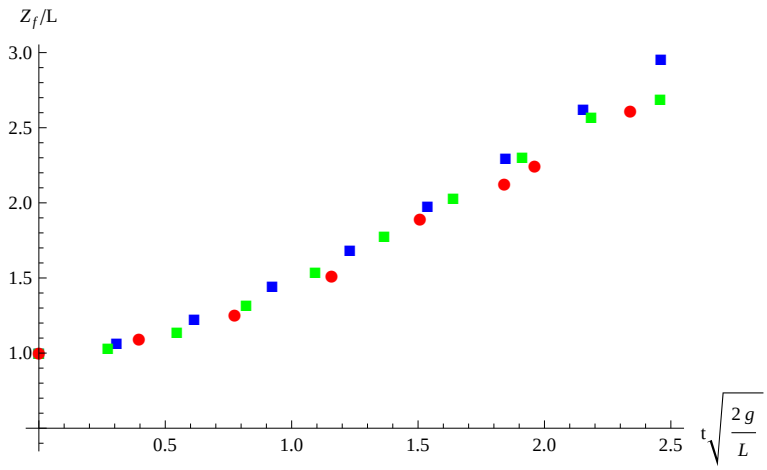
表–1 ベンチマークテストにおける計算機環境

計算環境	
OS(CPU)	CentOS 5.6
OS(GPU)	Ubuntu 12.04
CPU	Intel Core i7-3986X(3.30Ghz)
GPU	Geforce GTX 660Ti × 3
CPU 用コンパイラ	icpc 13.0
GPU 用コンパイラ	nvcc 4.0

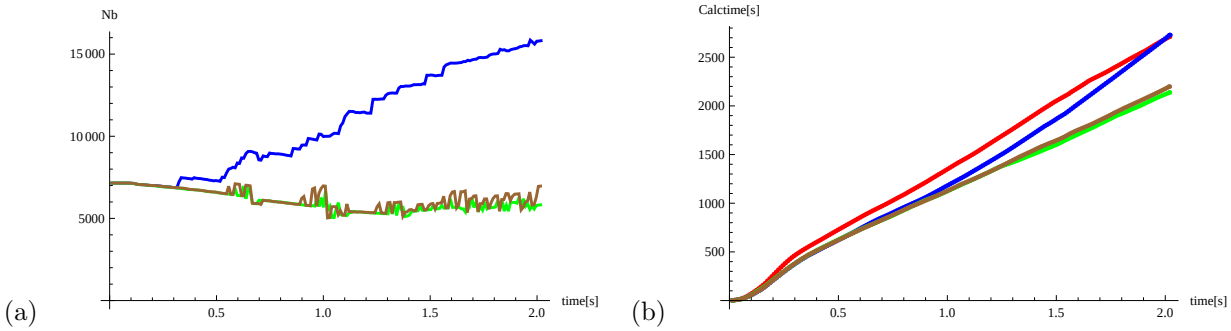
表-2 のように, 実験データ[17]と同等の初期設定において, 動的領域分割を用いた場合の 2GPU 数値実験を行い, 数値計算の信頼性について確認する.

数値実験では, 水柱に対して領域分割を崩壊方向 (x 軸) に垂直な断面に取る. この場合, **図-5** のように, 水塊が崩壊し重心が右に移動するにつれて, 領域の境界も移動する. 右壁にぶつかって反射した後は, 重心とともに領域境界も左に移動する. **図-5** で見られるように, 壁に水柱が崩壊して水面形が複雑になる場合でも領域分割した結果は, 単一領域での計算結果と同等になる. 右壁に到達するまでの水柱先端の時間発展を 2 領域分割で数値計算した結果は, **図-6** で見られるように, 実験と良く合っている.

ただし実験での初期水塊配置のように, 計算領域全体に対する境界体積比 ξ が大きい場合は, データ転送の時間が並列の効果を凌駕してしまうため, 計算速度の向上は見られない. 特に本研究での境界領域の定義では, 粒子混合が進めば境界粒子数は増大してしまうため, 複



図–6 2GPU 計算テスト (**表-2**) による水塊の先端位置. 横軸：物理時間 $t\sqrt{2g/L}$, 縦軸：先端の x 座標 x_f/L. ただし L は水柱の初期分布の横幅,g は重力加速度.　図で赤は実験データ[17], 青は 2 次元 2 GPU 計算, 緑は 3 次元 2 GPU 計算. 2 GPU 計算はいずれも x 軸に垂直な断面で水柱を等分割している.



図–7 粒子混合テスト（**表-3**) による 3 GPU 計算における境界粒子数と計算時間の変移.(a) 横軸：物理時間 [s], 縦軸: 境界粒子数 N_b,(b) 横軸：物理時間 [s], 縦軸:累積計算時間 [s].　各図で, 赤：1 GPU 計算, 青：3 GPU 計算, リセットなし, 緑：3 GPU 計算, リセット時間間隔 0.01[s], 茶：3 GPU 計算, リセット時間間隔 0.05[s]. リセットを行わない場合 (青）, 流体の壁への衝突が起こった後（約 0.3[s]),2 領域での粒子混合が進み, 境界粒子数が増大する. この場合, 累積の計算時間は増大し, 1 GPU 計算 (赤）に追いついてしまう. 一方, リセットを行った場合には, 境界粒子数は緩やかに減少し, 計算速度もほぼ一定に保たれる.

数 GPU 計算では, 境界粒子のリセットが必要である. これを確認するために, **表-3** の初期設定による 3 次元 3GPU 粒子混合テストを行う. このテストでは同一の初期条件の下で, 1 GPU 計算および 3 GPU 計算（リセットなし）および 3 GPU 計算（リセット有り）を比較する. **図-7** に見られるように, 水塊が右側の壁に衝突する前は, 各粒子はお互いの相対位置関係を保ったまま移動するので, リセットをしなくても粒子混合は起こらない. 一方, 壁に衝突すると激しい粒子混合が起こり, 境界粒子数は増大する. 実際, リセットを行わない場合, 時刻 $t \sim 2.0$ で, 全粒子数の約 8 割が境界粒子と見なされてしまい, 累積の計算時間は 1 GPU での計算と同じになる. 一方, リセットを行うと, 衝突により他領域の内部に進入したゴースト粒子はその領域の粒子として取り込まれるため, 境界粒子数はほぼ一定に保たれ累積計算時間も増加しない.　したがって動的領域分割で, 計算効率を上げるには, 粒子リセットが重要であることが確認できる. 次節では, 時間リセット間隔を行った 3 次元動的領域分割数値計算において, 計算が加速することを示す. 境界体積比 ξ の値を変えた場合の計算速度を複数 GPU を用いて検証する.

表–2 2GPU 計算テストにおける初期設定

パラメータ値	
計算領域 [m]	$0.6 \times 0.27 \times 0.36$
初期水塊領域 [m]	$0.15 \times 0.27 \times 0.3$
初期粒子間距離 b[m]	0.005 (2 次元) 0.0167（3 次元）
全粒子数	1633 (2 次元) 5916（3 次元）
境界体積比 ξ	0.025 (2 次元) 0.0835　(3 次元）

表–3 3 次元 3GPU 粒子混合テストにおける初期設定

パラメータ値	
計算領域 [m]	$2.5 \times 0.5 \times 1.5$
初期水塊領域 [m]	$1.66 \times 0.5 \times 0.4$
初期粒子間距離 b[m]	1.0/30
全粒子数	19364
境界体積比 ξ	0.08

5.2 3 次元ダム崩壊問題における複数ＧＰＵ計算ベンチマークテスト

次に 3 次元ダム崩壊の場合に複数 GPU 計算の計算速度の検証を行う. 以下の数値実験では, 領域分割をダムの崩壊方向（x 軸）に垂直に取る. この場合断面を固定すると, x 軸方向の水柱の幅が大きいほど, 境界体積比 ξ の値が小さくなり, 計算速度が向上する. まずそのような場合の典型例として, **表-4** のような状況を設定する.

表–4 ベンチマークテストにおける初期設定

パラメータ値	
計算領域 [m]	$4 \times 0.2 \times 0.4$
初期水塊領域 [m]	$2 \times 0.2 \times 0.2$
初期粒子間距離 b[m]	0.0167
全粒子数	31402
境界体積比 ξ	0.025

まず, この初期設定で最初の 10 ステップでの計算速度を**表-1** の環境設定での 1 CPU 計算 (1 コア）および複数 GPU 計算で比較する. **図 8(a)** に見られるように,1CPU 計算では, 近傍粒子探索にもっとも時間掛かっている. GPU 計算では, この部分では, 粒子番号とスレッドを対応させ, 近傍格子から近傍粒子を抽出し, 影響半径内部粒子数を近傍粒子番号登録用配列に登録するが, 行列ベクトル積部分のようなメモリアクセス遅延は見られない. 一方,GPU 計算では圧力ポアソン方程式解法部分にほとんどの計算時間が占められている. これは前述したように行列ベクトル積演算部分のメモリアクセス遅延が原因であると考えられる. 複数 GPU にした場合は, この部分が加速されるため, 全体としても 1 GPU に対して計算速度が向上することが分かる.

次に, 初期設定 4 および**表-1** の環境設定の元で, リセット時間間隔を約 $0.1[s]$ とし,5200 ステップまでの GPU 計算を比較する. 3 GPU の数値計算を, 行った場合の時間発展は**図-9** である.

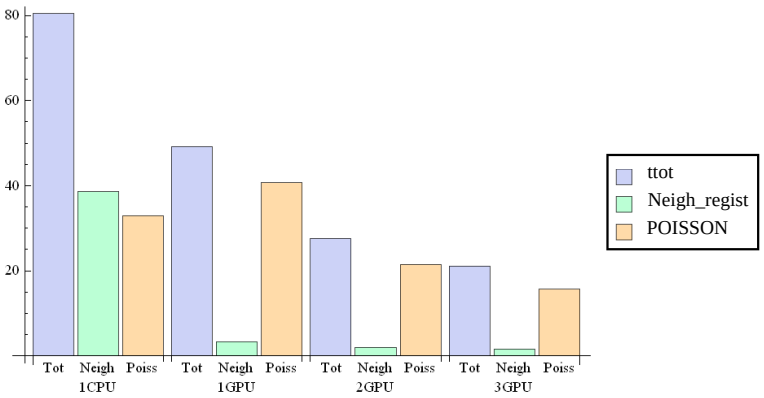
ただし, 初期粒子間距離を b として,MPS 粒子法での重み関数の影響半径は, 圧力ポアソン方程式解法部分で $h_p \equiv 3b$, それ以外で $h_c \equiv 2.1b$ とした. **図-9** で見られるように, 初期に x 軸に垂直な断面で 3 等分された水柱はダム崩壊により右側の壁に移動し, 3 領域の境界もそれに合わせて移動している. 同様の計算を 1 GPU および 2 GPU で行った場合の比較結果が**図-10** である. ある物理時間までの累積の計算時間を比較すると, 3 GPU で 1 GPU の約 2 倍, 2 GPU で 1 GPU の約 1.6 倍の計算速度が達成できている. また物理時間と計算時間の比はほぼ一定であり, 境界粒子のリセットの効果が現れていると言える.

次に加速率の ξ 依存性を検証するために,S を**表-4** の場合に固定し L_x を変えて ξ を変化させた場合の, 各 GPU 数での計算速度を比較する. **図-10** の結果より物理時間と計算時間の比はほぼ一定である. したがって初期の 50 ステップのみの計算を行い, ξ を変化させた場合の複数 GPU の加速率の値は**図-11** のようになる. **図-11** で見られるように, 加速率は $\xi \sim 0.05$ 程度では 3GPU でもせいぜい 1.5 倍程度であるが,$\xi < 0.01$ では 2GPU で約 1.8 倍,3GPU で約 2.5 倍まで大きくなることが分かる.

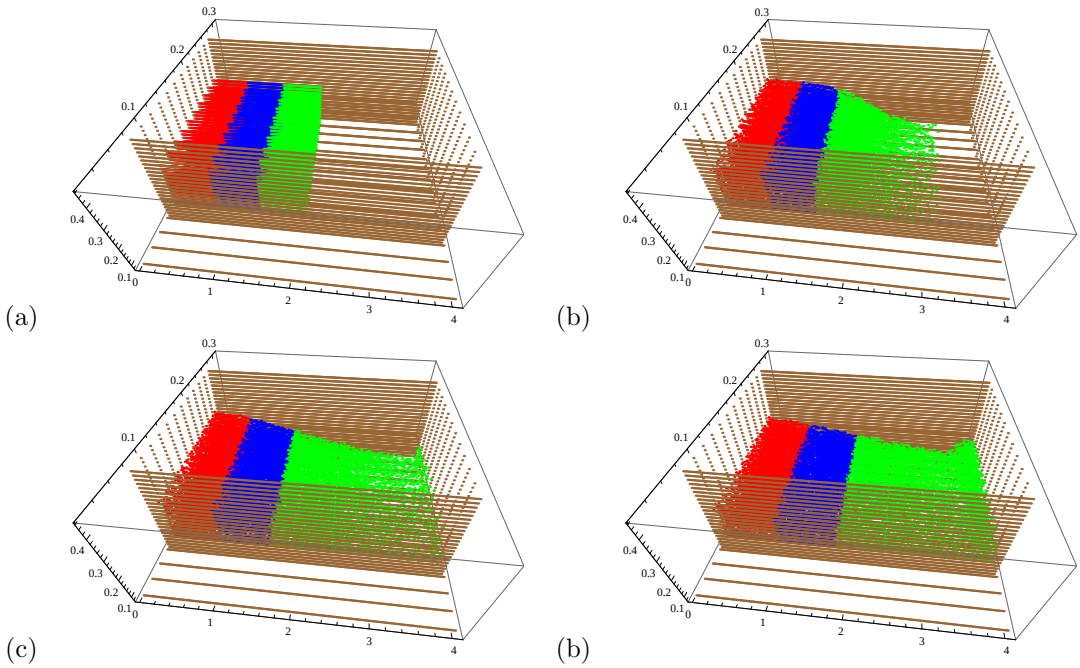
6. まとめ

本研究では, 領域分割を用いた粒子法メモリ分散型並列計算において, 流体全体が激しく動く場合にも有効な動的領域分割法を提案した. この方法では, 最初に分割した領域に従って, 粒子を各デバイスに分配した後, 粒子の動きに合わせて領域の境界を移動させていく. 基本的には, 流体の運動が激しくない場合, 各領域間での粒子の混合は少なく, 隣接領域間でのデータ転送の計算負荷は大きくならない. また, ダム崩壊のように流体全体が大きく移動する場合でも, 決められた時間間隔で境界粒子をリセットしてやれば, 粒子混合による計算負荷の増大は小さく抑えることができる. 実際, 我々は境界粒子をリセットすることによって, 複数ＧＰＵでの加速率をほぼ一定に保つことができることを確認した.

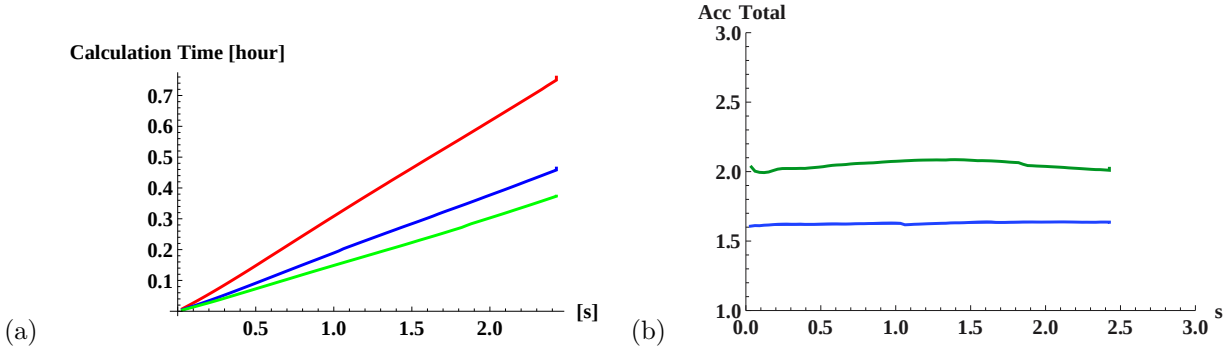
一般に MPS 陰解法の高速化では, 圧力ポアソン方程式の数値解法がボトルネックとなる. 我々は複数 GPU による 3 次元水柱崩壊数値計算において, 境界領域体積の全粒子領体積に対する割合 ξ が小さいほど行列ベクトル積演算での加速率が大きくなることを確認した. 一方で, メモリアクセス量の多い行列ベクトル積演算において, 最も単純なグローバルメモリを利用した 1 GPU での GPGPU 計算は,CPU 計算と同程度にしか加速されなかった. 一般にメモリ帯域の大きさからメモリ律速の状態では, GPU の計算速度はメモリ帯域の大きさから CPU のそれを上回るはずである. これについては, 行列, ベクトル積の部分でのメモリアクセスの遅延が GPU では顕著になっていることが原因であると考えられる. この点は、先行研究[9),10)]のように、今後テクスチャメモリ, シェアードメモリの利用やメモリアクセス



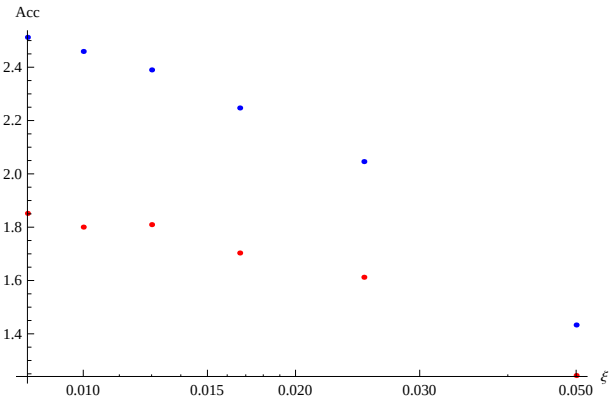
図–8 **表-4** の初期設定における,3 次元水柱崩壊シミュレーションの (初期 10 ステップにおける計算時間 [s] の比較.（紫：全計算時間, 緑：近傍粒子探索, オレンジ：圧力ポアソン方程式解法）. 左から 1CPU1 コア (Core i7), 1 GPU, 2 GPU, 3 GPU (Gefore GTX 660Ti).



図–9 **表-4** の初期設定における 3 ＧＰＵ計算の 3 次元ダム崩壊への適用例.(a)$t = 0[s]$,(b)$t = 0.4[s]$,(c)$t = 0.8[s]$,(d)$t = 1.2[s]$ でのスナップショット. 赤, 青, 緑は各時刻でデバイス１, デバイス２, デバイス３に格納されている粒子をそれぞれ表している.　初期時刻では, 水柱は左側（赤領域）, 中央 (青領域）と右側（緑領域）に粒子が等分割されて 3 デバイスに格納されている. 各デバイスの境界は, 時間発展するにつれて移動していく.



図–10 **表-4** の初期設定における 3 次元水柱崩壊シミュレーションの複数ＧＰＵ計算時間比較.(a) 物理時間 [s] に対する計算時間 [hour], (b) 物理時間 [s] に対する 1 GPU での計算時間と複数 GPU の計算時間との比（複数 GPU 計算の加速率）(赤：1 GPU, 青：2 GPU, 緑 3 GPU)



図–11 密閉領域内での 3 次元水柱崩壊シミュレーションの複数ＧＰＵ計算加速率の比較. 横軸：境界領域体積の全粒子領体積に対する割合 ξ, 縦軸：１ＧＰＵ計算に対する複数ＧＰＵ計算の加速率（50 ステップ ($t = 0.03[s]$ までの計算速度の比）. 赤は 2 GPU, 青は 3GPU での加速率を表す.ξ が小さいほど, 加速率は大きくなる.

の方法を変えることなどで改善する必要がある.

1 GPU での加速化を進めれば, 今後複数 GPU 計算が MPS 陰解法の高速化において有効になりうると考えられる. MPS 陰解法では, 圧力擾乱などの数値不安定性を解消するため, 圧力ポアソン方程式解法に対するさまざまな高精度化の試みが行われている [18),19)]. 本研究では, 越塚らによるオリジナルな形の圧力方程式を用いたが, 後藤らの高精度化されたラプラシアン項やソース項を用いても同様の高速化手続きが可能である. MPS 粒子法高速化に対しては圧力方程式を解かない陽解法も試みられている[14),15)]. しかしながら, さまざまな物理的状況で陽的解法でも陰的解法と同程度の高精度化が達成されるかは定かではない. 従って本研究での MPS 陰解法の高速化に対する試みは, 粒子法を高精度かつ高速なものにする上で意味を持つものと考えられる. また本研究における動的領域分割法は,MPI と併用することにより, 複数 GPU 計算に限らず分散メモリ型 CPU 並列計算にも適用可能である. この手法では, 流体の移動が激しい場合でも各デバイス上の粒子数がほぼ同数で固定されるため, 分散メモリ型並列計算に適しており,MPS 陰解法による大規模な数値計算や局所的な構造物周りの洗掘現象などにも適用できると期待される.

参考文献

1) 日本混相流学会, 混相流を伴う自然災害-数値解析の観点から現象を捉える-, 学術出版, 2012.
2) 越塚誠一, 数値流体力学, 培風館, 2002.
3) 越塚誠一, 粒子法, 丸善株式会社, 2005.
4) P. パチェコ,MPI 並列プログラミング, 培風館, 2007.
5) 入部綱清, 藤澤智光, 越塚誠一：粒子法による大規模解析におけるノード間通信の低減, Trans. Japan Society for Computational Engineering and Science, Paper No.20080020 2008 .
6) 青木尊之, 額田彰, はじめての CUDA プログラミング, 工学社, 2009.
7) J. Sanders, E. Kandrot,CUDA by Example 汎用 GPU プログラミング入門, インプレスジャパン, 2011.
8) 渡邊明英, 曽田康秀,CommonMP 利用における C#による実行上の課題と DLL の活用, 土木学会年次学術講演会講演会 2013 .
9) 原田 隆宏 , 政家 一誠 , 越塚 誠一, 河口 洋一郎 ： GPU 上での粒子法シミュレーションの空間局所性を用いた高速化, 情報処理学会論文誌 Vol.49, 2008.
10) 後藤仁志, 堀 智恵実, 五十里 洋行,A. KHAYYER：GPU による粒子法半陰解法アルゴリズムの高速化, 土木学会論文集Ｂ Vol.66, 2010 .
11) 原田 隆宏 , 政家 一誠 , 越塚 誠一, 河口 洋一郎 ： 複数の GPU を用いた粒子法シミュレーションの並列化, 情報処理学会論文誌 Vol.49, 2008 .
12) 室谷 浩平, 大地 雅俊, 藤澤 智光, 越塚 誠一, 吉村 忍：ParMETIS を用いた MPS 陽解法の分散メモリ型並列アルゴリズムの開発, Transactions of Japan Society for Computational Engineering and Science, Paper No.20120012, 2012.
13) P. Micikevicius, Supercomputing 2011 2011
14) 大地雅俊, 越塚誠一, 酒井幹夫 ： 自由表面流れ解析のための MPS 陽的アルゴリズムの開発 , 日本計算工学講演会論文集 20100013, 2010.
15) 大地雅俊, 山田祥徳, 越塚誠一, 酒井幹夫 ： MPS 陽解法における圧力および圧縮性の評価 , 日本計算工学講演会論文集 20110002, 2011.
16) 速水 謙 , 原田 紀夫：ベクトル計算機における Scaled CG 法の有効性について, 情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC） Vol.40, 1986 .
17) S. Koshizuka, A. Nobe and Y. Oka, Numerical analysis of breaking waves using the moving particle semi-implicit method. Int. J. Numer. Methods Fluids Vol.26, pp.751, 1998.
18) A. Khayyer, and H. Gotoh, A Higher Order Laplacian Model for Enhancement and Stabilization of Pressure Calculation by the MPS Method, Applied Ocean Res., Vol. 32, pp.124, 2010.
19) A. Khayyer, and H. Gotoh Enhancement of stability and accuracy of the moving particle semi-implicit method, Journal of Computational Physics, Vol.230, pp.3093, 2011.

(2013 年 3 月 18 日 受付)