修士論文

題目

GPUにおける
粒子シミュレーションの
細分化セルを用いた高速化手法

指導教員

大野 和彦 講師

2019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

新田 知生 (417M515)

内容梗概

Abstract

目 次

図 目 次

表 目 次

1　はじめに

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 スレッドは同じ命令を実行する SIMD 型の並列処理を行う．

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 L2 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スアクセスという．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.

分岐条件によりワープ内のスレッドが異なる命令を実行する場合，それぞれの命令を逐次実行する．これをブランチダイバージェンスといい，実行速度低下の要因となる．

2.2　SPH 法

粒子法の一つである SPH 法は，元々は宇宙物理学の問題を解くために Monaghan 等によって提案された手法である [1]．しかし，SPH 法は幅広く応用が利く手法であるため，圧縮性流体や非圧縮性流体，構造解析などに使用されている [2],[3],[4]．一般的な SPH 法の実装では，各粒子が位置，密度，圧力などの属性を持ち，GPU ではこれらの属性をメンバに持つ構造体の配列に格納し，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でのみ相互作用する．粒子は空間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 [5] とベルレリスト法 [6] がある．

2.2.1　セルリンクリスト法 (CLL)

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近傍 8 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 [6]．slide vector 法では図に示すように，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．

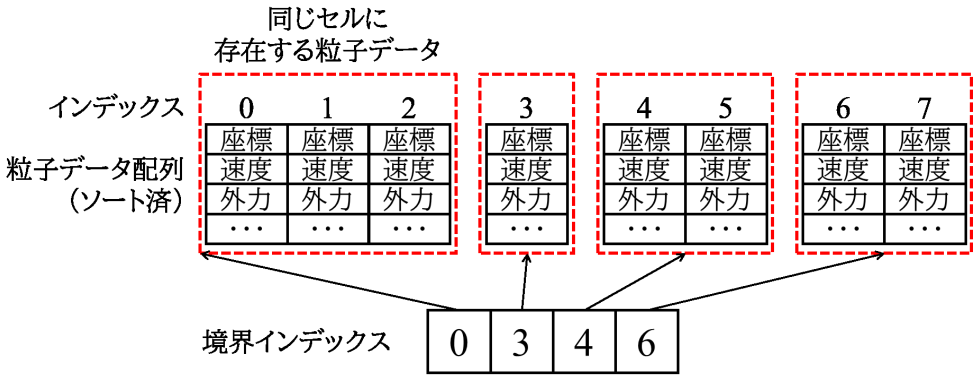

図 2.1: 削減前，及び $Cube, Sphere$ の探索範囲セル (3D)($k = 2$)

2.2.2　ベルレリスト法 (VL)

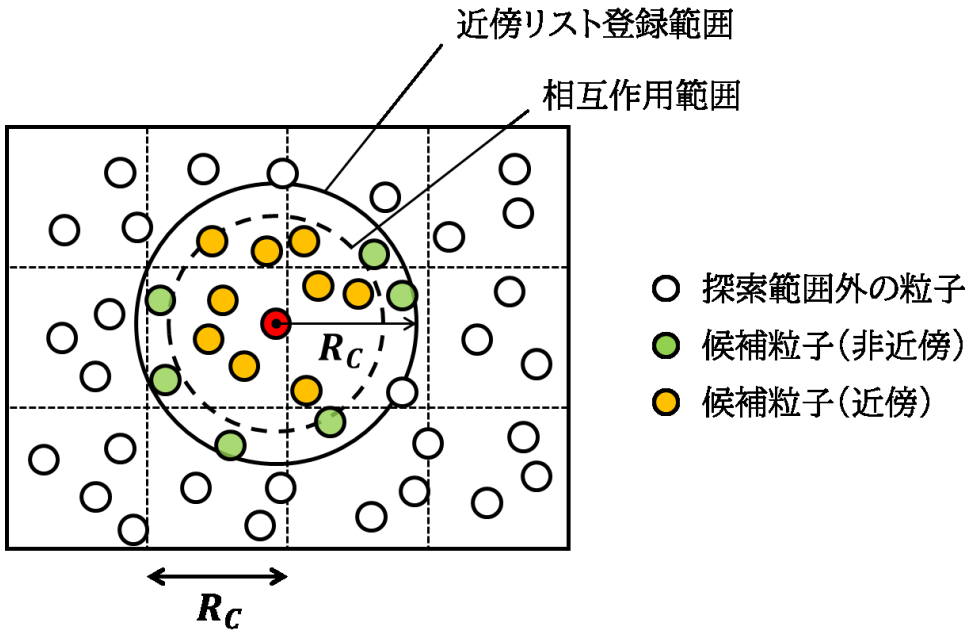

図 2.2: 削減前，及び $Cube, Sphere$ の探索範囲セル (3D)($k = 2$)

3　提案手法

先行研究では，CLL 法とデータレイアウト最適化を行った VL 法の比較が行われた [7]．結果，セル内の粒子密度が高いケースで十分な速度向上が得られず，近傍リスト構築がボトルネックとされていた．

本論文では，高密度なケースの近傍リスト構築にかかるコスト削減を目的とし，細分化セルを用いた高速化手法を提案する．セルの細分化により，近傍リスト構築の探索範囲をより限定できるため，処理削減が可能である．また，セル内の粒子間距離が小さくなり，同一ワープ内で処理される粒子が持つ近傍リストの類似率が高くなる．類似率向上により，キャッシュヒット率の増加が期待できる．一方でセル内の粒子数が減少するため，同一ワープで処理される粒子が複数のセルにまたがってしまい，類似率が低下する．そこでキャッシュヒット率の向上を目的とした，近傍リスト構築における粒子登録順序の最適化を行った．

3.1　セルの細分化

一般的に VL 法では，セルサイズ s_c は近傍リスト登録半径 R_C(CLL 法では r_C) と同じ値が用いられる．本手法において，s_c を式 (1) で定義し，細分化数 k は $0, 1, 2, 3$ とする．

$$s_c = \frac{R_C}{2^k} (k = 0, 1, 2, 3) \tag{1}$$

3.1.1　近傍リスト構築アルゴリズム

近傍リスト構築についてのアルゴリズムを Algorithm1 に示す．また，Algorithm1 に対応した変数が指すセル座標を図 3.3 に示す．$CalculateCellID$ はセル座標 x, y, z からセル ID を求める関数である．セル ID を添字とし，セルに含まれる粒子の最小・最大 ID である $BeginEndInCell[i].x, BeginEndInCell[i].y$ を $pini, pfin$ に代入する．その後，$pini$ から $pfin$ までの粒子に対して，近傍リスト登録の判定を行う．図 3.3 の一辺のセル数 n は式 (2)，総セル数 n_o は式 (3) で求まる．

$$n = 2^k \times 3 \tag{2}$$

$$n_o = n^3 \tag{3}$$

Algorithm 1 近傍リスト構築

```
for z ← zini to zfin do
  for y ← yini to yfin do
    for x ← xini to xfin do
      i ← CalculateCellID(x, y, z)
      if x = xini then
        pini ← BeginEndInCell[i].x
      end if
      pfin ← BeginEndInCell[i].y
    end for
    for p ← pini to pfin do
      粒子 [p] を近傍リストに登録するか判定
    end for
  end for
end for
```

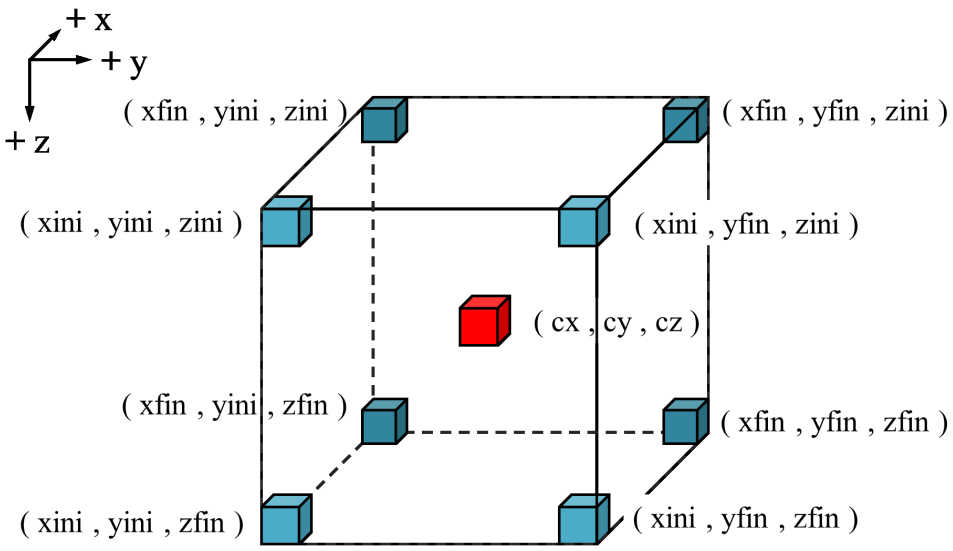

図 3.3: Algorithm1 に対応したセル座標 (3D)

3.1.2　探索セルの削減

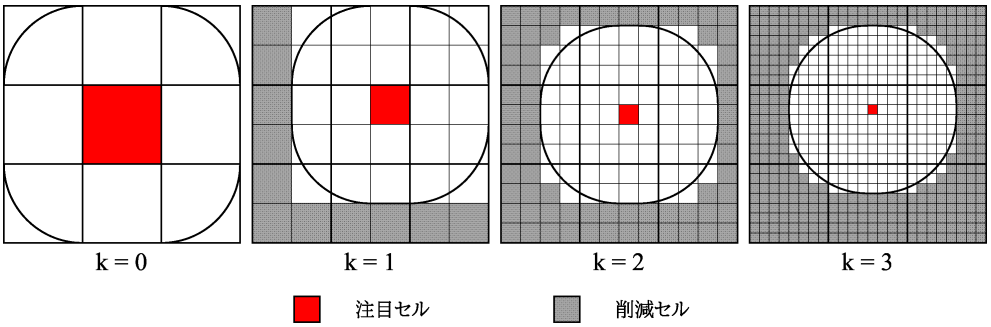

図 3.4: セルの探索範囲と削減可能セル (2D)($k = 0, 1, 2, 3$)

セル細分化により，削減できるセルを図 3.4 に示す．図の曲線は，注目セルに存在する粒子が，近傍リストを構築するために探索する範囲の限界を表している．k の値が大きくなるに伴い，探索の限界範囲が小さくなり，削減可能なセルが多くなる．一方で，探索範囲が球体に近似されるため，完全な削減では実装コストが大きくなる．そこで，本手法では，以下 2 種類の削減パターンを実装した．それぞれのパターンで削減された探索範囲，及び削減前の探索範囲を図 3.5 に示す．

A) *Cube*
セルの探索範囲が立方体になるよう，部分的にセルを削減する．削減された探索範囲の一辺のセル数 n は式 (4)，総セル数 n_r は式 (5) で求まる．完全な削減ではないが，容易に実装できるため低コストである．

$$n = 2^{k+1} + 1 \tag{4}$$

$$n_r = n^3 \tag{5}$$

B) *Sphere*
この削減パターンでは，*Cube* の探索範囲から削減可能なセルを完全に取り除く．近傍リスト構築の探索コストを最小にできるが，形状が複雑なため実装コストが大きくなる．

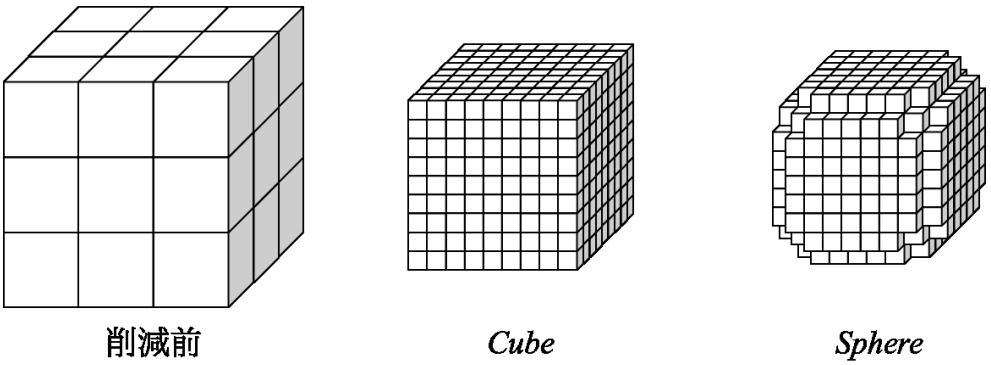

図 3.5: 削減前，及び $Cube, Sphere$ の探索範囲セル (3D)($k = 2$)

削減前と $Cube, Sphere$ による削減後の探索セル数 n_o, n_r，及び探索範囲の割合 S_R を表 3.1 に示す．S_R は式 (6) で求まる．

$$S_R = \frac{n_r}{n_o} \tag{6}$$

表 3.1: $k = 0, 1, 2, 3$ の探索セル数と比率

k		0	1	2	3
n_o		27	216	1728	13824
$Cube$	n_r	27	125	729	4913
	S_R	1	0.58	0.42	0.36
$Sphere$	n_r	27	125	613	3449
	S_R	1	0.58	0.36	0.25

3.1.3 実装

$Cube$ の近傍リスト構築アルゴリズムは，Algorithm2 に示すように，x, y, z 方向の ループ範囲を変更することで実装できる．シミュレーション空間上の境界面と探索範囲が交差する場合を考慮し，探索範囲の一辺のセル数は，2^k から 2^{k+1} の範囲となる．

Algorithm 2 近傍リスト構築

```
for z ← zini to zfin (2^k ≤ zfin − zini ≤ 2^{k+1}) do
  for y ← yini to yfin (2^k ≤ yfin − yini ≤ 2^{k+1}) do
    for x ← xini to xfin (2^k ≤ xfin − xini ≤ 2^{k+1}) do
      i ← CalculateCellID(x, y, z)
      if x = xini then
        pini ← BeginEndInCell[i].x
      end if
      pfin ← BeginEndInCell[i].y
    end for
    for p ← pini to pfin do
      粒子 [p] を近傍リストに登録するか判定
    end for
  end for
end for
```

Sphere は，Algorithm2 の $xini, xfin$ の値を増減することで実装できる．しかし，シミュレーション空間の境界面と探索範囲が交差する場合があり，増減値を一意に決めることが出来ない．また，並列処理において分岐命令で発生するブランチダイバージェンスは実行速度低下の要因となる．そこで xy, yz, zx 平面と探索空間が交差する場合に分岐命令が発生しない実装を行った．図 3.6, 3.7, 3.10 には，隣り合うセルに所属する粒子を同一ワープ内のスレッドが処理する場合に，境界面と探索範囲が交差することで生じる増減値 t の違いを示す．

A ）xy 平面，zx 平面

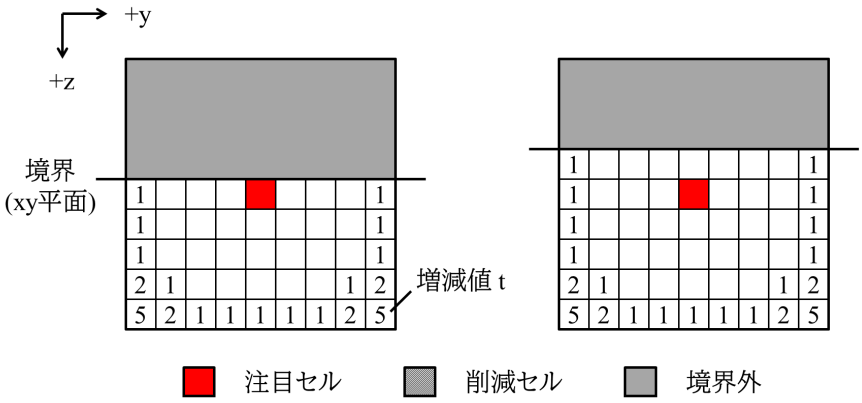

図 3.6: xy 平面と探索範囲の交差 (k=2)

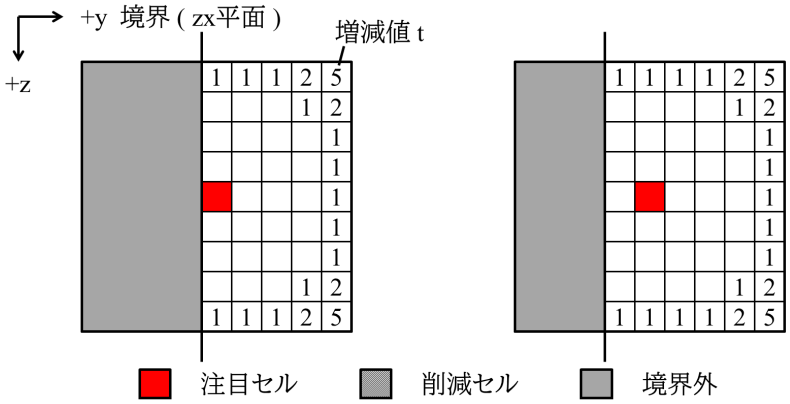

図 3.7: zx 平面と探索範囲の交差 (k=2)

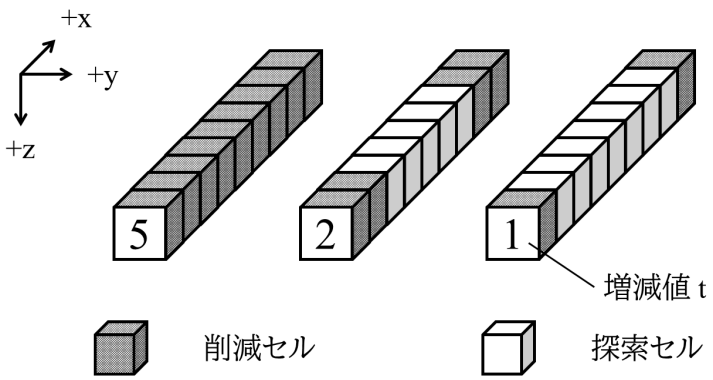

図 3.8: zx 平面と探索範囲の交差 (k=2)

図 3.6, 3.7 は，注目セルに存在する粒子の探索セルを表しており，セルに記載されている数字は，図 3.8 のように $xini, xfin$ からそれぞれ $+x, -x$ 方向に 削減できるセルの個数である．また，記載されていないセルは削減できないことを表している．境界内である探索範囲の左上セルからアクセスするため，同時にアクセスするセルの増減値は異なるが，注目セルから相対的位置にあるセルの増減値は同じである．そこで図 3.9 のように，増減値を記録した $CutTable$ 配列をコンスタントメモリに作成し，式 (7) で t を求めた．

```
CutTable[81]={  5, 2, 1, 1, 1, 1, 1, 2, 5,
                2, 1, 0, 0, 0, 0, 0, 1, 2,
                1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 1,
                2, 1, 0, 0, 0, 0, 0, 1, 2,
                5, 2, 1, 1, 1, 1, 1, 2, 5   };
```

図 3.9: 増減値を記録した配列 (k=2)

$$t = CutTable[(4 - (cz - z)) \times 9 + (4 - (cy - y))] \tag{7}$$

B) yz 平面

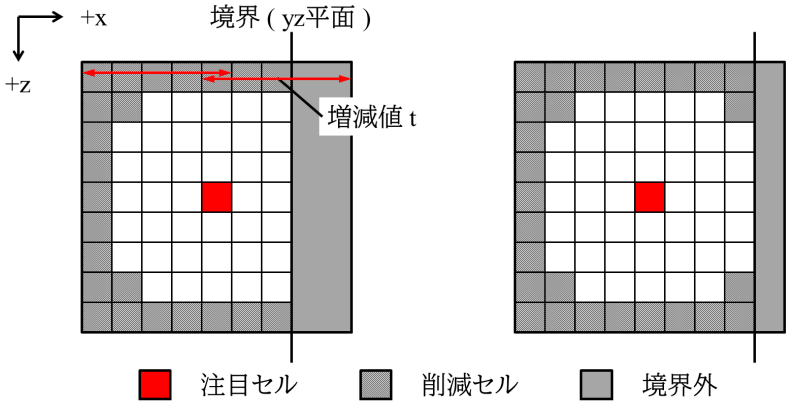

図 3.10: yz 平面と探索範囲の交差 (k=2,y=yini)

図は $y = yini$ の探索範囲である．この場合 $xfin$ からの増減値が小さくなり，また図 3.10 の左右で x 軸方向に境界外に出ているセル数が異なる．よって注目セルの座標から境界面までのセル数を，各スレッドが計算する必要がある．スレッドによって計算結果が異なるため，ブランチダイバージェンスが発生しないよう Algorithm3 の 5,6 行目で，条件式を用いず t を再計算した．4 行目で分岐命令を用いているが，真の分岐先のみなのでブランチダイバージェンスは起こらない．

Algorithm 3 近傍リスト構築

```
1: for z ← zini to zfin (2^k ≤ zfin − zini ≤ 2^{k+1}) do
2:   for y ← yini to yfin (2^k ≤ yfin − yini ≤ 2^{k+1}) do
3:     t = CutTable[(4 − (cz − z)) × 9 + (4 − (cy − y))]
4:     if t < 5 then
5:       xini = xini + (t − (4 − (cx − xini)))
6:       xfin = xfin − (t − (4 − (xfin − cx))
7:       for x ← xini to xfin do
8:         i ← CalculateCellID(x, y, z)
9:         if x = xini then
10:          pini ← BeginEndInCell[i].x
11:        end if
12:        pfin ← BeginEndInCell[i].y
13:      end for
14:      for p ← pini to pfin do
15:        粒子 [p] を近傍リストに登録するか判定
16:      end for
17:    end if
18:  end for
19: end for
```

以上がブランチダイバージェンスを回避した，探索範囲削減の実装である．

3.2 近傍リスト最適化

3.2.1 近傍リストの類似率

各粒子の近傍リストを構築した後，数ステップの間近傍リストに登録されている粒子と相互作用の判定を行う．判定対象である粒子データは，近傍リストから参照した粒子 ID を用いてメモリ上からロードされる．このロードは L1 キャッシュを通じて行われる．つまり，スレッドが粒子データをロードした後，同じワープ内のスレッドで必要な粒子データがキャッシュメモリに存在する場合，キャッシュヒットにより高速にアクセスできる．よって，同ワープ内で処理される粒子の近傍リストに登録される粒子の順序は，キャッシュヒット率に影響する．

図 3.11, 3.12 はセル細分化前後の，同一ワープ内で処理される粒子が同じセルに存在する場合に，粒子データをロードする図である．粒子 n と $n+1$ の，近傍リストの総登録範囲に対する重複範囲を類似率とする．

細分化前では類似率が低く，重複範囲内に存在する粒子の相互作用判定をスレッド n は後半に行うが，スレッド $n+1$ では前半に行われる．スレッド n が共通範囲内の粒子データをロードする時，スレッド $n+1$ は離れたアドレス先の粒子データをロードするためキャッシュヒット率が低い．

一方細分化後は類似率が高く，重複範囲内の粒子に対する判定処理は，近いタイミングで行われる．またスレッド n のロードでキャッシュに記録された粒子データの中に，スレッド $n+1$ が必要とするデータが存在する確率が高い．よって細分化によりキャッシュヒット率の向上が期待できる．しかし，細分化によりセル内の粒子数は減少するため，同ワープ内のスレッドが処理する粒子が複数のセルに分布する場合がある．図 3.13 のように近傍リスト登録範囲は x 軸方向に広がるため，32 スレッドとしての類似率が低くなる．そこで，類似率が低い場合でもキャッシュヒットが起こるよう，重複範囲内のセルに優先的にアクセスすることで，近傍リストの粒子登録順序を最適化した．

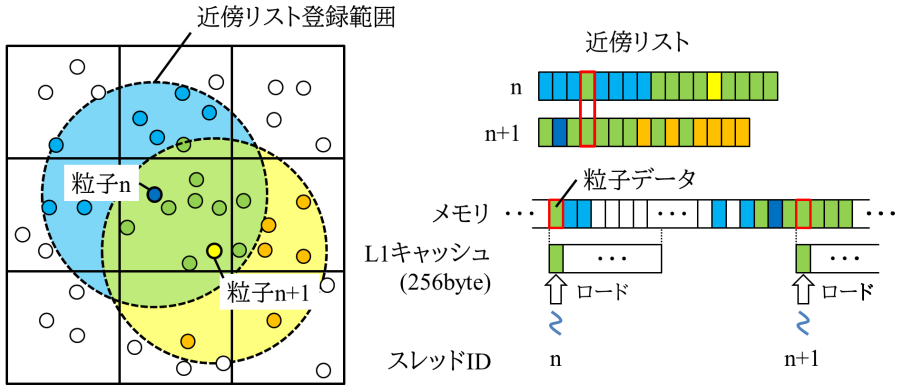

図 3.11: yz 平面と探索範囲の交差 (k=2,y=yini)

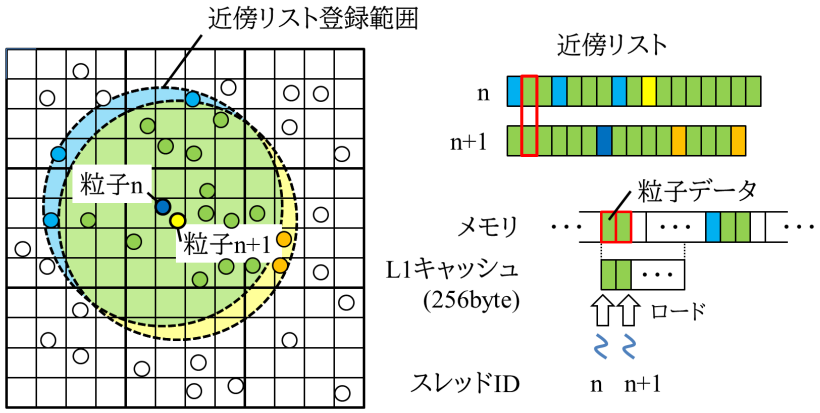

図 3.12: yz 平面と探索範囲の交差 (k=2,y=yini)

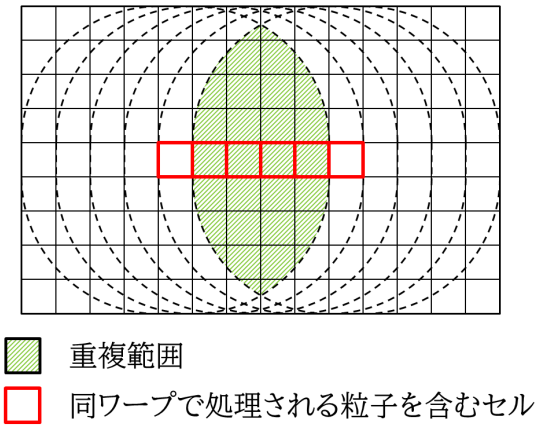

図 3.13: yz 平面と探索範囲の交差 (k=2,y=yini)

3.2.2　近傍リストの登録順序の最適化

提案する最適化では，x 軸方向に連続する 4 セルに含まれる粒子の登録範囲が重複するセルを優先して探索するよう実装した．つまり，4 セル内の粒子が共通してリストに登録する範囲に含まれる粒子が，同順に各近傍リストへ登録される．図 3.14,3.15 はそれぞれ，2 次元,3 次元で連続するセルに存在する粒子の探索範囲が重複した範囲内のセルを示す．最

適化により，共通範囲の粒子を同じタイミングでロードするため，キャッシュヒットが起きアクセスを高速化できる．

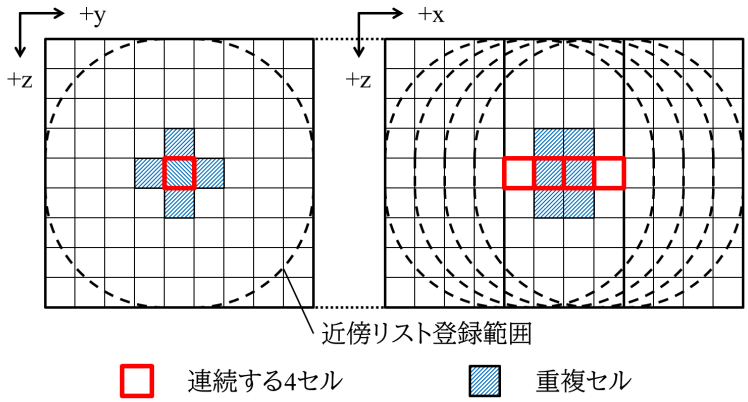

図 3.14: yz 平面と探索範囲の交差 (k=2,y=yini)

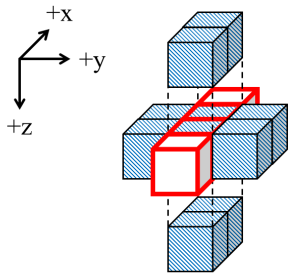

図 3.15: yz 平面と探索範囲の交差 (k=2,y=yini)

4　性能評価

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を比較した．

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．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．

4.2　実行時間の比較

4.3　粒子密度

5　考察

6　おわりに

謝辞

参考文献

[1] Gingold, Monaghan. Smoothed particle hydrodynamics: theory and application to no-spherical stars. *Mon.Not.R.astr.Soc.*, vol. 181, p. 375-389, Nov 1977

[2] 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

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

[4] R. B. Canelas, A. J. C. Crespo, J. M. 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

[5] A. J. C. Crespo, K. 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, 6(6), 2011

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

[7] K. Takada, T. Nitta, and K. Ohno. Acceleration of SPH-based fluid simulation on GPU (in Japanese). *High Performance Computing Symposium*, 2017:26-35, May 2017

[8] 著者, タイトル, 掲載誌, 巻, ページ, 発行年月

A　プログラムリスト

B　評価用データ