動的領域分割を用いた流体構造連成による サスペンション・フローの大規模 GPU 計算

都築 怜理[1,a)]　青木 尊之[1,b)]

概要：サスペンション・フローは流体中に多数の物体を含み，流体構造連成が支配的な複雑な流れである．粒子法による計算を行うために物体を小さい粒子の集合で表現し，物体間の衝突には個別要素法 (DEM) を用い，流体計算には改良型 SPH 法を導入する．物体構成粒子は流体粒子や異なる物体の構成粒子間で相互作用する．それらの受ける力とトルクを合算して物体の運動方程式の時間積分を行うが，多数の構成粒子からの総和計算を伴うために計算はかなり複雑になる．GPU による計算において Linked-list を用いた近傍粒子探索のデータアクセスを高速化するために，定期的な GPU のデバイス・メモリ上の粒子データのソートが有効であることを確認した．また，力とトルクの総和計算を行う際，物体数に対して CUDA のスレッド並列を行う実装が高速であることも分かった．複数 GPU を用いるためにスライスグリッド法による動的領域分割を行い，256 個の GPU により 8,743 万個の流体粒子と 230 万個の物体構成粒子 (物体としては 2,304 個) を用いたサスペンション・フローの大規模シミュレーションを実現した．

キーワード：流体-構造連成計算，粒子法，動的負荷分散，GPU コンピューティング

A Large-scale Suspension-flow Simulation directly solving fluid-structure interactions with Dynamic Load Balance on a GPU Supercomputer

Tsuzuki Satori[1,a)]　Aoki Takayuki[1,b)]

Abstract: Suspension flows including fluid-structure interactions is one of challenging topics in fluid dynamics and many engineering applications. A large-scale particle-based method is a promising approach to carry out the numerical simulation for suspension flows. We have proposed an effective combination of the SPH method for fluids with the DEM (Discrete Element Method) collision for objects on a multi-GPU system. It is found that the sorting of particle data for the neighboring particle list using linked-list method improves the memory access greatly with a certain interval. We compare the two GPU implementations for the reduction of forces and torques applied to the particles constructing the objects. It is successful to apply the dynamic domain decomposition based on the 2-dimensional slice-grid method to our suspension flow simulation on multi-GPUs. A large-scale suspension flow of a tsunami interacting with a lot of objects with 87.3M particles including 2,304 cubic-shaped objects are successfully carried out on TSUBAME 2.5 at Tokyo Institute of Technology. The weak scalability is also measured from 4 GPUs to 256 GPUs.

Keywords: Fluid-Structure Interactions, Particle Method, Dynamic Load Balance, Multi-GPU computing

1. 序論

科学技術計算における粒子法とは，空間を任意に移動できる (ある程度の制限がある場合も含めて) 点上で物理量を時間積分する手法を指すことが多く，惑星軌道計算のような単一粒子計算よりも多数の粒子が相互作用しながら集団運動する系の計算が中心となっている．粒子は原子や分子と 1 対 1 対応する場合もあれば，プラズマ電子・イオンの数万個を代表する場合もあるし，単に連続体を空間離散化

[1] 東京工業大学
Tokyo Institute of Technology, O-okayama 2-12-1, Meguro-ku, Tokyo 152–8550, Japan
[a)] tsuzuki@sim.gsic.titech.ac.jp
[b)] taoki@gsic.titech.ac.jp

するための代表点である場合もある．

ハイパフォーマンス・コンピューティングの分野では，重力多体問題，分子動力学計算，SPH(Smoothed Particle Hydrodynamics) 法による流体計算，個別要素法 (DEM: Discrete Element Method) による粉体計算，メッシュフリー法による構造解析等があり，粒子間の相互作用や連結情報などは大きく異なる．重力多体問題や分子動力学計算では相互作用レンジが長く，N 体問題に代表されるように相互作用の計算負荷が支配的となるが，粉体計算では接触している粒子間のみ反発や摩擦の相互作用が生じ，メモリ参照の比重が高い．SPH 法などの粒子法による流体計算は，カーネル半径と呼ばれる範囲内の 100 個程度の粒子と相互作用するため，その中間的な計算負荷とメモリ参照と言える．実際の系は粒子数が非常に多く，数値計算ではできる限り多くの粒子を用いることで高い精度や計算できる現象が変わるため，ACM ゴードンベル賞 [1][2] にも度々登場する大規模計算が求められる分野である．近年，演算性能・メモリ性能・電力比効率などの観点から，世界トップクラスのスパコンにおいて GPU がアクセラレータとして搭載されている．それらの GPU スパコンでのアプリケーション開発の観点からは，CUDA プログラミングなどが必要であるばかりでなく，GPU ボード上のデバイス・メモリ量の制限や On-Chip の高速な共有メモリなどの階層的メモリ構造をうまく使う必要がある．また，GPU のデバイス・メモリ間のデータ通信はホスト計算機を介する必要があり，通信時間が大規模計算の大きなオーバヘッドになる可能性がある．

複数 GPU を使う大規模な粉体計算は動的負荷分散の手法により達成されている [3]．これにより高速化を目指すレーザープリンターのトナー粒子の挙動解析や，各種の化学工学，錠剤製造プロセスにおいて，紛体シミュレーションが可能になりつつある．SPH 法等の粒子法による流体計算も，京コンピュータで ParMETIS を用いた領域分割による大規模計算 [4] や，複数 GPU の 1 次元の動的領域分割による大規模計算 [5] が報告されていて，本著者のグループでも大規模な GPU 計算 [6] を行っている．

一方，流体と構造が共存する現象は実際に非常に多く，さまざまな分野で重要性が広く認識されている．東日本大震災の際，無数の瓦礫を伴った津波が押し寄せてくる映像は印象的であり，それらの衝突を考慮した津波被害シミュレーションの重要性は容易に理解できる．地盤工学における液状化現象も砂粒と流体の連成問題であるが，実スケールでシミュレーションされた例はない．土石流は水と土砂を含んだ流れであり，斜面災害のより先進的なシミュレーションを行うためには必須である．プラントの配管内の固体を多数含んだ流れの解析は化学工学において大きなテーマである．これらはサスペンション・フローと呼ばれ，大規模・高精度シミュレーションが殆ど行われていない．流体に対する格子を用いた計算は精度の点で有利であるが，移動する複雑形状の物体を扱うのは苦手である．一方，粒子法は複雑形状や流体構造連成計算を容易に行うことができる利点を持っている．

本研究は多数の球形粒子の集合で表現する複雑形状の構造物と SPH の流体との大規模連成計算を GPU スパコンで実現させることを目的とする．連結していない球形粒子を用いた DEM 計算は近接相互作用だけであるが，多数の粒子の剛体連結で構成する複雑形状の DEM 計算は個々の粒子に働く力の総和計算が必要になり，そのような複雑形状の物体が流体中に多数存在するサスペンション・フローでは通信コストが急激に増加する．複雑物体が分割領域間に跨って存在する場合は通信が非常に煩雑になる．多数の複雑形状の物体に対するメモリ管理，物体間の通信，領域間の通信を効率的に制御する必要がある．

本論文では，動的な領域分割を行いつつ各計算領域での流体粒子計算，DEM 粒子で構成される複雑形状物体の計算の効率化と通信負荷の低減を行い，これまでに達成されていない規模のサスペンション・フローの大規模計算を GPU スパコンで実現する．

2. 粒子法による流体構造連成計算

計算する系を構成する流体，壁，物体の全てを粒子により表現しているため，計算量域内には**図 1** に示すように，流体粒子，壁粒子，さらに各物体を構成する物体構成粒子の 3 種類の粒子が存在する．通常は 3 種類の粒子のいずれも同じデータ構造を持ち，これらの粒子に対して以下に示す 4 種類の相互作用を計算することで時間積分を行い，粒子法による流体構造連成計算が実現できる．

(1) **流体粒子ー流体粒子間の相互作用**　SPH や MPS (Moving Particle Simulation)[7] などの粒子法を用いてカーネル半径内の粒子から受ける相互作用を計算し，粒子の速度や位置，圧力や密度を時間発展させることができる．本研究ではこれまでの粒子法に比べて計算精度を大幅に向上した改良型 SPH 法 [8] を導入する．

(2) **流体粒子ー物体構成粒子・壁粒子との相互作用**　流体

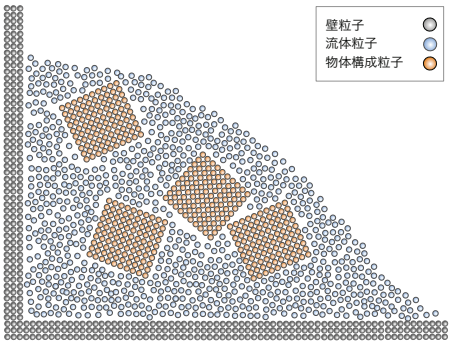

図 1　粒子法による流体構造連成計算の模式図．

Fig. 1　The kinds of particles to compute fluid-structure interaction simulations using particle-based method.

粒子計算の中で壁粒子は静止した境界条件として，物体構成粒子は移動境界条件として扱われる．

(3) **物体構成粒子ー流体粒子の相互作用**　物体構成粒子は流体粒子との接触点において圧力およびせん断応力を求める．接触点から重心を指す方向ベクトルと，それに垂直な面への射影成分ベクトルに分解する．

(4) **物体構成粒子ー物体構成粒子の相互作用**　複雑形状をした物体間の衝突の計算であり，接触している粒子間で DEM の衝突モデルによる反発と摩擦力を計算する．(3) と同じように接触点から重心を指す方向ベクトルと，それに垂直な面への射影成分ベクトルに分解する．

(1) と (2) の流体粒子の時間発展では通常の流体計算と同様に leap-frog 法，Runge-Kutta 法，予測子-修正子法等の 2 次精度以上の時間積分法を用いる．ここでは予測子-修正子法を用いている．(3) と (4) の物体の時間発展では，相互作用計算のあとに以下の力とトルクの総和計算を行い，並進運動と回転運動について前進オイラー法による時間積分を行う．物体構成粒子 $\boldsymbol{x}_i$ に作用する力を $\boldsymbol{f}_i$ とし，各粒子の剛体重心からの相対座標を $\tilde{\boldsymbol{r}}_i$ として，物体の質量を M とすると，各物体の並進速度 $\boldsymbol{v}$，角速度 $\boldsymbol{\omega}$，角運動量 $\boldsymbol{L}$ は，以下のように求められる．

$$\mathrm{M}\frac{d\boldsymbol{v}}{dt} = \sum_{i \in solid} \boldsymbol{f}_i \tag{1}$$

$$\frac{d\boldsymbol{L}}{dt} = \sum_{i \in solid} \tilde{\boldsymbol{r}}_i \times \boldsymbol{f}_i \tag{2}$$

$$\boldsymbol{\omega} = I^{-1}\boldsymbol{L} \tag{3}$$

I^{-1} は各時刻の慣性行列の逆行列であり初期の慣性行列の逆行列 I^{-1} から回転行列により形式的に求められる．また，物体の姿勢の管理にはクオータニオンを導入する．物体の並進運動により更新された重心位置と，回転運動により更新された姿勢から，$\boldsymbol{x}_i$ が更新される．

3. 粒子法の GPU 計算と高速化

3.1 Linked-list 法を用いた流体粒子の相互作用計算におけるソートの有効性評価

各粒子の持つ速度や座標，圧力などの時間更新される従属変数は通常，粒子構造体の中のメンバ変数として保持され，GPU の Device メモリ (CUDA ではグローバルメモリと呼ばれている) 上に確保される．GPU 上では 1 スレッドが 1 粒子を処理するスレッド並列計算を行う．各スレッドでは担当の粒子について粒子法の物理モデルに基いた演算を行う．

GPU による演算は CUDA コアと呼ばれる最小演算ユニットでスレッド毎に行われるのに対し，従属変数のデータはグローバルメモリ上にあるため粒子 i が粒子 j から受ける作用を計算する場合は粒子 j へのメモリアクセスの時間が問題となる．DEM では接触判定を行い，SPH や MPS などではカーネル半径内の粒子との相互作用を行う．全粒子との相互作用計算を行うことは非効率的であるため，計算領域を空間格子 (セル) に分割し自身と隣接するセルに属する粒子とのみ相互作用計算を行うセル分割法を用いて計算する．

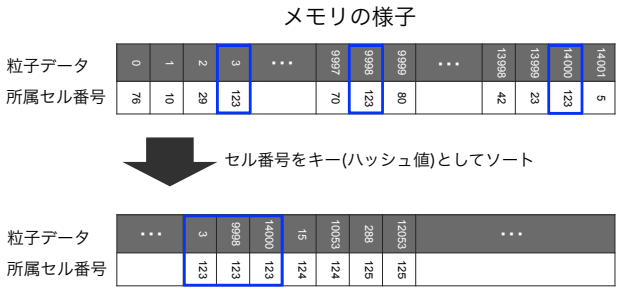

図 2　Linked-list に合わせて定期的に実行されるセル番号をハッシュ値とした粒子データのソート．

Fig. 2　The sorting of particle data by cell ID executed from CPU at fixed interval.

流体計算では通常，精度維持のためにカーネル関数の影響半径を初期粒子間隔の 3~4 倍程度の大きさに設定する．セル分割法の一セルの辺の長さをカーネル関数の影響半径以上にする必要があるため，少なくとも 27~64 個程度の粒子が一セルに格納される．静的に空間格子のメモリを確保し，すべての粒子番号をそれぞれの粒子が属するセルに登録する通常のセル分割法ではメモリ消費量が多過ぎる [7]．各セルにはセル内の一つの粒子の番号のみを登録し，各粒子が同一セル内の粒子の番号を鎖状につなぎ保持する Linked-list を GPU 上で導入している．Linked-list の生成は逐次的な処理を伴うため GPU では `atomic` 関数を用いて逐次的に行われるが，全体の計算時間と比べて Linked-list の生成時間は無視できるほど小さい．

粒子法の計算では，計算が進むにつれて粒子が移動するために周囲の粒子のメモリ・アドレスはランダムになる．例えば**図 2** に示すように，123 番のセル内に粒子番号が 3 番と 9998 番と 14000 番の粒子があったとすると，隣接するセル内のある粒子と 123 番のセル内の粒子との相互作用を計算する際には，123 番のセルに生成された Linked-list をたどって粒子番号が 3，9998，14000 番の粒子に順にアクセスする．しかしながら，図 2 の上側のように粒子が並んでいた場合，3 番の粒子と 9998 番，14000 番の粒子はメモリの位置が大きく離れている．CUDA の warp では一度に 16 連続アドレスの配列データを data load するが，3 番の粒子を読みに行った際の連続した 16 アドレス内には 9998 番や 14000 番の粒子のデータ存在しておらず，9998 番と 14000 番を含む 16 アドレスの data load がそれぞれ必要になり，メモリ・アクセスに時間がかかってしまう．

図 2 の下側のように，粒子データをセル番号順に並び替える操作を行うと，3 番と，9998 番，14000 番の粒子はメモリ上で連続するため，3 番の粒子にアクセスする際に，連続して隣接する 9998 番や 14000 番も data load され，高速に Linked-list をたどることができる．本研究では GPU の Device メモリ上で粒子の従属変数が連続となるよう粒子構造体の中に座標や速度や圧力など物理量のポインタをメンバ変数として保持し，それぞれのポインタに対して連続的に配列を確保する SOA(Structure of Array) 型のデータ格納形式による実装を行う．通常良く行われるような物理量をメンバ変数に持つ粒子構造体を配列として確保する AOS(Array of Structure) 型のデータ格納形式の実装の場合は，CUDA にも標準で付属する Thrust ライブラリの API を用いるなどにより粒子の並び替えの操作は比較的容易に行うことができる．しかし，SOA 型のデータ格納形式の場合は，並び替えのキー (ハッシュ値) となるセル番号を一時的にバッファに保存するなど実装は複雑であり，また 20 変数以上に及ぶ物理量のそれぞれに対して並び替えの操作を順次行うため効率的ではない．そこで簡単のため，セル番号による粒子データの並び替えは GPU から CPU 側にデータをコピーしてきた上で CPU 側で行う．セル番号による並び替えのコストや，CPU と GPU の粒子データのコピーが計算の大きなオーバーヘッドとならないように回数を抑えて定期的に実行する．Linked-list 法を用いない通常のセル分割法に対して粒子の並び替えを行う手法はこれまでにも多く報告されているが [9]，メモリ制約の大きい GPU 上で Linked-list 法を用いてメモリ容量を抑え，合わせてセル番号による粒子の並び替えを CPU 側から定期的に実行する本手法の有効性は報告されておらず，新しい手法の提案である．40 万個の壁粒子と 100 万個の流体粒子を合わせて 140 万個の粒子を用いた流体のダム崩壊計算に対して，セル番号による並び替えを行う場合と行わない場合の 1 ステップあたりの計算時間の変化を比較した結果を**図 3** に示す．セル番号による粒子の並び替えを行わない場合，計算ステップが進むにつれて実行性能が低下して計算時間が増加するのに対し，セル番号による並び替えを行う度に計算性能が回復し計算時間を大幅に短縮できていることがわかり，5000 ステップ目の段階で並び替えを行わない場合と比較して 8.6 倍の高速化を達成していることが分かる．

図 3 の計算では 50 ステップ毎に粒子の並び替えを実行しており，1 回の並べ替えに要する時間は 350 msec であった．50 ステップで平均すると 1 ステップ当たり平均 7 msec であり，全体の計算時間と比較して十分に小さい．

3.2 物体粒子の時間積分に対する GPU 実装

3.1 節で提案する方法は，2 章で述べた 4 種類の相互作用計算における流体粒子の時間発展 ((1) と (2)) において有効である．一方，物体粒子の時間発展は，複雑形状の物体が

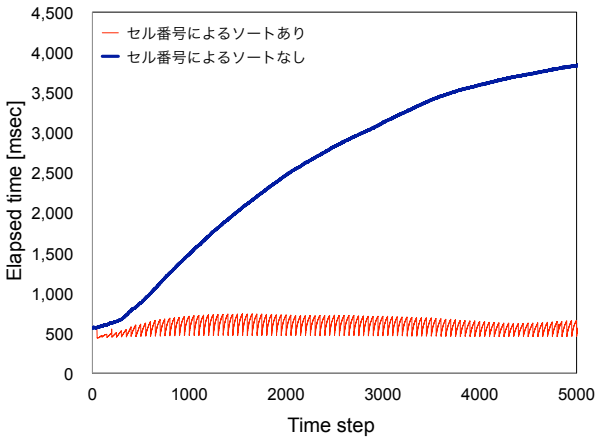

図 3 GPU における Linked-list 法を用いた近傍粒子探索にセル番号による粒子データのソートを行う場合と，行わない場合の計算時間の比較．

Fig. 3 The comparison of the computational elapsed time of the neighbor particle search using the linked-list with/without sorting by cell ID.

Algorithm 1 The time integration for objects

```
for k = 0, 1, 2... to N_obj do
  for j = 0, 1, 2... to N_particle do
    if particle[j] ∈ object[k] then
      object[k].force  += particle.force[j]
      object[k].torque += particle.torque[j]
    end if
  end for
  update object[k].quaternion
  update object[k].position, velosity
end for
```

流体中に多数存在するサスペンション・フローの計算を前提としているため，流体の時間積分の方法とは大きく異なる．計算領域内に全部で $N_{particle}$ 個の粒子があり，N_{obj} 個の物体があるとすると，物体の時間積分を行う疑似コードは **Algorithm 1** のように書ける．**Algorithm 1** 中の条件分岐は，粒子 j が物体 k の構成粒子であるかどうかを判定するものである．

GPU 計算では，(A) 物体数についての外側のループをスレッド並列化する場合と，(B) 領域内の粒子数についての内側のループをスレッド並列化する場合の 2 通りの実装が考えられる．(A) を実装する場合，外側のループに CUDA のスレッド並列を割り当て，各スレッドが内側のループの処理を行うように実装する．(B) を実装する場合，あらかじめ領域内の粒子数分の配列を用意しておき，GPU のスレッド並列で受ける力を書き込んでおく．外側のループで k 番目の物体の総和計算部分には Thrust ライブラリの `inclusive scan` を実行する．

サスペンション・フローの計算を前提とするために 140 万粒子のダム崩壊問題に対して計算領域内に一つあたり 1000 個の物体構成粒子からなる 100~400 個の立方体を加

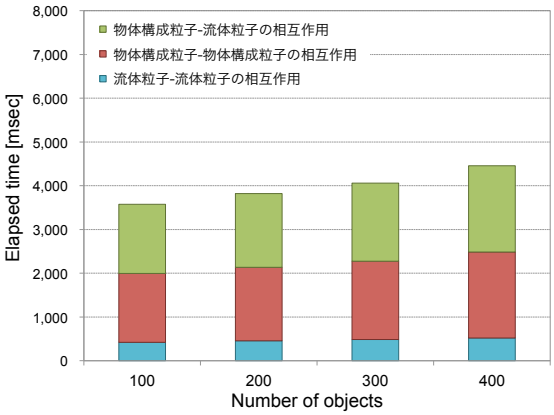

図 4 実装 (A) を用いた場合の計算時間の内訳.

Fig. 4 The breakdown of the computational elapsed time with the implementation of (A).

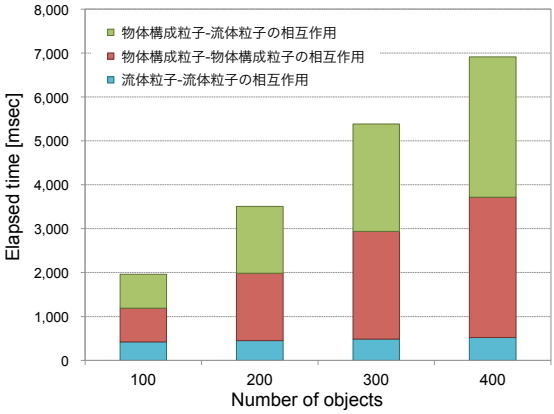

図 5 実装 (B) を用いた場合の計算時間の内訳.

Fig. 5 The breakdown of the computational elapsed time with the implementation of (B).

えて配置し，(A) と (B) のそれぞれの実装に対して，1 ステップあたりの計算時間を測定した結果をそれぞれ**図 4** と**図 5** に示す.

GPU 上で物体番号について並列化を行う実装 (A) は物体数が増えても緩やかに計算時間が増加するのに対し，物体数について並列化されていない実装 (B) を用いた場合，実装 (A) と比べ計算時間の増加が急である．物体数が 300 個を超えると実装 (B) の方が計算時間を要しており，数千個に及ぶ多数の物体を扱う本研究のような場合には，実装 (A) を用いることにより，物体数が増えても計算時間の増加を抑えることができる.

3.3 単体 GPU での実行性能

単体 GPU の計算について，NVIDIA 社の提供する性能解析ツールである Visual Profiler を用いてカーネル関数 の実行時間や FLOPS(Floating-point Operations Per Second) を測定した．140 万個の流体粒子に対して，1000 個の粒子から構成される立方体を 400 個配置したダム崩壊問題に対する実装 (A) を用いた場合の実行性能の内訳を**図 6** に示す.

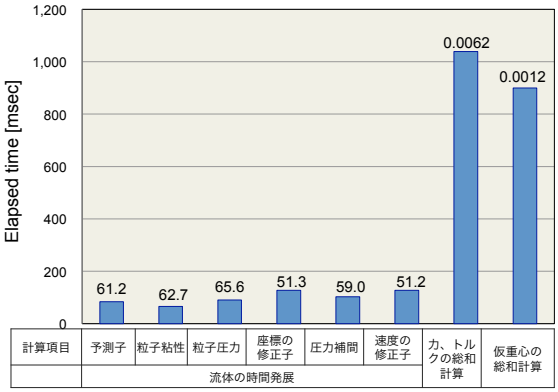

図 6 単体 GPU での実行性能 (棒グラフ上部の数値は GFlops 値).

Fig. 6 Elapsed time of each GPU Kernel (Numeric values on each bar show performance (Units: GFlops)).

表 1 セル番号によるソートを毎ステップ実行した場合の実行性能の理想値.

Table 1 The ideal performance for the case with the execution of sorting in every time step.

計算項目	単精度	倍精度
予測子 :	102.6 (2.59)	88.1 (6.72)
粒子粘性 :	94.3 (2.38)	82.5 (6.29)
粒子圧力:	104.7 (2.65)	122.1 (9.32)
座標の修正子:	67.4 (1.70)	91.1 (6.95)
圧力補間:	90.6 (2.29)	106.8 (8.15)
速度の修正子 :	67.2 (1.70)	91.7 (7.00)

単位 : GFlops (括弧内は理論ピーク性能に対する割合 (%))

浮動小数点演算は流体計算部分に対して単精度計算を行い，数値誤差の蓄積が生じ易い総和計算については倍精度計算を行っている．左から順に流体の時間発展計算 (図 4 における水色) 6 種類のカーネル関数と，物体の時間発展のための力とトルクの総和計算を行うカーネル関数，トルクの総和計算に用いる重心 (式 (2) の $\tilde{\boldsymbol{r}}_i$) を決めるための構成粒子の位置座標の総和計算を行うカーネル関数となっている．それぞれの実行性能 (GFlops 値) も記載してある.

流体の時間発展を計算する各カーネル関数の実行性能は図 6 の場合では最大で約 65.6 GFlops となり，NVIDIA Tesla K20X の単精度の理論ピーク性能である 3.95 TFlops に対して約 1.66 % と低い. セル番号による粒子のソートを毎ステップで実行した理想的な場合の各カーネル関数だけの実行性能を**表** 1 に示す．括弧内は理論ピーク性能に対する実行性能の割合を示しており，単精度で 2.65 %，倍精度で 9.32 % となっている．図 6 における流体の時間発展の計算はそれに対して約 62% 程度であり，ソートによるオーバーヘッドがかかることを考慮すれば，妥当な値である.

物体の時間発展のための総和計算を行うカーネル関数はメモリアクセスが支配的であるため，実行性能は非常に低い．一つあたりの構成粒子数が少ない物体が数万個から数十万個あるような粒度の細かいサスペンション・フロー計

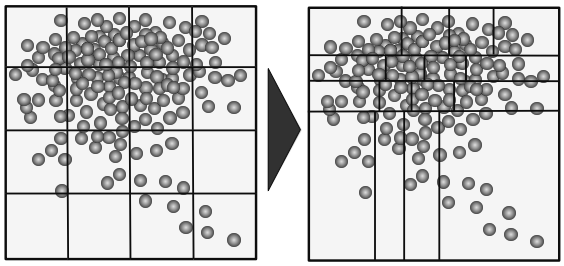

図 7 2 次元のスライスグリッド法を用いた動的負荷分散の概要.
Fig. 7 Outline of dynamic load balancing based on 2-dimensional slice-grid method.

算には実装 (A) はより有効となる．実装 (B) は総和計算に Thrust ライブラリの `inclusive_scan` を用いており計算効率は良いが，物体数について並列化できないため，物体数が数個以下の場合に有効になる．

本研究では計算コードの最適化は十分とは言えず，実行性能の上限値について詳細な議論を行うためには，Roofline Model[10] などを用いた検討が必要になる．

4. 動的負荷分散を用いた複数 GPU 計算

著者らの研究グループでは，これまで個別要素法による粉体シミュレーションの複数 GPU 計算および SPH 法による流体の複数 GPU 計算に対して，Fig.7 に示すように計算領域をスライスグリッド法により動的に 2 次元領域分割し，分割された各小領域内の粒子数を一定に保つ負荷分散を導入してきた [3][6][11][12]. ここでは，その手法を流体粒子と多数の物体が存在するサスペンション・フローの計算に**図 8** のように適用する．

小領域境界の移動は小領域を微少幅 Δl で分割して各分割内の粒子を数え上げ，粒子数の均一化に必要な境界領域に含まれる粒子を GPU 上でパッキングし，隣接小領域の GPU に転送する方法を取っている．異なるノード上に搭載された GPU 間では直接のデータ通信はできないため，ホスト計算機を介して通信を行っている．パッキングされた粒子を CPU 側に転送し，CPU 間で MPI ライブラリ [13] における `MPI_Isend` と `MPI_Irecv` を用いた非同期通信を行い，受信した粒子をもとに境界線を変更して領域の再分割が完了する．粒子データの GPU 間通信が発生するのは以下の 3 種，粒子が運動したことにより小領域間を移動する場合，小領域の境界を移動したことにより粒子の所属する小領域が異なった場合，カーネル半径内の粒子と相互作用するためにハロー領域 (= 小領域の境界から，カーネル半径の長さだけ内側の領域) に存在する粒子を隣接する小領域にコピーする場合である．これらの粒子移動により，GPU のグローバルメモリ上で粒子が移動前に占めていたメモリ領域は以降の計算で参照されなくなり．計算が進行するにつれてこのような未参照メモリがグローバルメモリ上に離散的に発生し，使用するアドレスが断片化していく．そこで，粒子データを先頭から詰め直して未参照メモリを除外するため，3.1 節で述べたセル番号によるソートとは別に粒子の再整列を実行する．

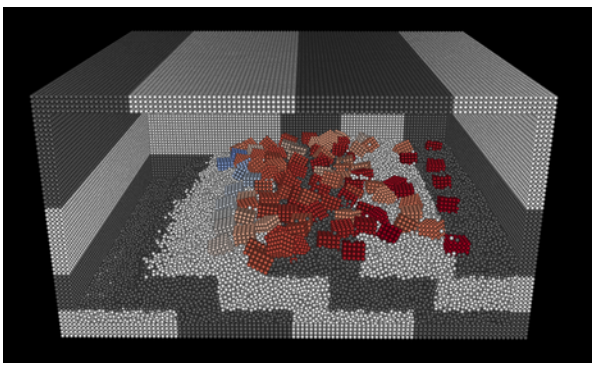

図 8 サスペンション・フローの流体構造連成問題の複数 GPU 計算に対する 2 次元のスライスグリッド法を用いた動的負荷分散.
Fig. 8 Dynamic load balance based on 2-dimensional slice-grid method for the suspension flow problems with multiple GPUs.

流体-構造連成の複数 GPU 計算では，一つの物体の構成粒子が同一小領域内にある場合は少なく，領域境界を跨り複数の GPU のグローバルメモリ上に構成粒子のデータが分散する．従って物体の時間積分は力とトルクの総和計算においてノード間で MPI ライブラリによる通信が必要になる．まず自身の小領域内で総和計算を行い `MPI_Gather` を用いて結果をルートに送信する．ルートでは, 受信した物理量を足し合わせて物体の時間積分した結果を `MPI_Bcast` を用いて各小領域にブロードキャストする．現在は 1 対多型の MPI による集団通信となっていて，これが並列化効率を低下させる．物体の時間積分は重心が存在する小領域のホストで行うことにより通信量を低減させることができるが，その物体の構成粒子が存在する小領域との関係を毎ステップ更新するための効率的な手法を開発する必要があるため本論文では行っていない．物体の時間積分に関する動的負荷分散は導入されていないが，サスペンション・フローであっても流体粒子に比べれば物体の数は圧倒的に少なく計算負荷は小さい．

5. TSUBAME2.5 における大規模計算

東京工業大学学術国際情報センターの GPU スパコン TSUBAME2.5 は，1 ノードに GPU (NVIDIA KeplerK20X) が 3 枚装着され，全体で 4,224 枚搭載されている．TSUBAME2.5 の複数 GPU を用いた粒子法によるサスペンション・フローの大規模計算を行い，合わせてその実行性能を測定する．

5.1 複数 GPU によるサスペンション・フローの大規模 GPU 計算

最大規模のサスペンション・フローとして，計算領域の

縦横高さを 144 m × 160 m × 60 m とし，深さ 1.6 m の静止した水を張り，そこに合計 2,304 個の浮遊する物体を配置する．流体の挙動を計算するために合計で 8743 万個の流体粒子を用い，物体を構成する粒子は 1,000 個に固定している．256 個の GPU を用い，スライスグリッド法による 2 次元の領域分割を行う．初期状態は物体が規則正しく並んでいて，1 GPU 当たりの計算領域は 9 m × 10 m × 10 m となり，6 個の物体を含んでいる．津波が押し寄せてくる状況を想定し，右側に高さ 4.8 m の水柱を設定している．

図 9 の最下図は時刻 6.8 sec の計算結果 (上側 8 枚の右列上から 2 番目) の拡大図である．サスペンション・フローの計算により，物体が浮遊していることで津波が砕波する先端がクリアでなくなっていることが分かる．物体を搬送するために波のエネルギーを奪われるために流れは緩やかになり，物体同士が衝突することで非常に不規則な流れになっている．大規模計算では多数の浮遊する物体と流れとの相互作用を計算することができ，物体が集団的に流される速度などの津波被害予測への貢献が期待できる．

物理時間で 6.8 sec に相当する計算を時間刻み 5×10^{-4} sec で 13600 ステップ計算するのに必要な計算時間は 43.1 時間であった．100 ステップ毎に計算結果のバックアップを行い，合計が 1.1 TB の計算結果を GPU 側から CPU 側に転送する時間と計算結果を出力するファイル I/O の時間まで含めた全計算時間は 96.1 時間であった．

5.2 TSUBAME2.5 における弱スケーリング

GPU の台数に比例させて，粒子を敷き詰める水平方向の面積を拡大させる．1 ステップの実行時間を T，全粒子数を $N_{particle}$ として Performance = $N_{particle}$/T により弱スケーリングを評価する．

図 10 において赤は 1 ステップの計算全体の実行性能，青は流体の時間発展部分の実行性能，緑は総和計算 (座標，力，トルク) 部分の実行性能を表す．黒の実線は 4GPU の全体の実行性能を GPU の台数に比例させ理想値であり，点線は 4GPU の流体の時間発展の実行性能を GPU の台数に比例させた理想値である．

流体計算部分のみの実行性能 (青) は，理想値 (点線) に対して，32GPU と 256GPU を用いた場合にそれぞれ 96.7%，52.8% であった．計算時間全体の実行性能 (赤) は，32GPU と 256GPU を用いた場合に理想値 (実線) に対してそれぞれ 73.4%，27.3% であった．流体計算のみの場合と比べて，総和計算が加わることにより並列化効率が悪化することを示している．総和計算部分では，座標，力，トルクの 3 種類について各々 Reduction 操作を行っており，ノード内で Reduction を行った後，全プロセス間で `MPI_Gather` を用いて全体の Reduction を行う．図 10 における 1 回の Reduction 操作に要する計算時間の内訳を**図 11** に示す．弱スケーリングが保たれていることによりノード内の Reduction 時間は一定に保たれており，GPU 数の増加に伴う実行性能の低下はノード間の Reduction(一対多型の MPI による集団通信) が原因だと分かる．

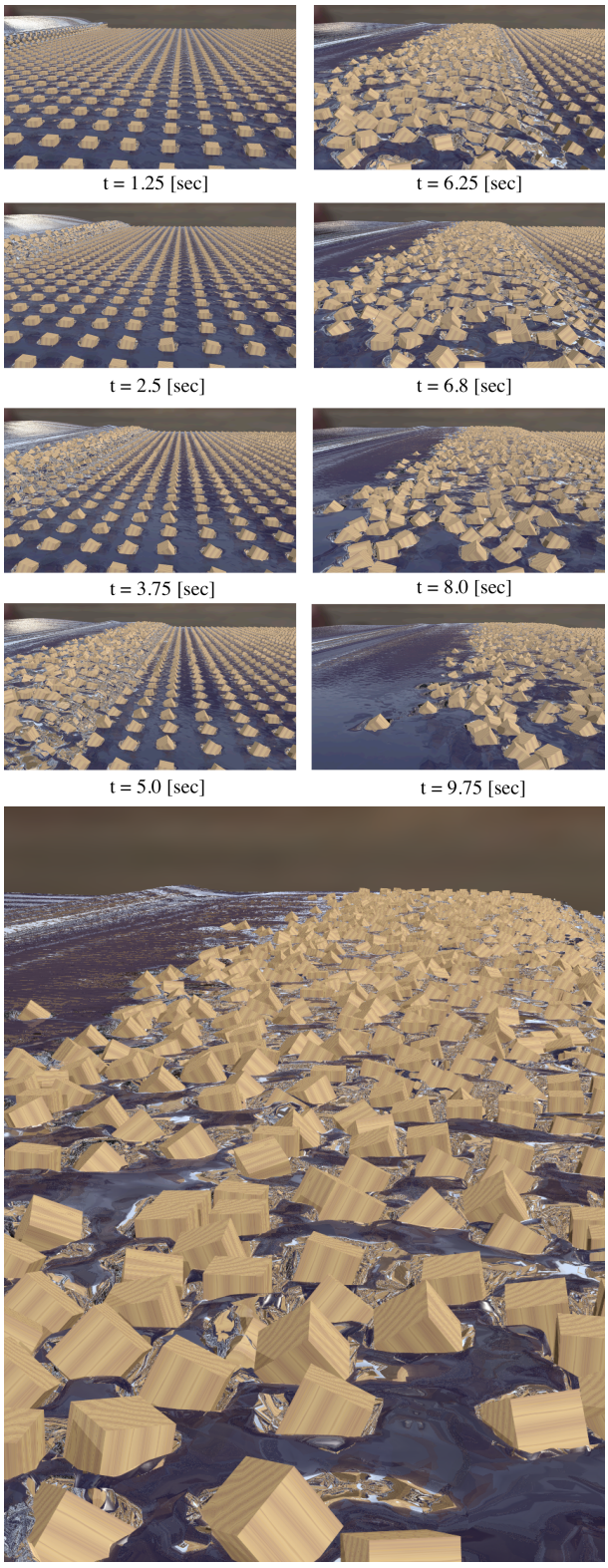

図 9 2304 個の物体を含む 8743 万個の粒子を用いた大規模サスペンションフロー計算を 256GPU を用いて計算した結果．

Fig. 9 A simulation result of suspension flow with using 87,430,000 particles with 2304 objects with 256 GPUs.

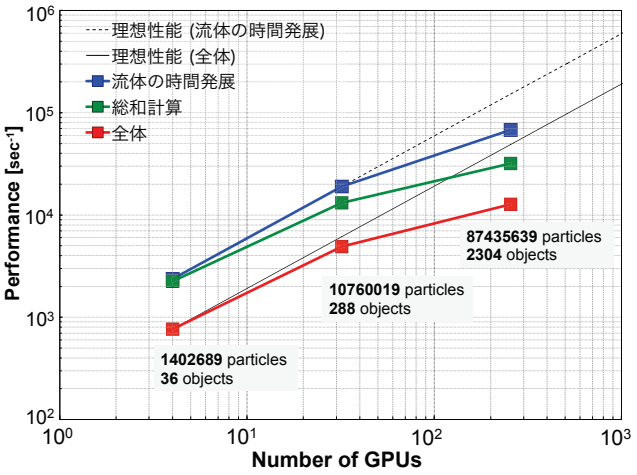

図 10 TSUBAME2.5 における弱スケーリングによる性能評価.
Fig. 10 Weak scalability of dam break simulations with cubes on TSUBAME 2.5.

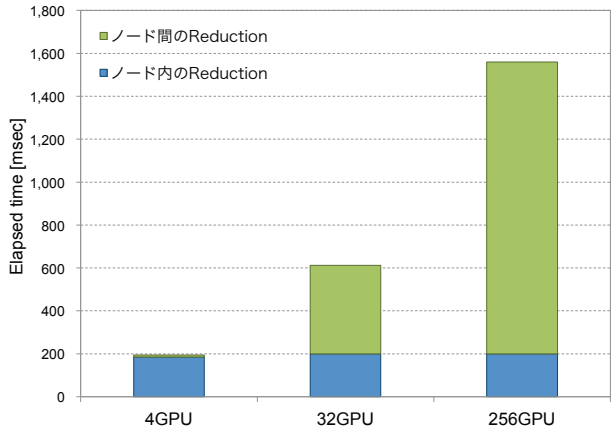

図 11 図 10 における Reduction の内訳.
Fig. 11 The break down of the data reductions in fig.10.

6. 結論と今後の課題

サスペンション・フローの GPU コードを開発し，最大で 256 GPU を用いて 144 m × 160 m × 60 m の計算領域に 8,743 万個の流体粒子と 2,304 個の浮遊する物体を含んだ大規模計算を実行することができた．また，以下のことが明らかになった．(a) Linked-list を用いた近傍粒子探索に対して定期的にソートすることで 8.6 倍に高速化することができた．(b) 物体構成粒子同士の相互作用により受ける力とトルクを合算して物体の時間積分を行う部分では，物体数に対して CUDA のスレッド並列を行うことにより，物体数が増えても計算時間の増加を抑えることができる．(c) これまで紛体のみの計算や流体のみの粒子計算に適用してきたスライスグリッド法による動的領域分割が流体と物体を含むサスペンション・フローの流体構造連成計算に適用することができた．

東京工業大学学術国際情報センターのスパコン TSUBAME2.5 において弱スケーリングを調べ，4GPU を用いた場合に対して，256 GPU を用いた場合には流体計算部分のみで 52.8%，連成計算を含む場合に 27.3% に並列化効率が低下する結果を得た．

今後の課題として，ノード内及びノード間の Reduction は多くの改善の余地がある．ノード間の Reduction について，今回は座標，力，トルクについて別々に Reduction を行ったが，力とトルクのノード間 Reduction についてはパッキングすることにより通信回数を半分に抑制できる．物体をマスターノードで管理しているが，重心を含むノードが管理することにより，通信負荷の分散を行うことができる．現在のノード内の Reduction では，物体が変形する場合も想定して毎回重心を求め直すことをしているが，剛体の問題に特化すれば，重心については総和計算が必要なくなる．また，物体の内部まで粒子を充填する必要もなくなるため，ノード内の Reduction による計算コストは軽減される．

本研究ではこれまで殆ど成されてこなかったサスペンション・フローの大規模計算を GPU スパコンで実現できたことが最も大きな意義であり，物体ごとに形状が異なるようなより現実的な場合も含め，効率化と高速化を進めて行く必要がある．

謝辞

本研究の一部は, 科学研究費補助金・基盤研究 (S) 課題番号 26220002「ものづくり HPC アプリケーションのエクサスケールへの進化」, 科学技術振興機構 CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」, 学際大規模情報基盤共同利用・共同研究拠点, 特別研究員奨励費・課題番号 14J12004「超大規模粒子シミュレーションの演算加速器型スパコンにおけるフレームワークの構築」, および革新的ハイパフォーマンス・コンピューティング・インフラから支援を頂いた．記して謝意を表す．

参考文献

[1] Hamada, T., Yokota, R., Nitadori, K., Narumi, T., Yasuoka, K. and Taiji, M.: 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'09, pp. 1–12 (2009).
[2] Ishiyama, T., Nitadori, K. and Makino, J.: 4.45 Pflops Astrophysical N-body Simulation on K Computer: The Gravitational Trillion-body Problem, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'12, pp. 1–10 (2012).
[3] Tsuzuki, S. and Aoki, T.: Large-scale granular simulations using Dynamic load balance on a GPU supercomputer, in *Poster at the 26th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'14 (2014).
[4] Murotani, K., Koshizuka, S., Ogino, M., Shioya, R. and Nakabayashi, Y.: Development of distributed parallel ex-

plicit Moving Particle Simulation (MPS) method and zoom up tsunami analysis on urban areas, in *Poster at the 26th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'14 (2014).

[5] Domnguez, J., Crespo, A., Valdez-Balderas, D., Rogers, B. and Gmez-Gesteira, M.: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters, *Computer Physics Communications*, Vol. 184, No. 8, pp. 1848 – 1860 (2013).

[6] 都築怜理, 青木尊之, 井元佑介, 田上大助：GPU スパコンにおける動的負荷分散を用いた粒子法による大規模流体シミュレーション, 第 28 回数値流体力学シンポジウム講演予稿集 (2014).

[7] 越塚誠一, 柴田和也, 室谷 浩平：粒子法入門, 丸善 (2014).

[8] 井元佑介, 田上大助：ある一般化粒子法に用いる近似作用素の打ち切り誤差解析とその応用, 第 27 回 計算力学講演会 講演論文集 (2014).

[9] Bader, M., Bode, A., Bungartz, H.-J., Gerndt, M., Joubert, G. R. and Peters, F. eds.: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, Vol. 25 of *Advances in Parallel Computing* (2014).

[10] Williams, S., Waterman, A. and Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures, *Commun. ACM*, Vol. 52, No. 4, pp. 65–76 (2009).

[11] 都築怜理, 青木尊之, 下川辺隆史：GPU スパコンにおける 1 億個のスカラー粒子計算の強スケーリングと動的負荷分散, 情報処理学会論文誌. コンピューティングシステム, Vol. 6, No. 3, pp. 82–93 (2013).

[12] Tsuzuki, S. and Aoki, T., : Large-scale Particle Simulations using Dynamic Load Balance on TSUBAME 2.0 Supercomputer, in *Proceedings of the 5th Asia Pacific Congress on Computational Mechanics and and 4th International Symposium on Computational* (2013).

[13] MPI Forum, : Message Passing Interface (MPI) Forum Home Page, `http://www.mpi-forum.org/` (2009).