SPH法による流体解析のGPU上での高速化

高田 貴正[1,†1,a)]　新田 知生[1,b)]　大野 和彦[1,c)]

概要：
流体シミュレーション手法の粒子法は，流体解析に限らず構造解析や衝突解析など幅広い分野で利用されている．一方で問題規模の拡大や高精度化などに伴い計算コストが大きくなっている．そのため，近年性能向上が目覚ましいGPUを用いた並列計算による高速化の研究が行われてきた．粒子法の一つであるSPH法では任意のカットオフ範囲内の粒子間でのみ相互作用する．粒子数の増加に伴い相互作用する近傍粒子の探索コストも大きくなるため，効率的な探索手法が提案されてきた．その一つにベルレリスト法がある．この手法では全ての粒子に近傍粒子を記録しておくための近傍リストを持たせ，探索時にこのリストのみを参照することで探索コストを削減できる．しかし，SPH法では粒子の密度が変化する圧縮性流体を扱うため近傍粒子の数が多くなる．そのためベルレリスト法を用いると，近傍リストの構築時および参照時のアクセスコストが大きくなってしまう．また，リスト全体のサイズを予測しづらく，あらかじめ十分な大きさのメモリを確保しておくことも難しくなる．そこで本研究では，リストサイズの計算，リストの構築，リストの参照でのアクセスをデータレイアウト最適化により高速化し，リスト構築時に必要なメモリを動的に確保することで，SPH法とベルレリスト法を採用した流体解析プログラムをGPU上に実装した．その結果，従来の手法と比較して全体の実行時間を短縮できた．

キーワード：SPH法，ベルレリスト法，CUDA，GPU，データレイアウト最適化

Acceleration of SPH-based fluid simulation on GPU

Kisei Takada[1,†1,a)]　Tomoki Nitta[1,b)]　Kazuhiko Ohno[1,c)]

Abstract: The particle method is widely used for physical simulations such as fluid analysis and structural analysis. One major issue of the method is its large computation cost thus GPU has been used to accelarate the computation. The SPH method is one of the particle method, in which particles interact with only particles within a cutoff range. Although it reduces the computation for the interaction, the neighboring particles must be searched every time step. Therefore, efficient scheme is required to reduce the cost of searching neighboring particles. Verlet list method is one of such efficient search methods. For each particle, this method constructs a neighboring list which stores neighboring particle indices. However, the number of neighboring particles often grows large in the SPH method when simulating compressive fluid. In such cases, the memory access cost on constructing and referring to the neighbor list is largely increased. Another issue is that static memory allocation is difficult because predicting the size of neighboring lists is difficult. We implemented SPH-Based fluid simulations on the GPU, adopting the Verlet list method. We introduced dynamic memory allocation for the neighboring lists. We also introduced data layout optimization to reduce the data access cost on computing the list sizes, constructing the lists, and referring to the lists. As the result of the evaluation, execution times of the simulations are reduced compared with the conventional method.

Keywords: SPH, Verlet list, GPU, Data layout optimazation

[1] 三重大学
Mie University
[†1] 現在，パナソニック株式会社
Presently with Panasonic Corporation
[a)] takada@cs.info.mie-u.ac.jp
[b)] nitta@cs.info.mie-u.ac.jp
[c)] ohno@cs.info.mie-u.ac.jp

ⓒ 2017 Information Processing Society of Japan

1. はじめに

連続体に関するシミュレーション手法の一つである粒子法は，連続体を粒子の集まりとして粒子同士の相互作用を計算することにより，流体などをシミュレートする手法である [1]．他の手法に対する利点として，形状データの生成が容易であること，大きな変形，ひずみを伴うシミュレーションを高精度に行えることなどが挙げられる．代表的な粒子法に SPH 法がある [2], [3]．SPH 法は，流体解析以外にも構造解析や衝突解析などに用いる研究が進み，幅広い分野で利用されている．一方で問題規模の拡大や高精度化などに伴い，シミュレーションに要する計算コストが大きくなっている．

大量のコアで並列に処理できる GPU は近年 CPU に比べて性能向上がめざましく，GPU に汎用的な計算を行わせる GPGPU では標準的な CPU 以上の処理の高速化を実現している [4]．このため，GPU を用いた粒子法の高速化の研究が行われてきた [5], [6], [7], [8], [9]．

SPH 法において，粒子間の相互作用は物理的に距離の近い近傍粒子のみに作用する．このため，各粒子の近傍粒子探索を毎ステップ行う必要があり，粒子数の増加に伴い探索コストも増加する．この近傍粒子探索を効率化する手法に，セルリンクリスト法とベルレリスト法 [8], [10] がある．セルリンクリスト法ではシミュレーション空間を分割しておき，粒子周辺の分割空間のみを参照することで探索範囲を限定する．ベルレリスト法では，各粒子に自身の近傍粒子を記録しておくための近傍リストを持たせる．近傍リストにカットオフ範囲内の粒子のみを記録することで，セルリンクリスト法よりもさらに探索範囲を限定できる．しかし，SPH 法では粒子の密度が変化する圧縮性流体を扱うので，強い圧力がかかった場合などに粒子が密集し近傍粒子の数が多くなる．そのため，近傍リストの構築時および参照時のアクセスコストが大きくなってしまう．またリストサイズを予測し，あらかじめ十分な大きさのメモリを確保しておくことも難しくなる．

そこで本研究では，リストサイズの計算，リストの構築，リストの参照時のアクセスをデータレイアウト最適化により高速化し，リスト構築時に必要なメモリを動的に確保することで，SPH 法とベルレリスト法を採用した流体解析プログラムを GPU 上に実装した．

以下，2 章で背景として GPU の概要とプログラム最適化手法，ならびに，SPH 法とその実装方式について概説する．続いて 3 章で提案手法を説明し，4 章で性能評価の結果を示す．最後に 5 章で結論と今後の課題を述べる．

2. 背景

2.1 GPU

GPU は演算を行うコアを大量に搭載し多数の処理を並列に実行できる．GPU ではコア数を超えるスレッドを生成でき，これらの大量のスレッドは 32 スレッド単位で分割され管理・実行される．この 32 スレッドのグループをワープという．ワープ内の 32 スレッドは同時に同じ命令を実行する SIMD 型の並列処理を行う．

ワープ内の各スレッドがアクセス命令を実行するとき，メモリ上で連続したアドレスへのアクセスは高速になる．GPU はキャッシュを搭載した階層型のメモリアーキテクチャを採用しており，デバイスメモリへのアクセスはキャッシュのラインサイズである 128byte 単位で行われる．ワープ内のスレッドが同時に同一キャッシュライン上のデータにアクセスすれば，複数のデータ転送を一度のデバイスメモリへのアクセスで行える．このようなアクセスをコアレッシングアクセスという．また，同一ライン内のデータに対して時間的局所性のあるアクセスを行えば，キャッシュメモリ上にデータが存在するので高速にアクセスできる．

各スレッドの実行パスが分岐処理により異なる場合，ワープは分岐部分のそれぞれのパスを逐次実行する．例えば，ワープ内のスレッドが if-else 文により 2 通りの実行パスに分かれた場合，各スレッドは true となったパスの各命令を実行した後に，false となったパスの各命令を実行する．異なるパスの命令を実行中のスレッドはアイドル状態になる．これをブランチダイバージェンスといい，アイドルスレッドの増加は性能低下の要因になる．

2.2 GPU 上でのデータレイアウト最適化

GPU 上の処理で構造体配列へアクセスするとき，構造体配列の各メンバのメモリ上での配置 (データレイアウト) をアクセスパターンに合わせて変更することにより，メモリアクセスを効率化できる [11]．データレイアウトの最適化手法として，構造体の分割とアライメントがある．

2.2.1 構造体の分割

構造体配列の特定メンバへの連続アクセスは，構造体を分割してコアレッシングアクセスの効果を高めたりキャッシュヒット率を向上させたりすることで高速化できる．

構造体のメンバはメモリ上に連続して定義順に並び，構造体の配列はこのメンバの並びが連続して繰り返される．`int` 型のメンバ `x,y,z` を持つ構造体の配列はメモリ上で**図 1** のように配置される．一般に GPU のスレッドはスレッド ID に対応した配列要素を処理するため，単純な `int` 型などの配列であればワープ内のスレッドは連続した領域へ同時にアクセスし，コアレッシングアクセスの効果が大きくなる．しかし，構造体配列の場合，各スレッドがスレッド ID に対応した要素の特定メンバへアクセスすると，不連続領域へのアクセスとなる．例えば，図 1 の構造体に対して各スレッドがスレッド ID に対応した要素のメンバ `x` にアクセスすると，メモリアドレスが不連続なアクセス

ⓒ 2017 Information Processing Society of Japan

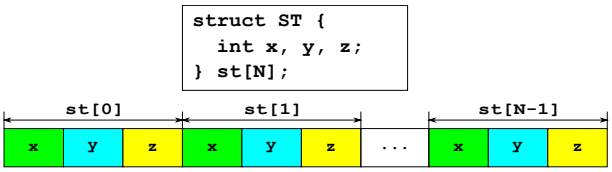

図 1　構造体配列のメモリ上の配置

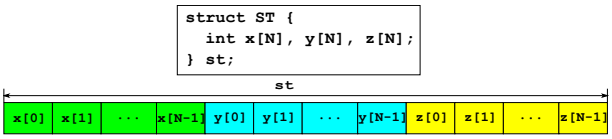

図 2　配列の構造体のメモリ上の配置

図 3　16 byte でアライメントした構造体配列のメモリ上の配置

となり，コアレッシングアクセスの効果が低下する．

構造体の配列を配列の構造体に変換することにより，このようなアクセスを高速化できる．図 1 の構造体配列を配列の構造体に変換すると，各メンバのメモリ上の配置は**図 2** のようになる．元の構造体のメンバ毎にメモリ上で連続して並んでいるため，各スレッドのメンバ x に対してのアクセスはコアレッシングアクセスとなり高速化できる．

2.2.2　アライメント

GPU の各コアによるデバイスメモリへの書き込み／読み出しは，1, 2, 4, 8, 16 byte 単位でのアクセス命令のいずれかにより実行される [12]．アライメントを行うことで，構造体配列の各要素へのアクセスを 8 byte や 16 byte 単位のアクセス命令でまとめて実行できる．例えば，図 1 のような 4 byte のメンバを 3 個持つ構造体の配列があるとして，その配列要素のすべてのメンバを更新するには 4 byte 単位の書き込み命令を 3 回実行する必要がある．しかし，このような構造体を 16 byte でアライメントした場合，デバイスメモリへの書き込みは 16 byte 単位の書き込み命令 1 回で実行される．図 1 を 16 byte でアライメントした場合のメモリ上の配置を**図 3** に示す．このように，アライメントにより複数ワードの書き込み／読み出しを 1 命令で実行することにより，メモリアクセスを効率化できる．

2.3　SPH 法

粒子法の一つである SPH 法 [2], [3] は，元々は銀河形成のシミュレーション手法として考案された．この計算法を応用し，圧縮性流体や非圧縮性流体，構造解析など幅広い分野で利用されている．一般的な SPH 法の実装では，1 個の粒子に対して位置，速度，密度，圧力などの属性をメンバとする構造体を定義し，そのような構造体の配列 (以下，粒子データ配列と表記する) に各粒子の属性値を格納する．各粒子に対しては個別の ID を割り振り，粒子データ配列の要素番号と対応させることで，個々の粒子の属性値を参照・更新できるようにする．

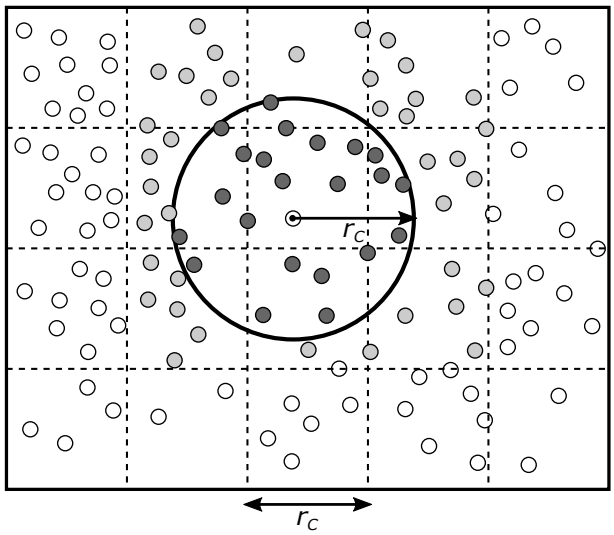

図 4　セルリンクリスト法

SPH 法では，相互作用はすべての粒子間ではなく，シミュレーション空間上で物理的に距離が近いカットオフ範囲内の粒子間のみ働く．ただし粒子は空間内を自由に移動できるため，タイムステップ毎に各粒子に対して，相互作用する近傍粒子を探索する必要がある．近傍粒子探索を高速化する手法としてセルリンクリスト法とベルレリスト法がある．

2.3.1　セルリンクリスト (CL)

セルリンクリスト法ではシミュレーション空間を同じサイズのセルに分割しておき，各粒子がどのセル内に所属 (存在) するかあらかじめ登録する [9]．**図 4** はセルリンクリスト法を用いた 2 次元での近傍粒子探索の例である．セルのサイズをカットオフ半径 r_C にすると，計算対象の粒子 p_i に対する近傍粒子の候補は，p_i が所属するセルおよびそれに隣接する 8 セル内の粒子に限定される．これらの候補の全粒子について p_i との距離計算を行い，距離が r_C 以下かどうかを判定する．このように探索範囲を限定することで，計算コストを削減できる．

近傍粒子候補との距離が r_C 以下か否かで，相互作用計算を行うか何もしないかの分岐が発生する．CPU 上の実装では何もしない場合に計算コストが削減されるが，GPU 上の実装では各スレッドが異なる粒子を担当して並列処理を行うため，ブランチダイバージェンスによるアイドルスレッドが発生し，性能が低下する．近傍粒子の候補内に範囲外の粒子が増えるとこの影響が大きくなるが，**図 5** に示すようにセルのサイズを半分にすることで，探索範囲をさらに限定し近傍粒子の候補を削減できる．

セルと粒子を対応付けるには，**図 6** に示すように，粒子データ配列を粒子が所属するセルでソートし粒子データ配列上で所属セルの境界となるインデックス値を求める．これにより，セル内の粒子データのみを参照できる．セルの

ⓒ 2017 Information Processing Society of Japan

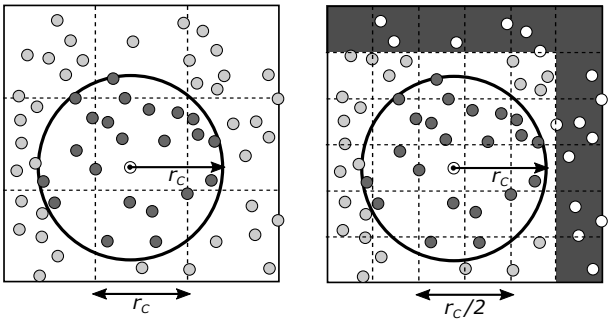

図 5 セルサイズの変更による探索範囲の削減

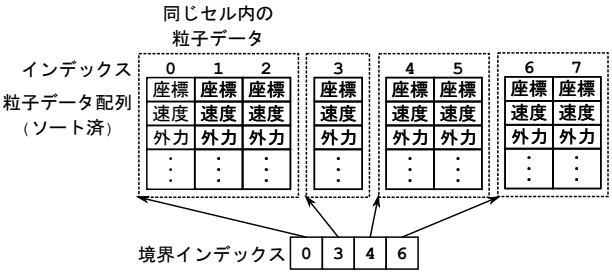

図 6 粒子データとセルの対応付け

サイズはカットオフ半径と同じなので，粒子が少しでも移動した場合に所属セルが変わる可能性があり，計算ステップ毎に粒子データ配列のソートおよび境界インデックス値の算出が必要になる．

2.3.2 ベルレリスト (VL)

ベルレリスト法では，各粒子に近傍粒子を記録する近傍リストを持たせる．あらかじめこの近傍リストを構築しておき，相互作用計算での近傍粒子探索時に近傍リストを参照することで，探索範囲を限定する．セルリンクリスト法に比べて探索時の近傍粒子の候補が少ないので，データアクセスおよび分岐を削減できる．

図 7 は 2 次元でのベルレリスト法を用いた近傍リスト構築の例である．リスト構築時にセルリンクリスト法を用いることで，高速化することができる．また，近傍リストには粒子間の距離がカットオフ半径 r_C 以下の粒子ではなく，$R_C(> r_C)$ 以下の粒子を登録する．これにより，リストの再構築を数ステップに 1 度に削減できる．

SPH 法でベルレリスト法を用いる場合，近傍リストに登録する半径 R_C は以下の式で求められる [8]．

$$R_C = r_C + \Delta h \tag{1}$$

$$\Delta h = 2 \cdot V_{max} \cdot C \cdot \Delta t \tag{2}$$

ここで r_C はカットオフ半径，V_{max} は全粒子中での最大速度，C はリストを維持するステップ数である．そして，リスト構築時から経過したステップ数分の粒子 P_i の総移動距離 D_i を以下の式 (3) のように求めておき，いずれかの粒子の D_i が $\Delta h/2$ を超えた時点でリストを再構築する．

$$D_i \mathrel{+}= |V_i| \cdot \Delta t \tag{3}$$

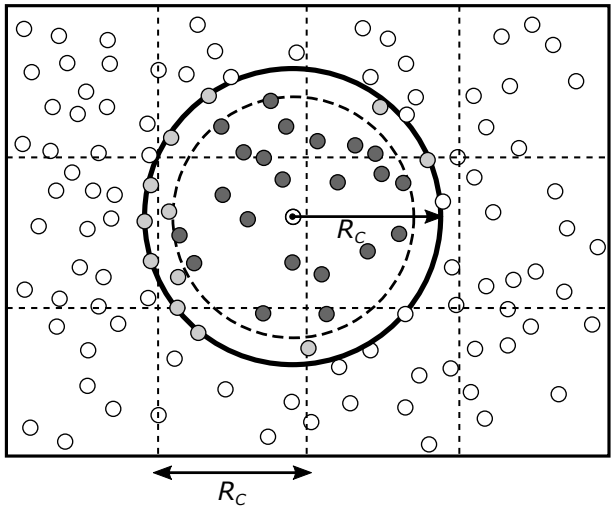

図 7 ベルレリスト法

このように再構築を行うことで，粒子のステップあたりの移動距離が小さい場合には，C ステップを超えてリストを維持できる．

3. 提案手法

本手法では，SPH 法とベルレリスト法を採用した流体解析 GPU プログラムの実装およびその高速化手法を提案する．

SPH 法は圧縮性流体のシミュレーション手法であり，粒子の密度が変化する．粒子に大きな圧力がかかった場合，密度が高くなり粒子が密集した状態になるため近傍粒子の数が増加する．このような場合，近傍リストに記録する粒子数が増加しリストの構築時および参照時のアクセスコストが大きくなってしまう．また，近傍粒子の最大数の事前予測が難しく，初期化時に十分な大きさのメモリを確保できない．そのため，リストサイズの計算，リストの構築，リスト参照時のアクセスをデータレイアウト最適化により高速化し，リスト構築時に必要なメモリを動的に確保する．

3.1 リスト参照時のアクセス最適化

GPU 上のスレッドは各々が 1 粒子の計算を担当し，対応する近傍リストのみにアクセスする．リスト参照時のアクセス最適化のため，全ての近傍リストを一つの 1 次元配列に格納し，ワープ内スレッドが同時に参照するリストの要素がメモリ上に連続して並ぶように配置する．各リストのメモリ上の配置を，**図 8**, **図 9** に示す．

近傍リストは，粒子データ配列 (図 6) に対するインデックスの並びで表現される (図 8 上)．リスト毎に含まれる要素数は異なるため，各リスト Li について要素数 N_{Li} および要素の配列 $I_{i,0}, \ldots, I_{i,N_{Li}}$ を保持する (図 8 下)．このとき同一ワープ内の 32 スレッドに対し，対応する 32 本の近傍リストの第 j 要素が配列上に連続して並ぶように格納する．これにより，ワープ実行時に各スレッドが各近傍リス

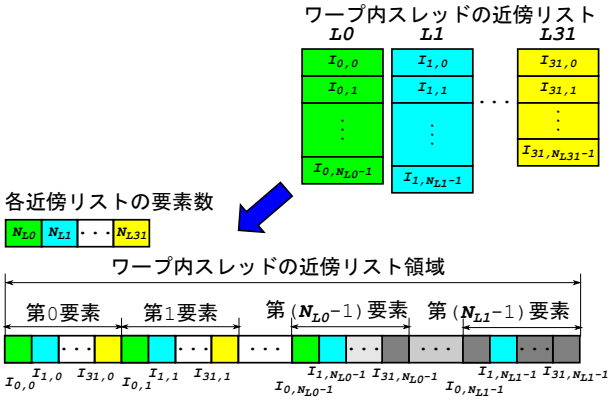

図 8　ワープ内の近傍リストのメモリ配置

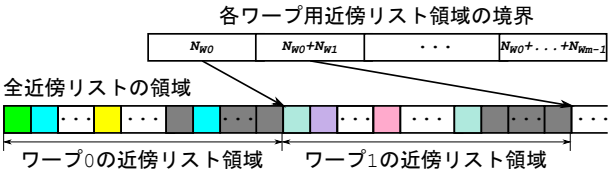

図 9　ワープ毎の近傍リストのメモリ上での割り当て

トの先頭から同じ位置にアクセスするとき，コアレッシングアクセスとなり高速化できる．ただしこの配置を実現するためには，ワープ内の近傍リストのうち要素数最大のものに合わせた領域が必要となる．このため近傍リストの要素数が異なる場合，配列内にはデータ未格納の要素が生じ，メモリ消費が大きくなる．

すべての近傍リストの格納領域は全体で一つの 1 次元配列として確保する (図 9 下)．これをワープの総数で等分すると，各ワープが図 8 の領域にアクセスするのが簡単になる一方，全近傍リストのうち最大要素数のものに合わせた領域確保が必要となり，さらにメモリ消費が大きくなる．このため，各ワープ毎に割り当てる近傍リスト領域を可変長とし，各ワープ用領域の境界を格納した配列を用意する (図 9 上)．粒子の総数を n とすると，これと一対一対応する近傍リストおよびスレッドの総数も n であり，ワープの総数は $m = n/32$ と表せる．ワープ $Wk(k = 0, 1, \ldots, m-1)$ の近傍リスト領域の大きさ (配列要素数) を N_{Wk} と表記すると，この配列の第 k 要素には N_{W0} から N_{Wk} までの和，すなわちワープ $W(k+1)$ の近傍リスト領域の先頭要素へのインデックスが格納されている．

3.2　近傍リストの構築手法

近傍リスト構築時には，各スレッドが自身に対応する近傍リストを構築することで，並列計算による高速化を行う．近傍リストの構築は以下の 4 段階の手順で行う．

(1) 各近傍リストの要素数の算出
(2) 各ワープ毎の近傍リスト要素数最大値の計算
(3) 各ワープ毎の近傍リスト領域確保
(4) 近傍リストへの近傍粒子の登録

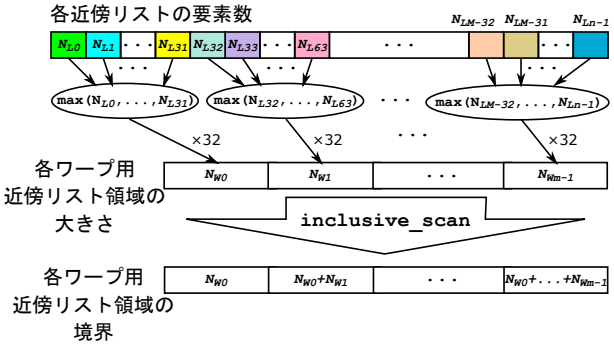

図 10　配列上の境界値の算出

提案手法では，手順 1 と 4 でそれぞれセルリンクリストを用いて近傍粒子の候補を限定し，全候補に対して距離計算を行う．このとき，図 5 に示したセルサイズの半減による探索範囲の削減手法を適用する．手順 1 では近傍リストに登録すべき粒子のインデックス値が求まるが，それを登録しておくための領域が確保されていないため，近傍リストに登録する要素数の加算のみ行う．そして，手順 4 では確保した領域に対し，近傍粒子のインデックスを登録していく．つまり，一度のリスト構築で近傍粒子の全候補に対して 2 度アクセスする．

3.3　近傍リスト領域の要素数算出

図 8 に示したように，近傍リスト要素の格納領域はワープ内近傍リストのうち最長のものに合わせて確保される．そのため，各ワープについて最大リスト長を求め，その値を 32 倍したものがそのワープに対し割り当てる近傍リスト領域の大きさになる．**図 10** に，各スレッドが対応する近傍リストの要素数を求めた後，各ワープ用の近傍リスト領域の大きさを計算する流れを示す．

各ワープにおいて近傍リスト要素数の最大値を計算するには Warp Shuffle 命令 [12] を用いる (図 10 上)．Warp Shuffle 命令はワープ内のスレッド間でローカルな変数を交換する命令であり，同期の必要がなくレジスタ間で直接データを交換でき，高速に最大値を計算できる．

次に，各ワープの近傍リスト領域の大きさに対し，並列プレフィックスサム (`inclusive_scan`) [13] を適用して総和を求める (図 10 中)．これは，配列の第 i 要素までの和を順に求める処理であり，この結果を格納した配列 (図 10 下) をワープ毎の近傍リスト領域の境界として利用する (図 9 上)．

3.4　リスト構築時のデータレイアウト最適化

近傍リスト構築では，各スレッドがセルリンクリスト法を用いて得られた全ての近傍粒子候補に対し，2 度のアクセスを行う．このとき，従来の SPH 法で用いるセルリンクリスト法 (図 4) と比べると各セルのサイズが大きくなっている (図 7)．このため，近傍粒子候補の数が多くなり全

体のアクセス回数も増加するため，アクセス最適化が必要になる．

近傍粒子候補とは距離計算を行うだけであるため，必要になる粒子の属性は座標 (x, y, z) のみである．このような特定メンバへのアクセスは，それらのメンバが連続してメモリ上に並んでいれば高速に実行できる．そのために，粒子データ配列の要素となる構造体のメンバのうち，座標のみの構造体を定義する．さらに，その構造体をアライメントすることで任意の粒子の座標メンバに対して高速にアクセスできる．各メンバが単精度の場合，x, y, z の 3 メンバを持つ構造体を 16 byte でアライメントする．各メンバが倍精度の場合，x, y の 2 メンバを持つ構造体を 8 byte でアライメントしたものと z メンバのみのスカラー配列に分割すれば，座標データに対して高速にアクセスできる．

4. 評価

提案した手法をオープンソースソフトウェア DualSPHysics 上に実装し，付属しているテストケースを用いて評価を行った．

4.1 評価プログラムと実行環境

DualSPHysics はダム崩壊や津波シミュレーションなどの問題を SPH 法を用いた流体解析により検証するオープンソースソフトウェアである [14]．大規模シミュレーションに適用するためにハードウェアアクセラレーションと並列コンピューティングによる高速化を行なっている．DualSPHysics_v4.0 ではセルリンクリスト法を採用しているが，本評価では提案手法を用いて DualSPHysics_v4.0 にベルレリスト法を実装し，付属されている動作確認のためのテストケースを用いてオリジナル版との比較を行った．本実験に用いたテストケースの概要を**表 1** に示す．表中のステップ数，粒子数の列はプログラム終了までに実行するステップ数とシミュレーション内で扱う粒子の総数である．また，総和計算の列は，流体構造連成などの総和計算を行うか否かを示している．

評価環境は**表 2** に示す 2 種類の GPU 搭載 PC 上で行った．Tesla K20c は Kepler 世代アーキテクチャ [15]，GeForce GTX980 は Kepler の次世代となる Maxwell 世代アーキテクチャ [16] を採用している．

4.2 実行時間の比較

各テストケースについて，オリジナル版と提案手法版の実行時間を計測した．また，単精度 (float) と倍精度 (double) を用いた 2 通りの実行時間も計測した．オリジナル版に対する提案手法版の速度向上率を，**図 11**, **図 12**(Tesla K20c 上) および**図 13**, **図 14**(GeForce GTX980 上) に示す．このうち，図 11, 図 13 は 2 次元空間，図 12, 図 14 は 3 次元空間のテストケースの結果である．

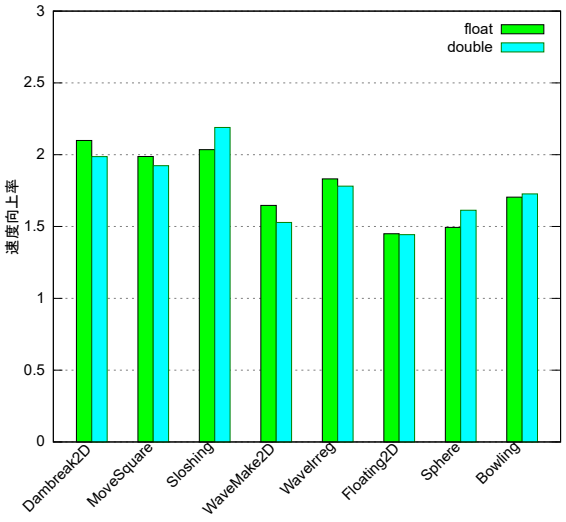

図 11 Tesla K20c の 2D テストケースでの速度向上率

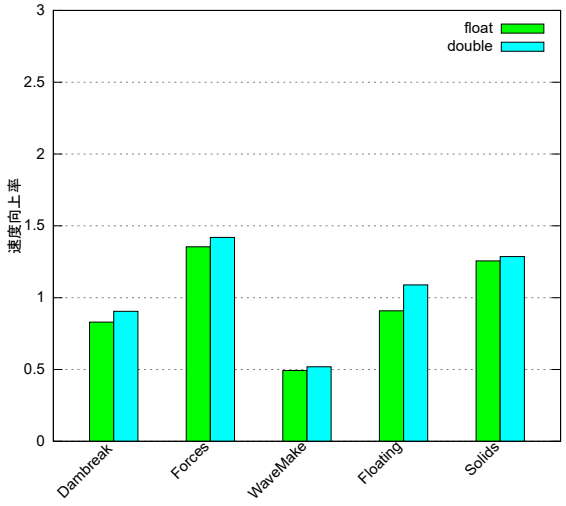

図 12 Tesla K20c の 3D テストケースでの速度向上率

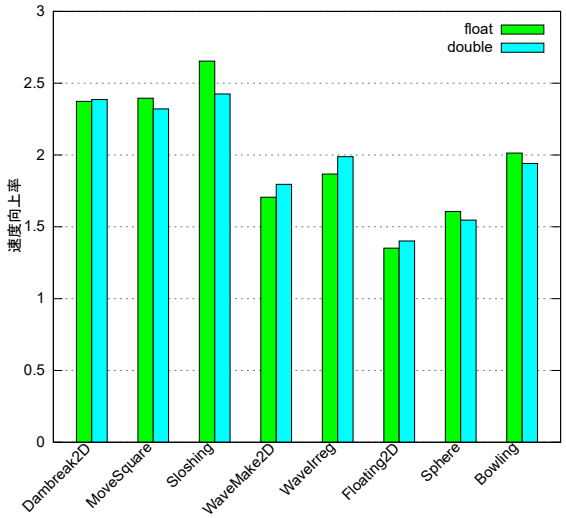

図 13 GeForce 980 の 2D テストケースでの速度向上率

2 次元空間の評価では，いずれの評価環境，テストケースにおいても提案手法により速度向上を実現している．一

ⓒ 2017 Information Processing Society of Japan

表 1　本実験に用いたテストケース

テストケース	空間	ステップ数	粒子数	総和計算	概要
DamBreak2D	2D	72415	21001	無	ダム崩壊シミュレーション
MoveSquare	2D	10400	20302	無	正方形の剛体が一定速度で水の中を進む
Sloshing	2D	411017	22064	無	水の入ったタンクを回転させる
WaveMake2D	2D	164715	53087	無	一定周期の波を生成する
WaveIrreg	2D	197227	51234	無	不規則な波を生成する
Floating2D	2D	675911	602920	有	箱を浮かべた水に波を生成する
Sphere	2D	45202	59092	有	水の入ったタンクに球体を落とす
Bowling	2D	155972	11576	有	積み上げた箱に，斜面を転がした球体を衝突させる
DamBreak	3D	18770	171496	無	ダム崩壊シミュレーション
Forces	3D	33656	44063	無	直接流体に外力を加え，タンク内の水をかき回す
WaveMake	3D	59331	518744	無	一定周期の波を生成する
Floating	3D	44823	469508	有	箱を浮かべた水に波を生成する
Solids	3D	43636	115016	無	空間内に剛体粒子を配置したダム崩壊 (SPH 法と DEM 法の併用)

表 2　評価環境

CPU	メモリ	GPU
Intel Core i7-930	6GB	Tesla K20c
Intel Xeon CPU E5-1620	16GB	GeForce GTX980

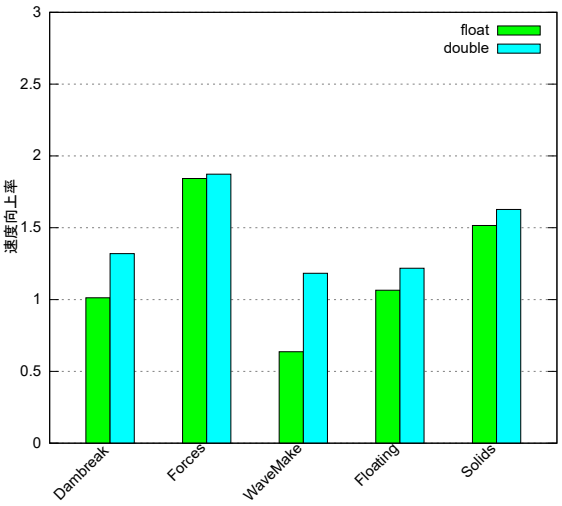

図 14　GeForce 980 の 3D テストケースでの速度向上率

方で，3 次元空間での評価では，テストケースによって性能に大きく差が出ており，Forces や Solids で 1.2–1.4 倍程度の速度向上を達成している一方で，WaveMake ではオリジナル版の半分程度に実行速度が低下している．

4.3　メモリ消費量の比較

各テストケースについて，オリジナル版と提案手法版のメモリ消費量を比較した．データの精度は単精度である．オリジナル版に対する提案手法版の消費メモリ増加率を，**図 15**，**図 16** に示す．図 15 は 2 次元空間，図 16 は 3 次元空間のテストケースの結果である．

いずれのテストケースにおいても，提案手法版ではオリジナル版よりメモリ消費量が増加している．しかしながら 2 次元空間の評価では，いずれの場合にも増加率は

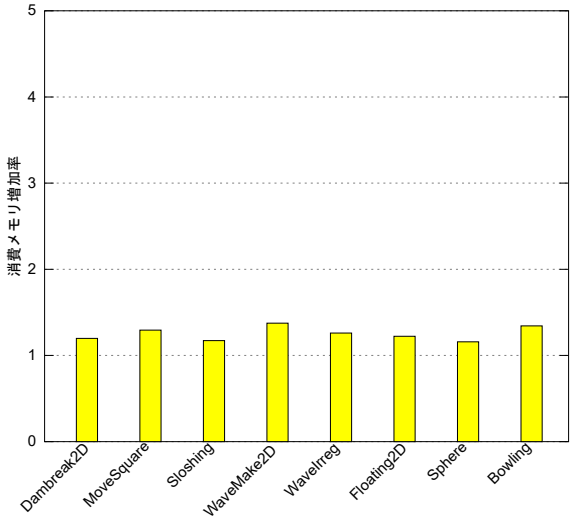

図 15　2D テストケースでのメモリ消費量の増加率

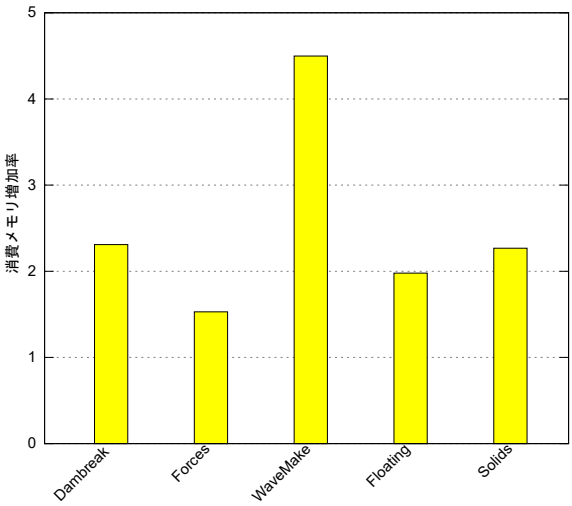

図 16　3D テストケースでのメモリ消費量の増加率

1.2–1.4 倍程度に留まった．一方で，3 次元空間での評価では 1.5–4.5 倍と大幅にメモリ消費量が増大した．とくに WaveMake ではオリジナル版の 4.5 倍となり，4.2 節に示

ⓒ 2017 Information Processing Society of Japan

表 3 Dambreak2D の各処理の実行時間 (秒)

	セルリンクリスト法		ベルレリスト法	
	実行時間	比率	実行時間	比率
粒子データ配列のソート	44.0	60.8%	4.2	13.8%
近傍リスト構築	0.0	–	4.1	13.4%
相互作用計算	12.4	17.1%	10.5	34.4%
その他	16.0	22.1%	11.7	38.4%
全体の実行時間	72.4	–	30.5	–

表 4 Dambreak の各処理の実行時間 (秒)

	セルリンクリスト法		ベルレリスト法	
	実行時間	比率	実行時間	比率
粒子データ配列のソート	17.6	16.9%	2.2	2.1%
近傍リスト構築	0.0	–	26.4	25.7%
相互作用計算	74.3	71.5%	65.0	63.4%
その他	12.0	11.5%	9.0	8.8%
全体の実行時間	103.9	–	102.6	–

表 5 Solids の各処理の実行時間 (秒)

	セルリンクリスト法		ベルレリスト法	
	実行時間	比率	実行時間	比率
粒子データ配列のソート	57.0	12.4%	6.9	2.1%
近傍リスト構築	0.0	–	55.8	17.2%
相互作用計算	350.5	76.5%	223.6	69.1%
その他	38.7	8.4%	28.2	8.7%
全体の実行時間	458.2	–	323.5	–

した実行速度低下の一因と考えられる．

4.4 各処理の実行時間の比較

セルリンクリスト法を用いるオリジナル版では，セルと粒子の対応をとるためにステップ毎に粒子データ配列をソートする必要がある．これに対しベルレリスト法を用いる提案手法では，近傍リスト構築時にソートが必要になるが，リスト構築は数ステップに一度でよいため，ソート回数は削減できる．その一方で，粒子データ配列への参照が近傍リストのインデックス値を用いた間接参照となる点や，近傍リスト構築時に用いるセルサイズを大きくしているため候補粒子が増大する点など，コストが増加する要因も抱えている．このため，テストケースや実行環境により両者のトレードオフ関係に差異が生じ，性能が向上する場合と低下する場合が生じると考えられる．

そこで，オリジナル版と提案手法版のそれぞれについて，テストケース Dambreak2D, Dambreak, Solids を用いて各処理ごとの実行時間を計測した．結果を**表 3**, **表 4**, **表 5** にそれぞれ示す．実行環境は GeForce GTX980 搭載 PC であり，データは単精度である．

いずれのテストケースでもオリジナル版と提案手法版を比較すると，データ配列のソート時間は 1/8–1/10 程度と大幅に削減できている．前記したように，ベルレリスト法では数ステップ毎の近傍リスト再構築時のみソートを行えばよいためである．

また，隣接セル内の全粒子を近傍粒子候補とするセルリンクリスト法と比べると，ベルレリスト法では近傍リストにある粒子のみを候補とするため，相互作用計算に要する時間が減少する．しかし今回評価したテストケースの範囲でも，Dambreak2D, Dambreak, Solids のそれぞれで 15%, 13%, 37%と，オリジナル版に対する時間削減率に大きな差が見られた．

一方，近傍リストの構築はベルレリスト法のみで必要な処理であり，オリジナル版に対して提案手法版の速度低下の原因となる．この処理時間が全体に対して占める割合も，Dambreak2D, Dambreak, Solids のそれぞれで 13.4–25.7%と差が見られた．

各処理が全体に占める割合に注目すると，オリジナル版において Dambreak2D では粒子データ配列のソート時間が 6 割以上を占めるのに対し，Dambreak, Solids では 7 割以上を相互作用計算時間が占めており，ソート時間は 2 割未満である．このため，前者では 2 倍以上の速度向上を得ているのに対し，後者はソート時間が同程度の削減率を達成しているにも関わらず，速度向上率はあまり高くない．Dambreak では，ベルレリスト法で近傍リスト構築と相互作用計算にかかる合計時間がリンクリスト法の相互作用計算よりも大きくなっている．DamBreak では壁粒子と水粒子の相互作用計算のみで比較的計算量が小さい．分岐後の計算量が小さい場合はスレッドのアイドル状態が短くなり性能低下が抑えられるため，ベルレリスト法を用いたことによるアイドルスレッドの削減が DamBreak ではあまり効果がなかったと考えられる．一方で，流体と剛体の複合的な相互作用をシミュレートする Solids では相互作用のための計算量が大きくなる．このためアイドルスレッドの削減により相互作用計算の実行時間を大幅に短縮できたと考えられる．つまり，相互作用計算が複雑になるほどベルレリスト法が効果的であるといえる．

4.5 テストケースの特性による影響

4.4 節の結果に見られるように，SPH 法実行時の各処理に要する時間はテストケースにより大きく異なり，全体の実行性能を左右している．そこで，提案手法への影響を考察するため，各テストケースの特性を測定した．結果を**表 6** に示す．

近傍リストの平均長は，近傍粒子の平均的な個数を表している．同程度の粒子密度であれば，近傍粒子の個数は 2 次元より 3 次元のほうが多くなるため，表 6 でも DamBreak 以降の 3 次元テストケースでは，2 次元のものより近傍リスト平均長が長くなっている．また，それらの中でも WaveMake は，他の数倍に達している．ベルレリスト法で

ⓒ 2017 Information Processing Society of Japan

表 6　各テストケースの挙動

テストケース	近傍リスト			スレッド
	平均長	再構築回数	再構築間隔	アイドル率
DamBreak2D	27.99	9772	7.41	23.4%
MoveSquare	31.79	2084	4.99	31.5%
Sloshing	23.23	82204	5.00	24.8%
WaveMake2D	57.80	32947	5.00	6.2%
WaveIrreg	37.18	39448	5.00	11.9%
Floating2D	36.33	135182	5.00	4.5%
Sphere	37.88	9039	5,00	10.1%
Bowling	31.84	31193	5.00	16.6%
DamBreak	151.82	2520	7.45	20.0%
Forces	70.96	6730	5.00	25.4%
WaveMake	564.26	11866	5.00	17.7%
Floating	169.47	8955	5.01	13.6%
Solids	129.18	43636	4.99	11.4%

は粒子データ配列へのアクセス時に近傍リスト内のインデックス値を用いた間接参照を行うので，近傍粒子の増加によるアクセスコストの増加量はセルリンクリスト法に比べて大きくなる．このため，近傍リスト平均長の長い 3 次元テストケースでは全体的に 2 次元の場合より性能が向上しにくくなり，平均長が比較的短い Forces ではある程度の性能向上が得られる一方で，とくに平均長の長い WaveMake では大幅な速度低下に繋がったと考えられる．

再構築回数は実行中に近傍リストの再構築が行われた回数である．再構築間隔は全ステップ数を再構築回数で除した値であり，平均して何ステップごとに再構築が行われたかを示す．2.3.2 項で述べたように，ベルレリスト法では実際の粒子の速度により，再構築までのステップ数が変化することがある．しかし Dambreak2D, Dambreak 以外のテストケースでは，その影響は約 1%以下であった．

スレッドアイドル率は，相互作用計算時にブランチダイバージェンスによりアイドルとなったスレッドの比率であり，実行全体を通してのアイドル率ではない．相互作用計算時，ワープ内の各スレッドはループを実行しながら，ループの各ステップで i 番目の近傍粒子候補について実際に相互作用を計算するか否か判定する．この際に 1 スレッドでも計算を行った場合，行わなかったスレッドをアイドルとした．各テストケースにおいて 4.5%–31.5%と大きな差がみられたが，速度向上率との相関性は見られなかった．これについては今後，オリジナル版と比較してのアイドル率の変化を評価する必要がある．

2 次元空間の WaveMake2D,WaveIrreg と 3 次元空間の WaveMake は，周期的境界条件を用いている．これはシミュレーション空間を有限サイズに限定する際に設定する境界の一つで，境界部分をもう一方の境界と繋げる手法である．境界部分の粒子は反対側の境界に影響を与えるため，周期境界粒子として登録しておき，境界付近の粒子は近傍粒子探索時に周期境界粒子も探索する．ベルレリスト法では近傍粒子の候補として，影響半径よりも広い範囲の粒子を登録する．周期的境界条件をサポートするためには周期境界粒子として登録する範囲を広げる必要があり，これらのテストケースの速度向上率が他と比較して低くなった原因と考えられる．

4.6　GPU アーキテクチャとデータ精度の影響

性能評価に用いた Tesla K20c, GeForce GTX980 はそれぞれ Kepler アーキテクチャ，Maxwell アーキテクチャを採用している．後者の方が新しい世代であるためメモリアクセス速度，コアの計算性能がともに向上しているが，倍精度演算器を持たないため倍精度演算性能は劣っている．また，データの精度が単精度と倍精度の場合を比較すると，倍精度の方が演算コスト・メモリアクセスコスト共に高くなる．

今回の評価ではほとんどのテストケースにおいて，GeForce GTX980 上で実行したほうが Tesla K20c 上の実行より高い速度向上率が得られた．これはメモリアクセス性能の向上により，ベルレリスト法のデメリットである近傍リスト構築のコストや粒子データ配列を間接参照するオーバヘッドが小さくなったことが原因と考えられる．

また，3 次元のテストケースのすべてにおいて倍精度を用いた方が単精度より速度向上率が高くなった．2 次元のテストケースでは倍精度が単精度より優位なものは半数程度であり，使用する GPU により優劣が逆転するなど，はっきりした差はみられない．

5.　まとめと今後の課題

本稿ではリスト構築時に必要なメモリを動的に確保することで, SPH 法とベルレリスト法を採用した流体解析プログラムを GPU 上に実装し，リストサイズの計算，リストの構築，リスト参照時のアクセスをデータレイアウト最適化により高速化した．また，提案した実装手法をオープンソースソフトウェア DualSPHysics に実装し，いくつかのテストケースを用いて性能評価を行った．その結果，剛体と流体などの複合的な問題のような相互作用計算にかかる処理が大きい場合に本手法が有用であることがわかった．

今後の課題として，より詳細な性能評価を行う必要がある．従来手法との優劣については原因が完全には判明しておらず，両者の間でスレッドアイドル率を比較するなど追加評価を行っていきたい．また，今回は近傍リストに登録する際の範囲 R_C を固定して評価を行ったが，これを変化させることにより，粒子密度が高い場合の性能低下を抑えられる可能性がある．さらに，本手法ではメモリ消費量が大きく増加するため，大規模なシミュレーションに適用する場合を想定し，GPU のデバイスメモリが溢れた場合の対応が必要である．

ⓒ 2017 Information Processing Society of Japan

参考文献

[1] Reeves, W. T.: Particle Systems–a Technique for Modeling a Class of Fuzzy Objects, *ACM Transactions on Graphics*, Vol. 2, No. 2, pp. 91–108 (1983).
[2] Monaghan, J. J.: Smoothed Particle Hydrodynamics, *Annual Review of Astronomy and Astrophysics*, Vol. 30, pp. 543–574 (1992).
[3] Monaghan, J. J.: Smoothed particle hydrodynamics, *Reports on Progress in Physics*, Vol. 68, No. 8, p. 1703 (2005).
[4] GPGPU.org: General-Purpose computation on Graphics Processing Units, , available from ⟨http://www.gpgpu.org/⟩ (accessed 2017-02-07).
[5] Harada, T., Koshizuka, S. and Kawaguchi, Y.: Smoothed Particle Hydrodynamics on GPUs, *Computer Graphics International*, pp. 63–70 (2007).
[6] Crespo, A. C., Domínguez, J. M., A., B., Gómez-Gesteira, M. and Rogers, B. D.: GPUs, a New Tool of Acceleration in CFD: Efficiency and Reliability on Smoothed Particle Hydrodynamics Methods, *PLoS ONE*, Vol. 6, pp. 1–13 (2011).
[7] Domínguez, J. M., Crespo, A. C., Valdez-Balderas, D., Rogers, B. D. and Gómez-Gesteira, M.: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters, *Computer Physics Communications*, Vol. 184, pp. 1848–1860 (2013).
[8] Domínguez, J. M., Crespo, A. J. C., Gómez-Gesteira, M. and Marongiu, J. C.: Neighbour lists in smoothed particle hydrodynamics, *International Journal for Numerical Methods in Fluids*, Vol. 67, No. 12, pp. 2026–2042 (2011).
[9] Green, S.: Particle Simulation using CUDA, Technical report, NVIDIA Corporation (2010).
[10] Verlet, L.: Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules, *Physical Review*, Vol. 159, pp. 98–103 (1967).
[11] Stratton, J., Anssari, N., Rodrigues, C., Sung, I., Obeid, N., Chang, L., Liu, G. and Hwu, W.: *Optimization and architecture effects on GPU computing workload performance* (2012).
[12] *NVIDIA Kepler GK110 Architecture Whitepaper*.
[13] Harris, M.: Parallel Prefix Sum (Scan) with CUDA, Technical report, NVIDIA Corporation (2007).
[14] Crespo, A. J. C., Domínguez, J. M., Rogers, B. D., Gómez-Gesteira, M., Longshaw, S., Canelas, R., Vacondio, R., Barreiro, A. and García-Feal, O.: DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH), *Computer Physics Communications*, Vol. 187, pp. 204–216 (2015).
[15] NVIDIA Corporation: *Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110* (2012).
[16] NVIDIA Corporation: *Whitepaper NVIDIA GeForce GTX 980* (2014).

ⓒ 2017 Information Processing Society of Japan