修士論文

題目

SPH法による
流体解析のGPU上での高速化

指導教員

大野 和彦 講師

2017年

三重大学大学院 工学研究科 情報工学専攻
コンピュータ・ソフトウェア研究室

高田 貴正(415M509)

内容梗概

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．

三重大学大学院　工学研究科

Abstract

The particle method is a fluid simulation method. It is widely used in physical simulations such as fluid analysis ,structural analysis ,collision analysis and so on. On the other hand, the calculation cost is increase, the simulation scale increased and the accuracy became higher. Therefore, It exposes a high degree of parallelism and has already been successfully ported to GPU. In the SPH method which is a particle methods, Particles interact only between particle within a cutoff range. The neighboring particles have to be searched every time step. As the number of particles increases, the neighboring particles search cost is increased. Verlet List method is an efficient search method. In this method, all particles have neighboring lists to store neighboring particle indices. Neighboring particles search is made efficient by referring only to this list in neighboring particles search. However, In the SPH method, the number of neighboring particles increase since the method simulate for compressible fluid. Therefore, using the Verlet List method, The access in constructing and referring to the neighbor list is increased. Moreover, it is hard to previously allocate a sufficient memory for neighboring lists. because it can not predict the size of the entire list size, In this paper, We implemented SPH-Based fluid simulations adopting the Verlet list method on the GPU by dynamically allocating the memory for neighboring lists. And optimize the data accesses when calculate list sizes, build lists and reffer lists by data layout optimization. As a result, overall execution time is reduced compared with the conventional method.

三重大学大学院　工学研究科

目 次

図 目 次

表 目 次

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]．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かかる計算コストが大きくなっている．

大量のプロセッサ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 [6]．これらのことから，GPU を用いた粒子法の高速化の研究が行われてきた．

粒子法では，各粒子は物理的に距離の近い近傍粒子とのみ相互作用する．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．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，粒子が存在する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近傍粒子を記録しておくための近傍リストをすべての粒子に持たせる．近傍リストにはカットオフ範囲内のみの粒子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．

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，GPU の主記憶であるデバイスメモリへのアクセスは 2 次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．コアレッシングアクセスによるデバイスメモリへのアクセス効率とキャッシュヒット率を以降コアレス化率とすし，コアレス化率を向上することでアクセスを高速化できる．

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 if 節のパスの各命令を実行した後に，else 節のパスの各命令を実行する．あるパスを実行中，そのパスを実行しないスレッドに対応するコア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，アイドルコアの増加は性能低下の要因になる．

2.1.1　GPU 上でのデータレイアウト最適化

GPU 上の処理で構造体型配列へアクセスするとき，アクセスパターンに合わせて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) を変更することで，アクセスを高速化できる．アクセスを効率化できる [7]．データレイアウト最適化手法として，構造体の分割とアライメントがある．

2.1.2 構造体の分割

構造体型配列の特定メンバへの連続アクセスは，構造体を分割することでコアレス化率を向上できる．各メンバは定義順にメモリ上に連続して並び，構造体型配列はこのメンバの並びが連続して繰り返される．例として，`int` 型のメンバ `x,y,z` を持つ構造体型配列のメモリ上でのデータ配置は図 2.1 のようになる．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，連続した領域へ同時にアクセスしコアレス化率が向上する．しかし，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 2.1 の場合，スレッド IDi のスレッドが `st[i].x` にアクセスしたとき，メモリアドレスが不連続なアクセスとなるためコアレス化率が低下する．そこで，構造体の配列を配列の構造体に変換することで，このようなアクセスを高速化できる．図 2.1 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 2.2 のようになる．メンバ毎に連続して並んでいるため，各スレッドの `st[i].x` へのアクセスはコアレス化率を向上できる．

2.1.3 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，または読み出しは，1, 2, 4, 8, 16byte 単位でのアクセス命令のいずれかにより実行される [8]．アライメントにより構造体配列の各要素へのアクセスを，8byte や 16byte 単位のアクセス命令でまとめて実行できる．例えば，図 2.1 のような 4byte のメンバを 3 個持つような構造体の要素にアクセスするとき，4byte 単位のアクセス命令を 3 回実行する．しかし，このような構造体を 16byte 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 16byte 書き込み命令 1 回で実行される．図 2.1 を 16byte でアライメントした構造体の配列を図 2.3 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，または読み出しを 1 命令で実行することによりアクセスを効率化できる．

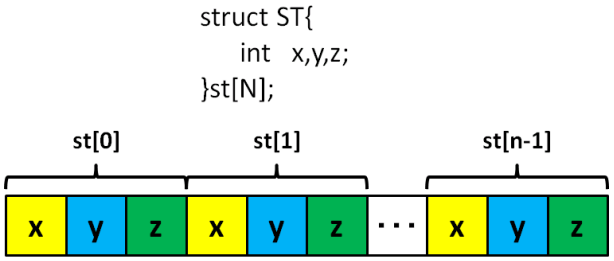

図 2.1: 構造体配列のメモリ上の配置

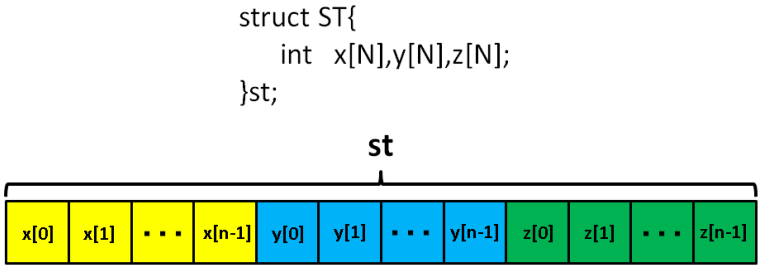

図 2.2: 配列の構造体のメモリ上の配置

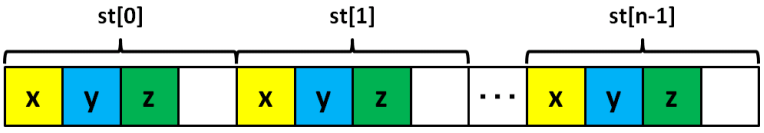

図 2.3: 16byte でアライメントした構造体配列のメモリ上の配置

2.2 SPH 法

粒子法の一つである SPH 法は，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 [2]．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 [3, 4, 5]．一般的な SPH 法の実装では，個々の粒子が位置，速度，密度，圧力などの属性をもち，GPU ではこれらの属性をメンバに持つ構造体の配列 (以降は粒子データ

配列と表記する) に格納し，粒子に個別の ID を割り振り配列の要素番号と対応させる．

SPH 法では，全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相互作用する．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の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 [9] とベルレリスト法 [10] がある．

2.2.1 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存在するかあらかじめ登録する．セルのサイズをカットオフ半径 r_C にすることで，計算対象の粒子が存在するセルとその周りのセル内の粒子を近傍粒子の候補として，候補の粒子との距離計算を行う．そして，粒子との距離が 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探索にかかるコストを削減する．セルと粒子との対応付ける手法に slide vector 法があり，セル内の粒子へ高速にアクセスできる [10]．slide vector 法では図 2.6 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とで，セル内の粒子データを参照できる．しかし，セルの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る．そのため，ステップ毎に粒子データ配列のソートおよび境界となるインデックス値の算出が必要になる．

GPU の実装では，影響範囲外の候補粒子の数だけアイドルコアが発生するという問題がある．図 2.4 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り，計算対象の粒子が存在する空間とその周りの 8 セル内の粒子を近傍粒子の候補として，候補の全粒子との距離計算を行う．このとき探索対象の粒子との距離が r_C 以下かどうかで分岐が発生し，範囲内であった場合は true となり相互作用計算し，範囲外であった場合は false となり何もしない．各スレッドで担当する粒子が異なるためブランチダイバージェンスによるアイドルコアが発生する．しかし，図 2.5 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候補粒子の数を削減できる．

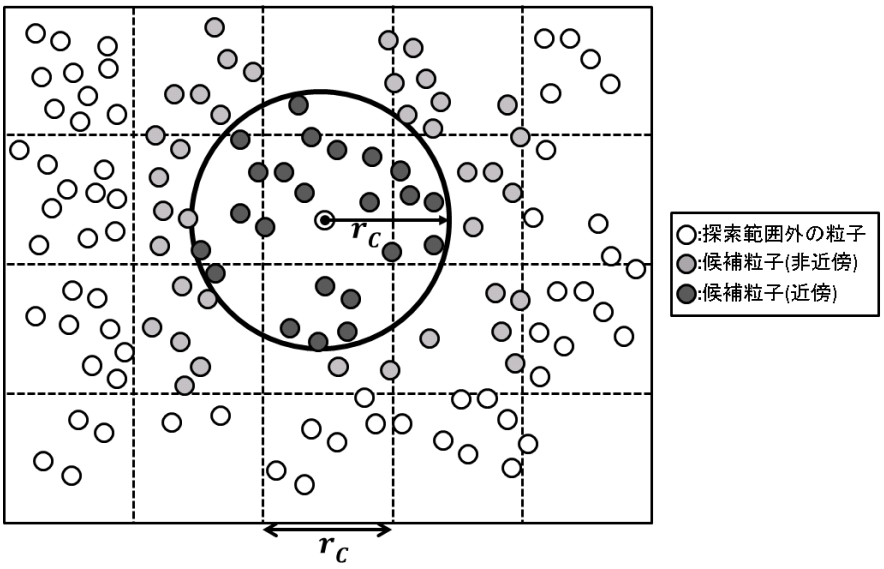

図 2.4: セルリンクリスト法を用いた場合の探索範囲

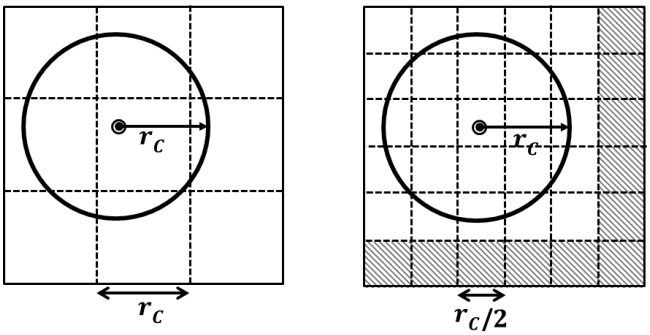

図 2.5: セルサイズの変更による探索範囲の削減

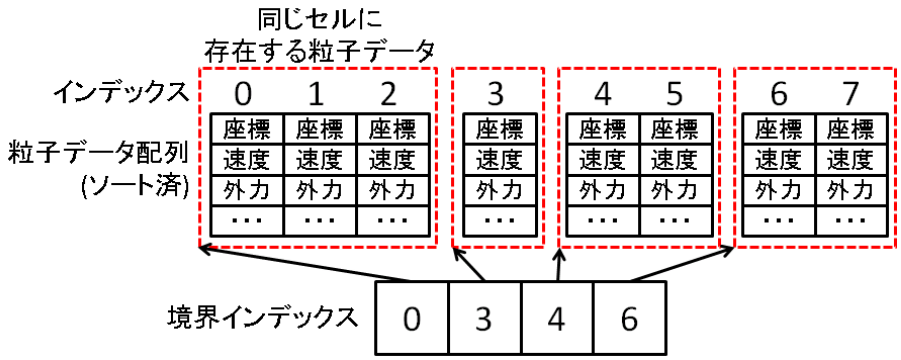

図 2.6: 粒子データ配列とセルの対応付け

2.2.2 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のみ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 2.7 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り，構築はセルリンクリスト法を用いることで高速化する．このとき近傍リストに粒子間の距離が，カットオフ半径 r_C よりも大きい 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 度に削減する．

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 [10].

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

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

ここで r_C はカットオフ半径，V_{max} は全粒子のなかでの最大速度，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 を式 (3) のように求めておき，

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

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．粒子のステップあたりの移動距離が小さくなる場合には，C ステップ以上の間リストの再構築を省略できる．

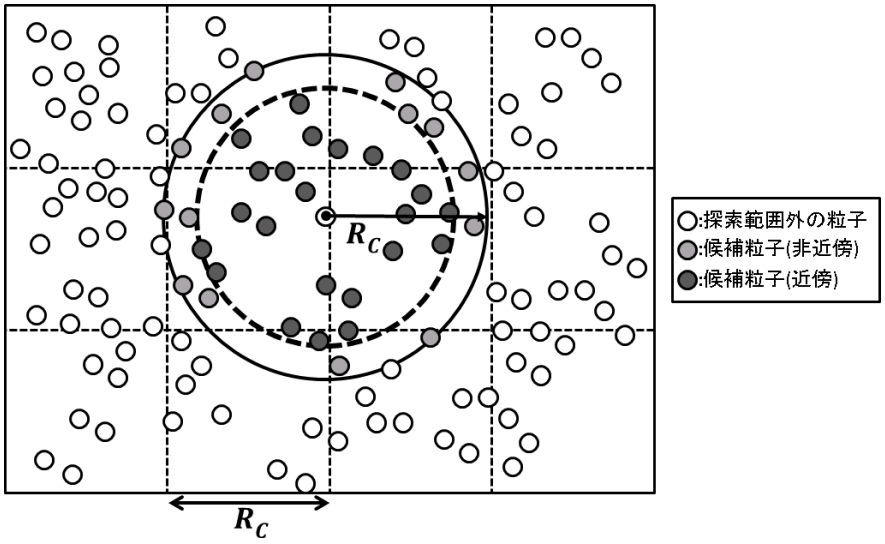

図 2.7: ベルレリスト法を用いた場合の探索範囲

3　提案手法

本稿では，SPH 法と高速化を目的とするベルレリスト法を採用した流体解析 GPU プログラムの実装手法を提案する．

しかし，SPH 法は圧縮性流体のシミュレーション手法であるため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．

3.1　リスト参照時のアクセス最適化

各近傍リストの要素を一つの配列にまとめて格納し，各要素の配置を最適化することでリスト参照時のアクセスのコアレス化率を向上する．

各粒子は近傍粒子の候補を集めた近傍リストを持つ．各スレッドは相互作用計算時に計算対象の粒子の近傍リストを先頭から順にたどる．ワープ内のスレッドはスレッド ID が連続し，各スレッドはスレッド ID に対応した近傍リストへアクセスする．ワープ内の各スレッドは担当する近傍リストの先頭要素から順に同時アクセスするため，図 3.9 のように各リストの第 1 要素のメモリアドレスが連続になるように配置し，第 2 要素以降の要素も同じように配置する．これにより，リストへのアクセス時のコアレス化率が向上する．

近傍粒子の数は粒子によって異なり，各近傍リストの要素数が異なる場合がある．このときリストの各要素を前述した配置にすると，データが格納されない空の領域が存在する．リスト毎の格納される要素数の差が大きくなる場合，空の領域が増加しメモリ消費量が大きくなる．

そこで，図 3.8 のように 32 リスト毎に各要素を連続して配置する．これによりリストに格納される要素数の差が大きい場合の影響をワープ内のスレッドが用いるリストのみに限定できメモリ消費量を削減できる．ただし，各リストの先頭要素にアクセスするために配列内の 32 リスト毎の境界のインデックスを求める必要がある．

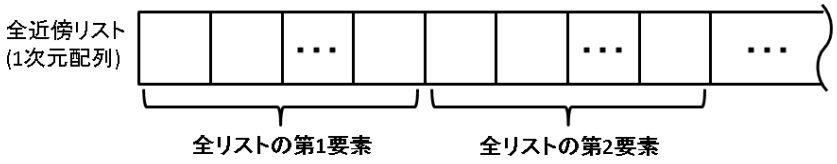

図 3.8: アクセス最適化したリストの各要素の配置

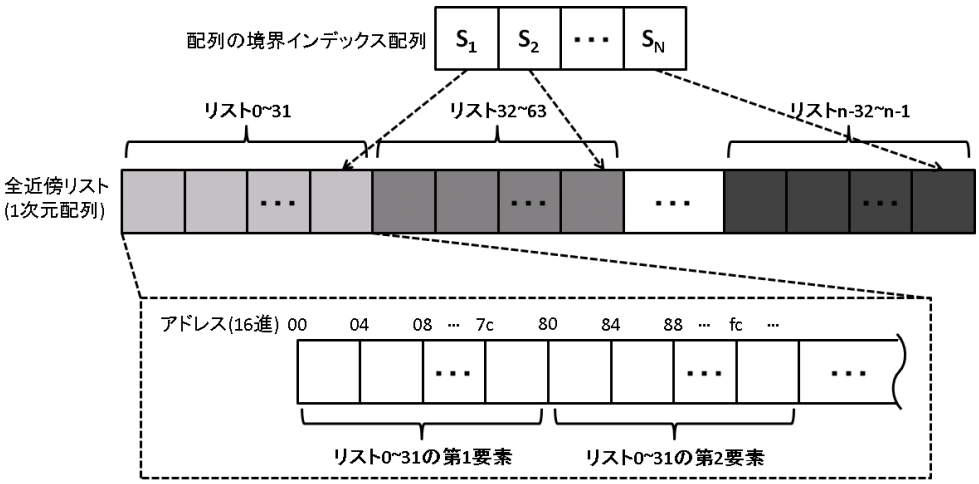

図 3.9: 最適化した近傍リストの配列上での配置

3.2 近傍リストの構築手法

近傍リスト構築では，各スレッドが担当する粒子の近傍リストを構築する．近傍リストの構築は各リストの要素数の算出，全体のリストサイズの計算，近傍リストの領域確保，リストへの近傍粒子の登録の 4 段階の手順で行う．この手法では，リストの要素数の算出時とリストへの近傍粒子の登録時に，セルリンクリストを用いた近傍粒子探索を行う．リストの要素数の算出時には，各近傍リストに登録する要素数 (近傍粒子の数) を求める．そして各リストの要素数の合計し，全近傍リストサイズと求める．リストへの近傍粒子の登録時には確保したメモリに対して近傍粒子のインデックスを登録していく．近傍粒子の数および近傍粒子のインデックスを求める際に近傍粒子探索が必要になる．つまり，一度のリスト構築でセルリンクリストを用いた近傍粒子探索を 2 度行う．

3.3　近傍リストサイズの算出

全近傍リストのサイズ計算時に，図 3.8 に示したようにリストサイズはスレッドが所属するワープ内の最大サイズに統一する必要がある．そのため，各ワープの最大リストサイズを求め，その値を 32 倍したものがワープ内の全近傍リストサイズの合計になる．図 3.10 に各スレッドが近傍リストサイズを求めた後から，近傍リストの全体サイズを計算するまでの全体の流れを示す．

各ワープのリストサイズの最大値の計算は Warp Shuffle 命令を用いる [8]．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，同期の必要がなくレジスタ間で直接データを交換できるので高速に最大値を計算できる．

次に，各ワープの最大リストサイズの合計は，並列プレフィックスサム (`inclusive_scan`) によって求める [11]．各ワープの最大リストサイズを格納した配列のプレフィックスサムを求め，これによって得られた配列の最後の要素が全近傍リストサイズの合計値となる．また，この配列は図 3.8 での配列の境界インデックス配列となる．

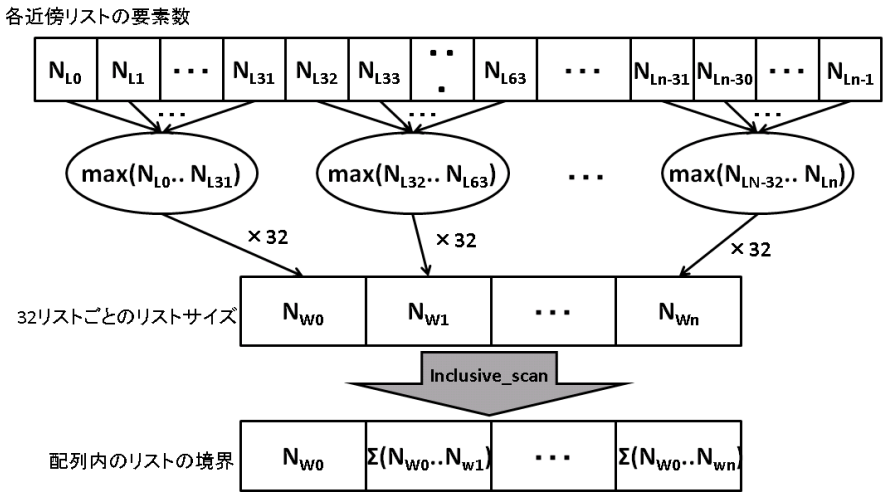

図 3.10: 配列上の境界値の算出

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へのアクセスを 2 度行う．このとき，図 2.7 のように各セルのサイズを図 2.4 より大きくし，近傍粒子候補の数が増加し全体のアクセス回数が増加するため，これらの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なので，必要になる粒子の属性は位置座標 (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位置座標 (x,y,z) が連続してメモリ上に並んでいれば高速にアクセスできる．そのために，粒子の実データを格納するための構造体のメンバのうち，位置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位置座標メンバに対して高速にアクセスできる．各メンバが単精度であれば位置座標 (x,y,z) の 3 メンバを持つ構造体を 16byte でアライメントし，倍精度であれば位置座標 (x,y) の 2 メンバを持つ構造体を 8byte でアライメントしたものと (z) の配列に分割すれば，位置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 に実装し，ソースに付属しているテストケースを用いて，実装前後の実行時間およびメモリ消費量の比較を行った．

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 [12]．本プログラムは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り高速化されている．DualSPHysics_v4.0 ではセルリンクリスト法を実装している．

本評価では提案した実装手法により DualSPHysics_v4.0 にベルレリスト法を適用し，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ベルレリスト法の実装前後の実行時間の比較を行った．本実験に用いたテストケースの概要を表 4.1 に示す．

評価は表 4.2 に示す 2 種類の環境で行った．環境 1 で用いた GPU Tesla K20c は第 3 世代 Kepler アーキテクチャ，環境 2 で用いた GPU GeForce GTX980 は第 4 世代 Maxwell アーキテクチャを採用している [13, 14]．

表 4.1: 本実験に用いたテストケース

TestCase	空間	概要
DamBreak2D	2D	ダム崩壊シミュレーション
MoveSquare	2D	正方形の剛体が一定速度で水の中を進む
Sloshing	2D	水の入ったタンクを回転させる
WaveMake2D	2D	一定周期の波を生成する
WaveIrreg	2D	不規則な波を生成する
Floating2D	2D	箱を浮かべた水に波を生成する
Sphere	2D	水の入ったタンクに球体を落とす
Bowling	2D	積み上げた箱に, 斜面を転がした球体を衝突させる
DamBreak	3D	ダム崩壊シミュレーション
Forces	3D	直接流体に外力を加え, タンク内の水をかき回す
WaveMake	3D	一定周期の波を生成する
Floating	3D	箱を浮かべた水に波を生成する
Slolids	3D	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 4.2: 評価環境

評価環境	CPU	メモリ	GPU
1	Intel Core i7-930	6GB	Tesla K20c
2	Intel Xeon CPU E5-1620	16GB	GeForce GTX980

4.2 実行時間の比較

各テストケースでのベルレリスト法の実装の有無による実行時間を単精度 (float) と倍精度 (double) を用いた 2 通り計測した．評価環境 1 でのベルレリスト法の実装無に対する実装有の実行速度向上率を図 4.11, 図 4.12 に示す．

つぎに，評価環境 2 でのベルレリスト法の実装無に対する実装有の実行速度向上率を図 4.13, 図 4.14 に示す．図 4.11, 図 4.13 は 2 次元空間でのテストケース，図 4.12, 図 4.14 は 3 次元空間でのテストケースの速度向上率である．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る速度向上を実現している．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いる．

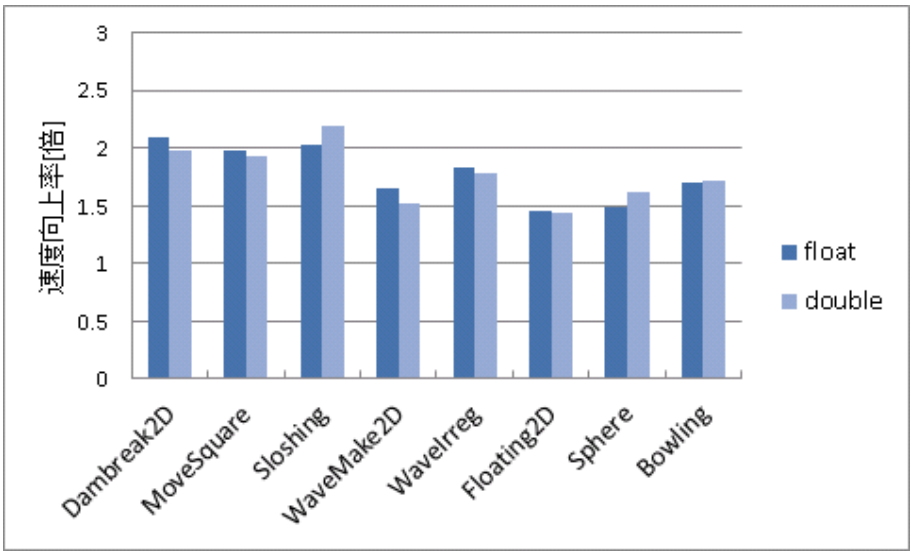

図 4.11: Tesla K20c の 2D テストケースでの速度向上率

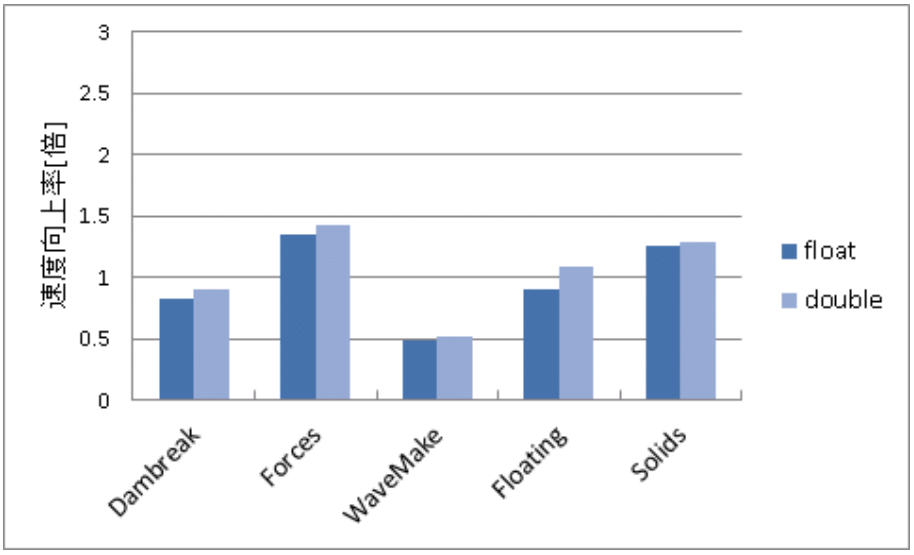

図 4.12: Tesla K20c の 3D テストケースでの速度向上率

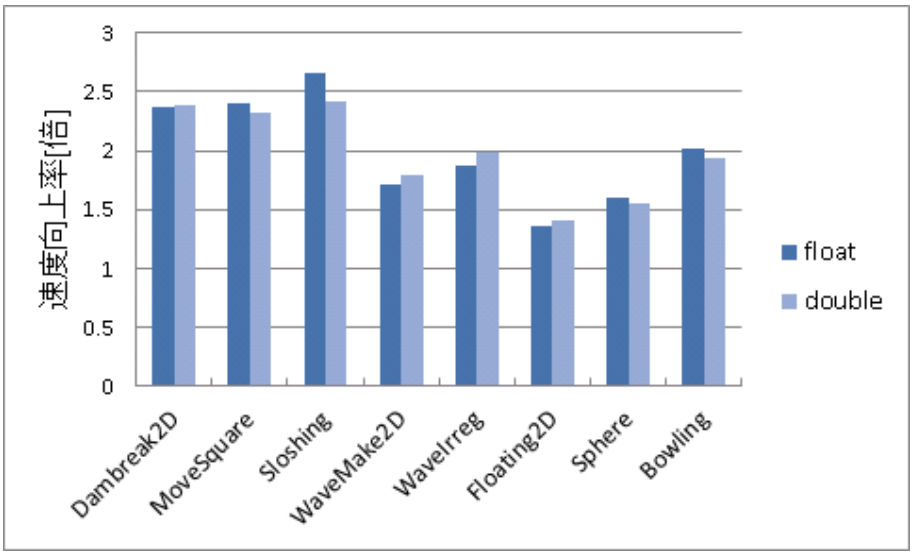

図 4.13: GeForce 980 の 2D テストケースでの速度向上率

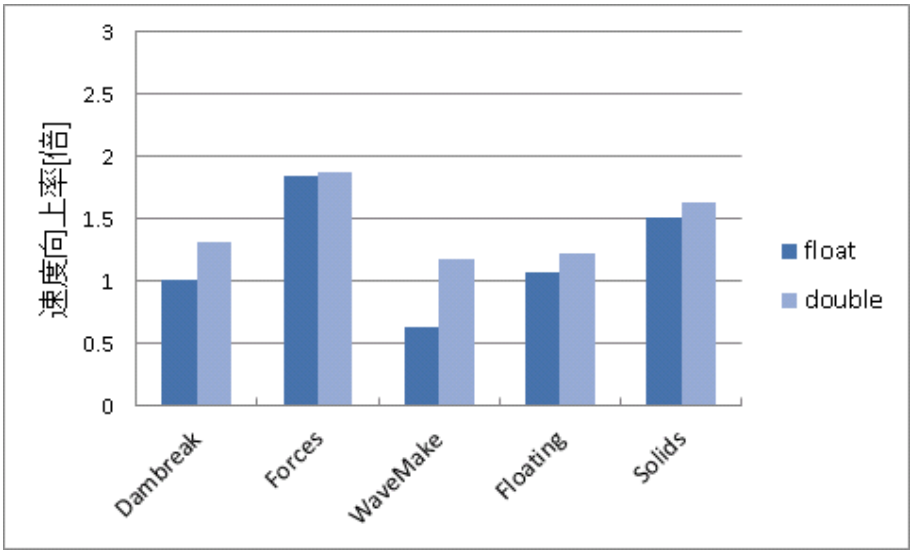

図 4.14: GeForce 980 の 3D テストケースでの速度向上率

4.3　メモリ消費量の比較

各テストケースでのベルレリスト法の実装無に対する実装有のメモリ消費量の増加率を図 4.15，図 4.16 に示す．図 4.15 は 2 次元空間でのテストケース，図 4.16 は 3 次元空間でのテストケースのメモリ消費量の増加率である．全てのテストケースでメモリ消費量が増加しており，3 次元空間のテストケースではメモリ消費量が大きく増加している．

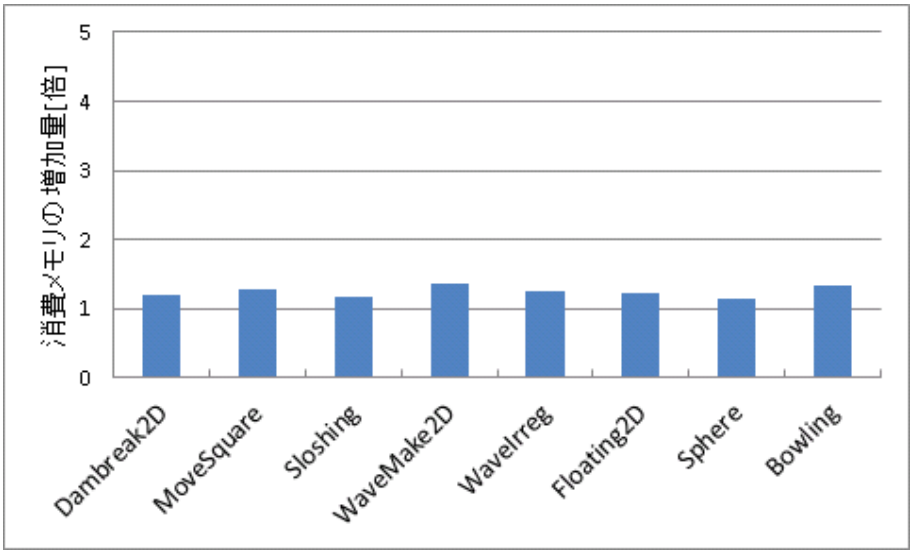

図 4.15: 2D テストケースでのメモリ消費量の増加率

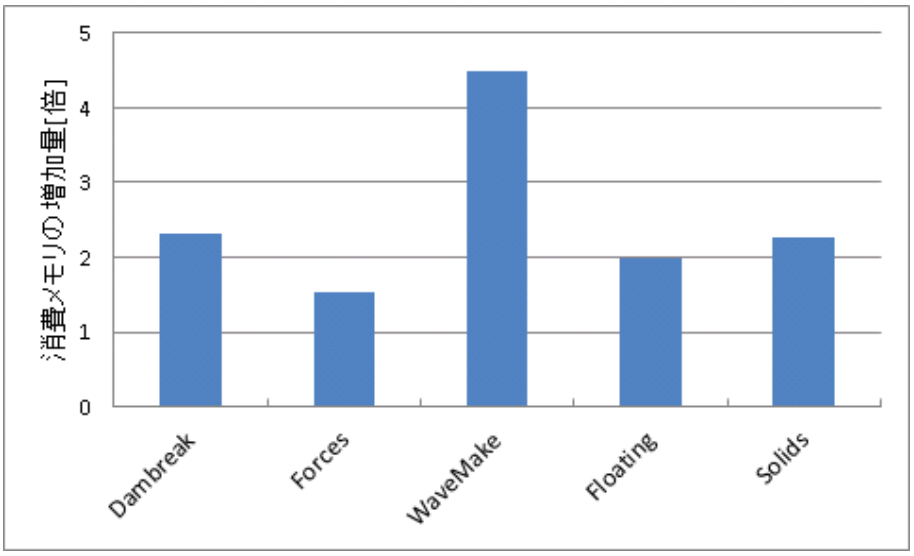

図 4.16: 3D テストケースでのメモリ消費量の増加率

5　考察

5.1　テストケースによる性能への影響

図 4.11，図 4.13 の 2 次元空間での評価では全てのテストケースにおいて提案手法により実行時間を短縮できた．速度向上の大きな要因として，リスト構築およびソート処理のコストの削減が考えられる．表 5.3 に評価環境 2 での DamBreak2D の単精度を用いた場合の各処理の実行時間を示す．セルリンクリスト法を用いた場合，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．対してベルレリスト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のでその分のソート回数も削減できる．2 次元空間では近傍粒子の数が比較的少なく，粒子間の相互作用計算にかかる処理が小さくなる．そのため，相互作用計算以外の処理が実行時間を占める割合が多くなり，ソート処理削減の影響が大きく出たと考えられる．

表 5.3: 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
粒子データ配列のソート	44.0	4.2
近傍リスト構築	–	4.1
相互作用計算	12.4	10.5
その他	16.0	11.7
全体の実行時間	72.4	30.5

つぎに，図 4.12，図 4.14 の 3 次元空間での評価ではテストケースによって大きく性能に差が出ている．ベルレリスト法の性能低下の大きな要因として 2 点考えられる．1 点目はシミュレーション時の全体の密度が高い場合である．ベルレリスト法では粒子の密度が高くなると近傍リストに記録する粒子の数が増加する．探索時の粒子データ配列へのアクセスは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．本実験で用いたテストケースの中では，DamBreak および WaveMake の性能低下が確認できた．図 4.16 でメモリ消費量の増加率が高いため，これらのテストケースでは粒子の密度が高くなり，性能低下につながったと考えられる．

2 点目は相互作用計算の処理量が小さい場合である．GeForce GTX980 を搭載した計算機で提案手法を用いた場合に DamBreak では速度低下が確認できた．DamBreak での各処理の実行時間を表 5.4 に示す．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みの相互作用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．そのため，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く性能低下につながったと考えられる．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．Solids の各処理の実行時間を表 5.5 に示す．アイドルスレッドの削減による効果が大きく相互作用計算の実行時間を大幅に短縮できている．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．

表 5.4: 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
粒子データ配列のソート	17.6	2.2
近傍リスト構築	–	26.4
相互作用計算	74.3	65.0
その他	12.0	9.0
全体の実行時間	103.9	102.6

表 5.5: 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
粒子データ配列のソート	57.0	6.9
近傍リスト構築	–	55.8
相互作用計算	350.5	223.6
その他	38.7	28.2
全体の実行時間	458.2	323.5

最後に，2 次元空間の WaveMake2D,WaveIrreg と 3 次元空間の WaveMake は他のテストケースと比較して性能が低下している．これらのテス

トケース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，この境界粒子の増加による影響が性能低下につながったと考えられる．しかし，Floating でも周期的境界条件を用いているが性能向上が確認できた．これは Floating が流体と剛体の複合問題でありベルレリスト法による処理時間の短縮が大きくなり，周期的境界粒子の増加による影響を上回ったからだと考えられる．

5.2 GPU のアーキテクチャによる性能への影響

表 4.2 の評価環境 1,2 では異なる GPU を用いており，環境 2 の方が提案手法により性能を大きく向上できた．

評価環境 1 の GPU Tesla K20c は Kepler アーキテクチャ，評価環境 2 の GPU GeForce GTX980 は Maxwell アーキテクチャを採用している．Maxwell は Kepler より新しいアーキテクチャであり，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．ベルレリスト法では近傍リストを用い粒子データ配列を間接参照するためメモリアクセスが遅くなる．しかし，アイドルスレッドの削減できコアの稼働率を向上できる．そのため，GPU のメモリアクセス性能およびコアの計算性能の向上は間接参照による速度低下を抑え，コア稼働率の改善による速度向上が大きくなったと考えられる．

5.3 単精度と倍精度による性能への影響

単精度から倍精度に変更した場合の性能へ影響を与える大きな要因として，演算コストとメモリアクセスコストの 2 点が挙げられる．倍精度を用いた場合，単精度に比べて演算およびアクセスにかかるコストが増加する．ベルレリスト法では相互作用時の計算量が大きくなると，性能を大きく向上できたためこのような結果になったと考えられる．

6　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の GPU 上での実装し, 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た．そして，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提案手法の性能評価を行った．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，本研究における実験では近傍リストに登録する際の範囲 R_C を変えたときの検証を行わなかった．粒子の密度が大きくなる場合，性能低下が確認できたが R_C を変えた場合に性能低下を抑えられるかの検証が必要である．また，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，GPU のデバイスメモリが不足した場合の対応が必要になる．

謝辞

本研究を行うにあたり，御指導，御助言頂きました大野和彦講師，並びに多くの助言を頂きました山田俊行講師に深く感謝致します．また，様々な局面にてお世話になりました研究室の皆様にも心より感謝いたします．

参考文献

[1] W. T. Reeves, *Particle Systems - a Technique for Modeling a Class of Fuzzy Objects*. ACM Transactions on Graphics, Vol. 2, pp. 359-375, 1983.

[2] J. J. Monaghan, *Smoothed particle hydrodynamics*. Annual Review of Astronomy and Astrophysics, Vol. 30, pp. 543-574, 1992.

[3] L. D. Libersky, A. G. Petschek, T. C. Carney, J. R. Hipp and F. A. Allandadi, *High strainLagrangian Hydrodynamics: A three-dimensional SPH code for dynamic material response*. Journal of computational Physics, Vol. 109, pp. 67-75,1993.

[4] J. J. Monaghan, A. Kos and M. Issa, *Fluid motion generated by impact*. Journal ofWaterway, Port, Coastal and Ocean Engineering, Vol. 129, pp. 250-259, 2003.

[5] R. B. Canelas, A. J. C. Crespo, J. M. Domnguez, R. M. L. Ferreira and M. Gmez-Gesteira, *SPH-DEM model for arbitrary geometries in free surface solid-fluid flows*. Computer Physics Communications, Vol. 202, pp. 131-140, 2016.

[6] *GPGPU.org: General-Purpose computation on Graphics Processing Units*, `http://www.gpgpu.org/` (2017-2-7).

[7] J. A. Stratton, N. Anssari, C. Rodrigues, I. Sung, N. Obeid, L. Chang, G. D. Liu and W. Hwu, *Optimization and architecture effects on GPU computing workload performance*. In INPAR: Workshop on Innovative Parallel Computing. IEEE, 2012.

[8] *CUDA C Programming Guide*, `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`, (2017-2-7).

[9] A. J. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gomez-Gesteira and B. D. Rogers, *a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods*. PLoS ONE, 6(6), 2011.

[10] J. M. Dominguez, A. J. C. Crespo, M. Gomez-Gesteira and J. C. Marongiu, *Neighbour lists in Smoothed Particle Hydrodynamics*. International Journal for Numerical Methods in Fluids, Vol. 67, issue 12, pp. 2026-2042, 2011.

[11] M.Harris. *Parallel Prefix Sum (Scan) with CUDA*,NVIDIA technical report,2007.

[12] A. J. C. Crespo, J. M. Domnguez, B. D. Rogers, M. Gmez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro and O.Garca-Feal, *DualSPHysics: open-source parallel CFD solver on Smoothed Particle Hydrodynamics (SPH)*. Computer Physics Communications, vol. 187, pp.204-216, 2015.

[13] NVIDIA.*NVIDIA Kepler GK110 Architecture Whitepaper.* `https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`, (2017-2-7).

[14] NVIDIA.*NVIDIA GeForce GTX 980* `http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF`, (2017-2-7).