

論文 / 著書情報
Article / Book Information

題目(和文)	GPUスパコンにおける動的負荷分散を用いた大規模粒子法シミュレーション
Title(English)	
著者(和文)	都築怜理
Author(English)	Satori Tsuzuki
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第10097号, 授与年月日:2016年3月26日, 学位の種別:課程博士, 審査員:青木 尊之,末包 哲也,長崎 孝夫,肖 鋒,横田 理央
Citation(English)	Degree:., Conferring organization: Tokyo Institute of Technology, Report number:甲第10097号, Conferred date:2016/3/26, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

平成 27 年度 博士論文

GPU スパコンにおける動的負荷分散を用いた
大規模粒子法シミュレーション

東京工業大学 大学院総合理工学研究科
創造エネルギー専攻
都築 怜理

指導教員
青木尊之 教授
平成 28 年 3 月

目次

第 1 章	序論	1
1.1	研究背景	1
1.2	研究目的	4
1.3	本論文の構成	6
第 2 章	近接相互作用に基づいた粒子法シミュレーションの数値計算手法	7
2.1	粉体計算手法	7
2.1.1	個別要素法	7
2.1.2	時間刻み幅	9
2.1.3	クォータニオンによる回転角の管理	10
2.1.4	壁境界の設定	10
2.1.5	初期値の生成方法	11
2.1.6	符号付き距離関数法による任意形状の表現	12
2.1.7	粉体計算の検証	12
2.2	流体計算手法	19
2.2.1	改良型 SPH 法	19
2.2.2	境界処理および例外処理	24
2.2.3	流体構造連成計算	25
2.2.4	流体・流体構造連成計算の検証	26
第 3 章	粒子法シミュレーションの GPU コンピューティングによる実装	37
3.1	GPU コンピューティングの概要	37
3.1.1	GPU のアーキテクチャ	37
3.1.2	GPU を用いたスレッド並列計算	38
3.1.3	CUDA プログラミング	39
3.2	単一 GPU による粒子法計算	41
3.2.1	粒子のデータ格納形式	41

3.2.2	近傍粒子探索	42
3.2.3	Linked-list 法	43
3.2.4	セル番号のソートによるデータロードの効率化	43
3.2.5	流体構造連成の GPU 計算	47
3.2.6	単体 GPU での実行性能	49
3.3	複数 GPU を用いた粒子移動の計算	55
3.3.1	パッシブ・スカラー粒子の複数 GPU 計算	56
3.3.2	均一分布・一様速度場を用いた性能評価	59
3.3.3	再整列の頻度のモデル化の提案	63
第 4 章	複数 GPU を用いた粒子法シミュレーションの動的負荷分散法	69
4.1	スライスグリッド法による動的負荷分散	69
4.1.1	隣接小領域への転送粒子数の決定方法	70
4.1.2	GPU における粒子の数え上げの実装	71
4.1.3	不均一粒子分布のパッシブ・スカラー粒子計算による性能評価	73
4.1.4	粉体・流体の複数 GPU 計算への適用	79
4.1.5	DEM を用いた粉体計算の強・弱スケーリング	89
4.2	空間充填曲線を用いた動的負荷分散	91
4.2.1	木 (ツリー) を用いた格子細分化による領域分割法の利点	91
4.2.2	空間充填曲線を用いた動的負荷分散法の概要	93
4.2.3	空間充填曲線の数学的背景	94
4.2.4	空間充填曲線の特徴と辿り (たどり) 順序	96
4.2.5	領域再分割を用いた動的負荷分散アルゴリズム	99
4.2.6	複数 GPU 実装	102
4.3	GPU スパコンにおける各動的負荷分散法の性能評価	110
4.3.1	スライスグリッド法と空間充填曲線の比較	110
4.3.2	パッシブ・スカラー粒子計算による 3 次元領域分割法の検討	114
4.3.3	空間充填曲線を用いた 2 次元領域分割と 3 次元領域分割の比較	125
4.4	流体構造連成問題への適用	130
4.4.1	大規模サスペンション・フロー計算の実現と弱スケーリング	132
第 5 章	複数 GPU による大規模粒子法シミュレーションの実問題への適用	139
5.1	実行環境	139
5.2	ゴルフ・バンカーショットの解析	140
5.3	複雑形状を含むダム崩壊シミュレーション	148

5.4	建物と流木を含む津波シミュレーション	153
5.5	多数の瓦礫を含む土石流シミュレーション	160
第 6 章	結論	167
付録 A	ゴルフ・バンカーショットの解析 (別視点から可視化した結果)	173
	謝辞	175
	参考文献	177
	研究業績一覧	183

目次

2.1	個別要素法の相互作用モデル	8
2.2	仮想粒子の配置	11
2.3	初期値の生成	12
2.4	符号付き距離関数法による物体表現	13
2.5	粒子同士の衝突前後でのエネルギーの保存	14
2.6	複数回の衝突前後でのエネルギーの保存	15
2.7	粘性を 0 としない場合のエネルギーの変化	15
2.8	一定速度を与えた粉体の転がり試験	16
2.9	粉体のホッパー計算例	16
2.10	粉体のダムブレイク問題において法線方向のみを考慮して計算した結果	17
2.11	粉体のダムブレイク問題において接線方向を導入し摩擦を考慮して計算した結果	18
2.12	粒子法による流体構造連成計算の模式図	25
2.13	2次元の正方領域におけるキャビティ・フロー問題の概念図	27
2.14	2次元キャビティ・フロー計算 ($Re=1,000$)	28
2.15	実験配置図	29
2.16	Lobovský らの実験値との比較	30
2.17	初期の配置図	31
2.18	重心速度の変化	33
2.19	初期の圧力分布	34
2.20	浮力の検証計算における実験配置図	34
2.21	直方体を落下させた場合の重心位置の変化	35
3.1	Kepler 世代の GPU アーキテクチャ	38
3.2	ビルトイン変数と配列要素の対応関係	40
3.3	均一空間格子を用いた近傍粒子探索	42

3.4	セル分割法への Linked-list 法の適用	44
3.5	Linked-list に合わせて定期的に行われるセル番号をハッシュ値とした 粒子データのソート	45
3.6	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	46
3.7	実装 (A) を用いた場合の計算時間の内訳	48
3.8	実装 (B) を用いた場合の計算時間の内訳	48
3.9	演算密度を一定にする場合の DEM 計算の問題設定	49
3.10	ダムブレイク問題に対する DEM 計算の実行性能の比較	52
3.11	Roofline Model による実行性能の妥当性の検証	53
3.12	単体 GPU での実行性能 (棒グラフ上部の数値は GFlops 値)	54
3.13	粒子移動の GPU 間通信	55
3.14	時間積分の手順	56
3.15	GPU メモリの断片化	57
3.16	粒子番号の再整列	58
3.17	4,194,304 個の粒子を粒子構造体のメンバ (所属領域の番号) でソートす る場合の CPU と GPU の計算時間の比較	59
3.18	均一分布・一定速度場を用いた弱スケーリング	60
3.19	均一分布・一定速度場を用いた強スケーリング	61
3.20	4 GPU を用いて 100 回に 1 回ソートを行う場合 (赤) と、ソートを行わな い場合 (青) の 1 ステップの計算時間の変化	62
3.21	64 GPU を用いた場合のソート頻度の違いによる積算計算時間の変化の比較	63
3.22	図 3.21 のソートを行わない場合の測定値を、式 (3.21) によりフィッティ ングした結果	66
3.23	図 3.21 の測定値とモデル値の比較	67
3.24	再整列の頻度の違いによる計算時間の変化	68
4.1	2 次元スライスグリッド法による動的領域分割の概要	69
4.2	小領域の番号と粒子の通信量の関係	70
4.3	GPU 上での粒子の数え上げと領域境界線の移動	71
4.4	16,777,216 個の粒子に割り振られた 10 種類のセル ID の GPU 上での数 え上げにかかる計算時間の比較	72
4.5	渦度速度場中のスライスグリッド法による動的負荷分散のスナップ ショット (前半)	74

4.6	渦度速度場中のスライスグリッド法による動的負荷分散のスナップ ショット (後半)	75
4.7	4 GPU を用いた場合の小領域間の最大粒子数と最小粒子数の比率の変化	76
4.8	強スケーリングによる性能評価	77
4.9	DEM における複数 GPU の計算手順	80
4.10	改良型 SPH 法における複数 GPU の計算手順	82
4.11	CAD データ (左) と CAD データから生成した符号付距離関数	83
4.12	64 GPU を用いた粉体のダム崩壊シミュレーションにスライスグリッド 法を適用した様子	85
4.13	32 台の GPU を使い 416 万個の粒子を使った螺旋すべり台シミュレ ーション	86
4.14	64 台の GPU を使い 433 万個の粒子を使った搬送計算	87
4.15	64 台の GPU を使い 412 万個の粒子を使った攪拌計算	88
4.16	DEM による粉体の強・弱スケーリング	89
4.17	代表的な領域分割法の比較	91
4.18	木 (ツリー) を用いた格子細分化による動的負荷分散法の概要	94
4.19	ヒルベルト曲線, モートン曲線, ペアノ曲線の 3 種類の空間充填曲線に おける特徴的な辿り順序	97
4.20	2 次元ペアノ曲線を用いた場合の 9 分木に対する 4 通りの辿り方と, 子 リーフに対する次の順序の指定方法	98
4.21	ヒルベルト曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割さ れた小領域を 12 種類の色で塗り分けた様子 (下段)	105
4.22	モートン曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割され た小領域を 12 種類の色で塗り分けた様子 (下段)	106
4.23	ペアノ曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割された 小領域を 12 種類の色で塗り分けた様子 (下段)	107
4.24	複数 GPU 計算における粒子数・領域番号テーブルを用いた領域分割の手順	108
4.25	隣接小領域の探索の様子	108
4.26	直方体の計算領域に対して, 実際に木 (ツリー) が生成される領域 (粒子 数・領域番号テーブルの空間格子を生成する領域)	109
4.27	3 次元の各空間充填曲線における領域番号・粒子数テーブルに消費され るメモリ使用量の比較	110
4.28	弱スケーリングによるスライスグリッド法と 2 次元空間充填曲線の実行 性能の比較	111

4.29	図 4.28 の各測定点において GPU の台数を増加させて測定した強スケールリングの結果	113
4.30	動的負荷分散を 1 回実行するのに要する計算時間の内訳	114
4.31	時間方向における動的負荷分散の精度の N_{min} への依存性	118
4.32	空間方向における動的負荷分散の精度の N_{min} への依存性	119
4.33	A)~C) のそれぞれの条件で計算した最大接続数の変化	120
4.34	C) による計算時の 500 ステップ目における接続数のヒストグラム	121
4.35	各空間充填曲線におけるツリーの時間方向のメモリ使用量の変化について、計算条件 A)~C) で比較した結果	122
4.36	各空間充填曲線を用いた場合の 3 次元パッシブ・スカラー粒子計算における動的負荷分散の様子	123
4.37	ペアノ曲線を用いた場合のリーフの変化 (左), リーフの接続の様子 (中央), 及び水平方向の速度分布の変化 (右)	124
4.38	2 億 1,100 万個の粒子と 512 GPU を用いたダムブレイク計算における、空間充填曲線の違いによる計算時間の比較	125
4.39	3 次元ヒルベルト曲線, モートン曲線, ペアノ曲線による 3 次元領域分割を改良型 SPH 法の 3 次元ダムブレイク計算に適用した結果	129
4.40	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	131
4.41	流体構造連成の複数 GPU 計算の手順	132
4.42	大規模サスペンションフロー計算の初期配置	133
4.43	TSUBAME 2.5 における弱スケールリングによる性能評価	134
4.44	図 4.43 における Reduction の内訳	135
4.45	2,304 個の物体を含む 8,743 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 256 GPU を用いて計算した結果	136
4.46	2,304 個の物体を含む 8,743 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 256 GPU を用いて計算した結果 (時刻 6.8 sec における拡大図)	137
5.1	TSUBAME2.5 のシステム構成	139
5.2	計算配置とサンドウェッジの軌道	141
5.3	垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化	142
5.4	ゴルフボールの軌道	142
5.5	64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算	143

5.6	表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを 0.1 秒後に停止させて計算した場合の垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化	144
5.7	表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを 0.1 秒後に停止させて計算した場合のゴルフボールの軌道	144
5.8	64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (表 5.1 と同一の計算条件のもと、サンドウェッジを 0.1 秒後に停止させた場合)	145
5.9	図 5.8 において、初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置して計算した場合の、垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化	146
5.10	図 5.8 において、初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置して計算した場合のゴルフボールの軌道	146
5.11	64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (図 5.8 において、初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置した場合)	147
5.12	自由の女神を含むダムブレイク計算の配置図	148
5.13	自由の女神の CAD データ (左) と CAD データから生成した粒子群データ (右)	149
5.14	1,000 万粒子を用いた自由の女神を含むダムブレイク計算における圧力分布の変化	150
5.15	図 5.14 の計算における動的負荷分散の様子	151
5.16	7,250 万粒子による大規模ダムブレイク計算	152
5.17	建物を含んだ津波シミュレーションの配置図 (側面から見た図)	153
5.18	4 種類の流木の CAD データ (上) と、生成された粒子群データ (下)	153
5.19	建物を含んだ津波シミュレーションの配置図 (上側から見た図、黄色は圧力センサーを設置した側面)	155
5.20	2,062 万粒子、256 GPU を用いて実行した瓦礫と建物の構造物を含んだ津波シミュレーション	157
5.21	建物 A~G の壁面における圧力値の変化	158
5.22	建物 B の壁面における圧力分布 (白線囲み部分は図 5.21 の測定の際のセンサー設置部分)	159
5.23	土石流シミュレーションの配置図	160
5.24	12 種類の瓦礫の CAD データ (上) と、生成された粒子群データ (下)	161

5.25	256 台の GPU を用いて計算した 10,368 個の瓦礫と相互作用する 1 億 1,756 万粒子による大規模土石流シミュレーション	164
5.26	10,368 個の瓦礫の運動エネルギーの総和の時間変化	165
5.27	大規模土石流シミュレーションにおける計算時間の内訳	165
5.28	動的負荷分散を 1 回実行するのに要する計算時間の内訳	166
5.29	壁粒子以外の流体粒子と物体構成粒子を所属領域ごとに 17 種類の色で塗 り分けて示した結果	166
A.1	64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (表 5.1 と同一の計算条件のもと, サンドウェッジのスイング速度を 2 倍 にした結果)	174

表目次

2.1	2次元キャビティ・フロー問題の計算及び物理条件	28
2.2	ダムブレイク問題の計算及び物理条件	29
2.3	球の沈降問題の計算及び物理条件	32
2.4	浮力のつり合い問題の計算及び物理条件	32
3.1	各計算項目における浮動小数点演算回数と，メモリアクセス量	51
3.2	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	54
4.1	図 4.8 の 512 GPU 使用時の計算時間の内訳	78
4.2	図 4.16 の 512 GPU 使用時の計算時間の内訳	90
4.3	それぞれの空間充填曲線の分割方式と使用する木 (ツリー), 子リーフの 辿り方の種類	96
4.4	弱スケーリングにおける小領域間の最大接続数の変化	112
4.5	512 GPU, 2 億 1,100 万粒子の改良型 SPH 法のダムブレイク問題に対し て, 3次元空間充填曲線による3次元領域分割を適用した場合の最大デー タ通信量, 通信バンド幅, 最大接続数	127
4.6	512 GPU, 2 億 1,100 万粒子の改良型 SPH 法のダムブレイク問題に対し て, 2次元空間充填曲線による2次元領域分割を適用した場合の最大デー タ通信量, 通信バンド幅, 最大接続数	127
5.1	バンカーショットの計算条件	140
5.2	計算条件	148
5.3	図 5.18 における各物体の個数とそれぞれの構成粒子数	154
5.4	計算条件	154
5.5	各建物のサイズと位置	155
5.6	計算条件	160
5.7	図 5.24 における各瓦礫の個数, 及びそれぞれの構成粒子数	162

第 1 章

序論

1.1 研究背景

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、粉体シミュレーションを行う重要性は広く認識されている。粉体粒子のサイズはミリからマイクロメートルであるが、我々の関心のある現象のサイズは数 10 センチから時には数メートル以上になる。そのため、数億個以上の粒子が興味対象の領域内に存在する。粉体挙動を実際よりも少ない粒子でモデル化する「粗視化」は現在では粉体シミュレーションにおける有効な手段の一つである [1]。しかし、粗視化の適用限界も数倍から高々 10 倍程度であるため [2]、計算機資源を節約する粗視化モデルでは表現できない現象も多く、近年、モデル化によるアプローチの限界が強く指摘されている。一方、最近のスパコンの発展は目覚ましく、その総合演算性能は 10 年で確実に 1000 倍に向上している。これは従来とは質的に異なる計算が可能になることを意味しており、現実に近い粒子数で大規模計算を行うことにより計算精度は飛躍的に向上し、粉体シミュレーションの信頼性が大幅に向上する。さらには、高解像度化が進むことでこれまでは実現できなかった第一原理的な計算へと粉体シミュレーションを近づけることができ、それによって粉体現象の詳細なダイナミクスや新しいメカニズムを明らかにできる可能性がある。

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、プラズマやイオンの代表点である場合もあり、また、連続体を空間離散化するための代表点である場合もある。

非圧縮性流れ問題に対する粒子法では、領域に分布する粒子と各粒子に対応する重み関数に基づいて非圧縮性 Navier-Stokes 方程式を離散化して流体現象を表現する [3][4]。流体に対して格子を用いる計算は移動する複雑形状の物体を扱うのが苦手であるのに対し、

粒子法は複雑形状や流体構造連成の計算を容易に扱うことができる利点を持っている。

流体と構造が共存する現象は非常に多く、その重要性については広く知られている。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。また、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。これらは多数の物体を含むことから「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の実現が期待されている。粒子法を用いることでこれらの問題を解決できる可能性は高いが、解析には多数の粒子が必要になる。複雑形状の物体を正確に表現したり数百～数キロメートル四方の領域に粒子を配置することを考えると容易に理解できる。また、粒子法による 3 次元流体シミュレーションの場合、計算精度は最大で粒子数の $2/3$ 乗に比例する [5][6][7][8][9][10]。つまり、8 倍の粒子数を使っても計算精度は 4 倍しか向上しないため、高精度なシミュレーションを行うためには大規模計算が必要になる。

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 [11][12][13][14][15]、分子動力学計算 [16]、個別要素法 (DEM: Discrete Element Method)[17] による粉体計算、SPH(Smoothed Particle Hydrodynamics) 法 [18][19][20] による流体計算、メッシュフリー法 [21][22][23][24] 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。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 [11][25] 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は Graphics Processing Unit (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 [26][27] 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上手に使う必要がある。

スパコンは多数のノードから構成されており、ノードにメモリが分散され、ノード間は高速なネットワークで接続されている。それぞれのノードが割り当てられた粒子を計算することで大規模粒子計算が実現できる。これらの大規模計算においては計算領域を空間方向に分割し、分割された各領域にノードを割り当てて計算するのが効率的である [28]。重力多体計算や分子動力学計算は遠距離相互作用に基づいているため計算はメモリ・ア

アクセスよりも四則演算が支配的であり、GPU スパコンのような Byte/Flops が低いスパコンでも容易に高い実行性能を得ることができる。一方、個別要素法 (DEM) による粉体シミュレーションや SPH (Smoothed Particle Hydrodynamics) 法による流体計算などは近接相互作用に基づいた粒子計算であり、計算時間の大半がメモリ・アクセスであるため、Byte/Flops の低いスパコンではそもそも十分な性能が出ない。さらに、計算コストは各領域内の粒子数に依存するため、多数のノードから構成されたスパコン環境では、粒子分布が時間・空間的に激しく変化することでノード毎の粒子数が不均一になり、粒子が密集した領域の計算負荷は著しく上昇することになり、大規模計算は実質的に続行できなくなる。そのため、DEM や SPH 法などの近接相互作用に基づく粒子法のシミュレーションにおいて十分な実行性能を達成するには、計算領域を粒子分布に合わせて再分割して、各ノードの計算負荷を均一にする動的負荷分散の導入が必須である。しかし、これらの粒子法シミュレーションの分野では動的負荷分散などの大規模計算手法の研究への取り組みは殆どなく、分散メモリ環境では大規模粒子シミュレーションが行われた先攻研究例は非常に少ない [29][30][31]。参考文献 [29][30][31] などの研究でも複数の動的負荷分散法を比較することは行っておらず、スパコン環境において近接相互作用の粒子法シミュレーションに適した動的負荷分散法を議論する研究は非常に希である。

階層的なメモリ構造を持つ GPU スパコンでは、近接相互作用の粒子法シミュレーションにおける動的負荷分散の実現はさらに難しい問題である。計算コードの開発では既に述べたように CUDA プログラミング [26][27] や MPI ライブラリ [32] など大規模並列計算特有の専門技術を併せて用いるため実装の難易度は一層高くなる。さらに、粒子分布が激しく変化しノードを跨いでランダム方向に移動する場合にもそれらの粒子データを GPU 上で効率的に管理し、ノードを跨ぐ GPU 間での粒子データの通信コストを抑制しつつ、ノード内でも GPU-CPU 間の通信コストを最小限に抑える必要がある。個々の GPU では 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 GPU の持つ階層的メモリ構造を考慮した効率的な計算方法を検討する必要がある。GPU スパコンにおいて近接相互作用の大規模粒子シミュレーションを効率的に実行する計算手法は提案されておらず、GPU が潜在的に持つ高い演算性能を引き出し様々な問題を解決するために実現が強く求められている。一方、関連する重力多体計算や分子動力学計算、プラズマ粒子シミュレーションなどの分野では、空間分割された領域に対して領域境界線を移動させるスライスグリッド法 [33][34]、ParMETIS[35] や、Zoltan[36][37] 等のライブラリに代表される k-平均法 [38][39] 等のグラフ理論 [40] に基づく領域分割法、木 (ツリー) 構造による再帰的な領域分割法である二分法による直交再帰分割を行う KD-Tree[41] や Orthogonal Recursive Bisection (ORB)[42]、空間充填曲線を利用した木 (ツリー) を用いた格子細分化による領域分割法 [43][44] などの領域分割法が用いられている。これらを参考にしながら GPU スパコンなどの Byte/Flops の低いスパコンにおける近接相互作用の粒

子シミュレーションに最適な動的負荷分散について詳細に検討する必要がある。

流体構造連成問題の粒子法シミュレーションでは、これらの粒子法自体の大規模化の難しさに加えて連成計算特有の障壁がある。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。これらが原因となり GPU スパコンに限らず大規模計算の研究例はほぼ皆無である。

1.2 研究目的

複数 GPU を用いた近接相互作用に基づく粒子法シミュレーションを GPU スパコンにおいて高効率に実行するための計算手法を提案し、粉体や流体、その連成問題に対し、GPU スパコンを用いて粒子法による詳細な解析が可能であることを明らかにする。

粒子法の複数 GPU 計算では、計算領域を空間分割し、分割された小領域に GPU を割り当て、各 GPU が自身の担当する小領域内の粒子を計算する。しかし、計算領域を均等に分割して固定した場合、粒子分布が時間・空間的に変化するため計算が進行するにつれて各 GPU が処理する粒子数に偏りが生じる。個々の GPU のメモリ容量を超過する個数の粒子が特定の GPU に流入した場合にメモリ不足を引き起こす。数 GB メモリ容量に制限された GPU で構成される GPU スパコンでは特に深刻な問題であり、メモリが枯渇することで計算の続行自体が出来なくなる。また、粒子が密集した小領域を担当する GPU の計算負荷が上昇することにより並列化効率が著しく低下し、大規模計算は実質的に困難になる。粒子データの GPU 間での通信の繰り返しによって生じる GPU メモリの断片化により実行性能はさらに低下する。一方、個々の GPU で実行される粒子法計算ではランダム・メモリアクセスが発生するため、GPU の性能を引き出すために GPU のアーキテクチャに合わせた高速な計算手法を導入する必要がある。

本研究では近接相互作用に基づく粒子法シミュレーションを GPU スパコンにおいて実行する際に生じるこれらの問題を解決するために、階層的なメモリ構造をもつ GPU スパコンに適した効率的な動的負荷分散法を提案し、GPU スパコンにおける近接相互作用に基づく粒子法の大規模シミュレーションの高効率な計算を実現する。

本研究は GPU スパコンなど Byte/Flops の低いスパコンにおける近接相互作用に基づく粒子法シミュレーションに適した動的負荷分散法を明らかにし、それらの高効率実行を実現しようとする先駆的な研究である。具体的な研究内容を以下に述べる。

- 単一 GPU での先攻研究で提案されている近傍粒子探索法のためのセル分割法 [45] や Linked-list 法 [46] などの高速化手法を導入した後, GPU におけるグローバルメモリへのアクセス効率などを考慮しながら更なる高速化手法について検討する.
- 効率的な複数 GPU による粒子計算を実現するために, まずは GPU 間の粒子移動の問題に注力し, 複数 GPU 間で粒子の移動が発生しても効率的に処理できる計算手法を検討する. 特に頻繁な粒子のデータ通信の繰り返しにより GPU メモリが断片化進行する問題に対し, 粒子の再整列の利用による GPU メモリの断片化の解消を試みる. そのような処理を行うことによるオーバーヘッドと, メモリの断片化が解消されたことにより短縮される計算時間について, それらの関係性を表す数理モデルを提案して定量的な評価を行う.
- 粒子の空間分布が偏り各 GPU の粒子数が不均一になることで生じるメモリの枯渇や, 並列化効率が著しく悪化する問題の解決を図るため, GPU 間で動的負荷分散を導入する. 関連分野で用いられる動的負荷分散法はいくつかの種類があり, 現時点ではどの手法が最適であるかは分からない. ParMETIS[35] や Zoltan[36] の PT-Scotch[37] などのグラフ理論 [40] による領域分割は, 最小カット分割により各領域サイズを均等化して通信コストを抑えることが可能である. しかし, 領域分割 1 回あたりの分割コストが高いため, 静的な領域分割を初期において一度だけ行うような場合に適している. 動的負荷分散では頻繁な領域再分割を行うため実行コストをできるだけ抑える必要がある. そこで, 本研究では水平と垂直方向に領域境界線を移動させる単純なアルゴリズムにより領域再分割のコストを抑えることが可能な 2 次元のスライスグリッド法 [33][34] をまずは導入する. GPU スパコンにおいて強・弱スケーリングを実施し, 動的負荷分散を行わない場合と比較して大幅な実行性能の向上を試みる. スケーラビリティが悪化する部分ではその原因について検討する.
- スライスグリッド法を用いた場合の強・弱スケーリングの結果から, 粉体や流体の複数 GPU 計算における並列化効率の悪化の問題点を明らかにし, 空間充填曲線を用いた木 (ツリー) による格子細分化を用いた動的負荷分散などの導入を検討する. 新たに解決手法として提案した動的負荷分散により GPU スパコンにおける流体計算における強・弱スケーリングを実施して並列化効率の大幅な向上を試みる.
- 以上の研究から GPU スパコンにおける近接相互作用に基づく粒子法シミュレーションの高効率な実行を可能にする動的負荷分散法を明らかにし, 動的負荷分散が達成されたことではじめて取り組むことが可能になる実問題に対する粉体や流体の大規模シミュレーションを実現し, 有用性を実証する.

1.3 本論文の構成

本論文の構成は全部で 6 章から構成される。第 2 章「**近接相互作用に基づいた粒子法シミュレーションの数値計算手法**」では、本研究で用いる粉体や流体、流体構造連成のための近接相互作用に基づく粒子法の数値計算手法について説明する。第 3 章「**粒子法シミュレーションの GPU コンピューティングによる実装**」では、単一 GPU を用いた粒子法の高速計算手法について前半で説明し、粒子法の複数 GPU 計算における粒子移動の効率的な処理の方法を後半で提案する。第 4 章「**複数 GPU を用いた粒子法シミュレーションの動的負荷分散法**」では、本研究で提案する GPU 間の動的負荷分散法について説明する。すなわち、スライスグリッド法、及び 3 種類の空間充填曲線に基づく動的負荷分散法を DEM や SPH 法など近接相互作用に基づく粒子法の複数 GPU 計算で導入する方法を提案し、スライスグリッド法と空間充填曲線に基づく動的負荷分散の GPU スパコンにおける並列化効率を比較する。第 5 章「**複数 GPU による大規模粒子法シミュレーションの実問題への適用**」では、第 4 章までに提案した複数 GPU の計算手法を実問題に適用し、粉体や流体、及び流体構造連成の大規模シミュレーションを実現する。第 6 章「**結論**」で各章を総括した後、本論文の結論を述べる。

第 2 章

近接相互作用に基づいた粒子法シミュレーションの数値計算手法

2.1 粉体計算手法

2.1.1 個別要素法

個別要素法 (DEM: Discrete Element Method) は粉体挙動を表現する物理モデルとして広く用いられている。接触する粒子間で図 2.1 のように反発力とダンパー（バネとダッシュポット）が法線方向と接線方向に作用し、それらの粒子間力と重力など外力の合力のもと運動方程式に従い時間積分して粒子の位置や速度を更新する。粒子間の相互作用は粒子が接触している場合にのみ作用する。バネの力は粒子同士の食い込み深さに比例し、ダッシュポットは粒子間の相対速度に比例した減速力を粒子に与える。粒子に働く力は法線方向に働く反発力と、接線方向に働く摩擦力に分けることができ、接線方向のスライダーは、粒子の摩擦係数に応じて接線方向の力の上限を与える。

DEM において、粒子 i が粒子 j から受ける力のベクトル \mathbf{F}_{ij} は以下のように計算できる。

$$\mathbf{F}_{ij} = \mathbf{F}_{ij}^N + \mathbf{F}_{ij}^T = k^N \mathbf{L}_{ij} + c^N \frac{\Delta \mathbf{L}_{ij}}{\Delta t} + k^T \mathbf{S}_{ij} + c^T \frac{\Delta \mathbf{S}_{ij}}{\Delta t} \quad (2.1)$$

ここで、 \mathbf{F}_{ij}^N , \mathbf{F}_{ij}^T はそれぞれ法線方向と接線方向の力のベクトル、 k^N , k^T は弾性係数、 c^N , c^T は粘性係数である。また、 \mathbf{L}_{ij} , \mathbf{S}_{ij} は各方向のバネの圧縮量ベクトルを表し、 $\Delta \mathbf{L}_{ij}$, $\Delta \mathbf{S}_{ij}$ はその相対変位増分ベクトルである。さらに時間刻み幅を Δt としている。

接線方向の力による粒子に加わるモーメントのベクトル \mathbf{M}_{ij} は、粒子 i の半径を R_i ,

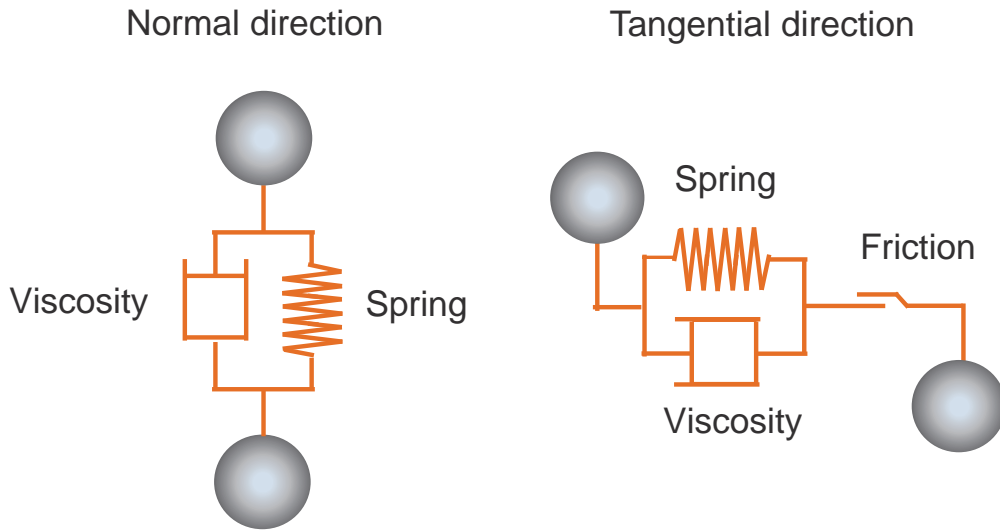


図 2.1 個別要素法の相互作用モデル

位置ベクトルを \mathbf{r}_i として次のように表される.

$$\mathbf{M}_{ij} = \frac{R_i}{|\mathbf{r}_j - \mathbf{r}_i|} \{(\mathbf{r}_j - \mathbf{r}_i) \times \mathbf{F}_{ij}^T\} \quad (2.2)$$

各粒子は一度に複数個の周囲の粒子と衝突する可能性があるため、粒子 i と接触している全ての粒子に関して力 \mathbf{F}_{ij} 及びモーメント \mathbf{M}_{ij} を計算し、粒子 i に作用する力とモーメントの合力ベクトルをそれぞれ求め、粒子 i についての並進運動と回転運動の運動方程式を解く.

$$m_i \frac{d^2 \mathbf{r}_i}{dt^2} = \sum_{i \neq j} \mathbf{F}_{ij} + m_{ij} \mathbf{g} \quad (2.3)$$

$$I_i \frac{d^2 \boldsymbol{\omega}_i}{dt^2} = \sum_{i \neq j} \mathbf{M}_{ij} \quad (2.4)$$

式 (2.3), 式 (2.4) を全ての粒子に対して計算し、それぞれ速度, 位置, 角速度及び角度を更新する.

法線方向の弾性係数 k^N は, Hertz の弾性接触理論 [47] に基づき, 粒子のヤング率 E 及びポアソン比 ν , 粒子半径 R , 各粒子の添え字を i, j とし, 以下のように表される.

$$k^N = \frac{4}{3\pi} \left\{ \frac{1}{\delta_i + \delta_j} \right\} \sqrt{\frac{R_i R_j}{R_i + R_j}} \quad (2.5)$$

$$\delta_i = \frac{1 - \nu_i^2}{E_i \pi} \quad (2.6)$$

$$\delta_j = \frac{1 - \nu_j^2}{E_j \pi} \quad (2.7)$$

また, 接線方向の弾性係数 k^T はラメの定義式より, 法線方向の弾性係数 k^N と次に関係にある.

$$k^T = \frac{k^N}{2(1 + \nu)} \quad (2.8)$$

さらに, 粘性係数 c^N, c^T は, 弾性バネとダッシュポットから成る 1 次元振動方程式の解析解において, 減衰が最大となる以下の関係を用いることが DEM の考案者である Cundall らによって提唱されている [17].

$$c^N = 2\sqrt{mk^N} \quad (2.9)$$

$$c^T = 2\sqrt{mk^T} \quad (2.10)$$

本研究における粉体計算では, 粒子のヤング率 E 及びポアソン比 ν を入力として与え, 上述の式 (2.6)~式 (2.7), 式 (2.8), 式 (2.9), 式 (2.10) から, 法線方向と接線方向の弾性係数 k^N, k^T , 粘性係数 c^N 及び c^T を自動的に決定している.

2.1.2 時間刻み幅

DEM の数値安定性の条件は, 一般に弾性バネとダッシュポットから成る 1 次元振動方程式の調和振動子の固有振動数を考えることにより, 以下の様に与えられる [48].

$$t_c = \frac{\pi}{\omega}, \quad \omega = \sqrt{(k/m_{ij}) - \eta_0^2} \quad (2.11)$$

ここで, m_{ij} は換算質量であり, $m_{ij} = m_i m_j / (m_i + m_j)$ である. また $\eta_0 = c / (2m_{ij})$ である. 弾性定数 k と粘性係数 c は通常は法線方向の k^N と c^N を用いる. また, 本論文では均一の球形粒子による DEM 計算を行うため, 常に $m_i = m_j$ である.

DEM の数値安定性条件の式 (2.11) では, t_c は法線方向の弾性定数 k^N と粘性係数 c^N から決定される. k^N と c^N はヤング率 E とポアソン比 ν から決定されるため, 粒子のヤング率 E とポアソン比 ν を入力として与えると, DEM の数値安定性条件 t_c も自動的に求まる. 時間刻み幅 Δt は t_c を何ステップで計算するかにより決まる. 時間刻み幅 Δt と t_c の間には以下の関係がある.

$$\Delta t \leq C t_c \quad (2.12)$$

定数 C は通常, $1/10 \sim 1/20$ 程度の値を用いる.

2.1.3 クォータニオンによる回転角の管理

3次元空間における回転表現の形式として, 基準座標系 x, y, z 軸周りの回転角 $\theta_x, \theta_y, \theta_z$ を用いて回転を表す Euler 角がある. Euler 角は直感的に理解しやすい利点があるが, 有効な自由度や次元が失われる数学的な特異点を含むために回転を正確に表現できない. この問題を回避するための常套手段のひとつに, クォータニオン [49] を利用して回転角を管理する方法がある [50]. クォータニオンは1つの実数部分と3つの虚数部分からなる4元数であり, クォータニオンを回転の表現に使うことにより Euler 角で発生する特異点の問題を回避し, 数学的に厳密に回転を管理することが可能になる [49][50][51].

ある粒子が規格化された回転軸 \mathbf{n} の周りに θ 回転した場合, クォータニオンの変化量を $d\mathbf{q}$ とすると,

$$d\mathbf{q} = \left[\cos \frac{\theta}{2}, \mathbf{n} \sin \frac{\theta}{2} \right] \quad (2.13)$$

と表すことができる. 現在のクォータニオンを \mathbf{q}^n とすると, 更新後のクォータニオン \mathbf{q}^{n+1} は,

$$\mathbf{q}^{n+1} = d\mathbf{q}\mathbf{q}^n \quad (2.14)$$

により得ることができる. ここで, \mathbf{n} は規格化された角速度ベクトルであり, θ は角速度の絶対値に時間刻みを掛けた値である.

$$\mathbf{n} = \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} \quad (2.15)$$

$$\theta = |\boldsymbol{\omega}| \Delta t \quad (2.16)$$

2.1.4 壁境界の設定

壁境界を設定する方法について, 粒子同士の場合と同様に粒子が壁から受ける力を計算し, 粒子に作用する合力の中に取り入れる. 壁の内側に仮想の粒子を設定することに

より、壁から受ける力についても粒子同士の場合と同じように計算できる。図 2.2 のように、粒子 i が壁と接触した際には壁内部に仮想粒子 j が存在するものとして 2 粒子間の粒子間力を計算し、それを粒子が壁から受ける力として評価する。仮想粒子 j の位置は壁の位置と粒子 i の位置から計算する。また、仮想粒子 i は回転していないものとして、角速度は $\omega_i = 0$ とする。並進速度については、仮想粒子 j の速度は固定壁の場合は 0 とし、壁が動く場合は仮想粒子 j の速度を壁の移動速度とする。

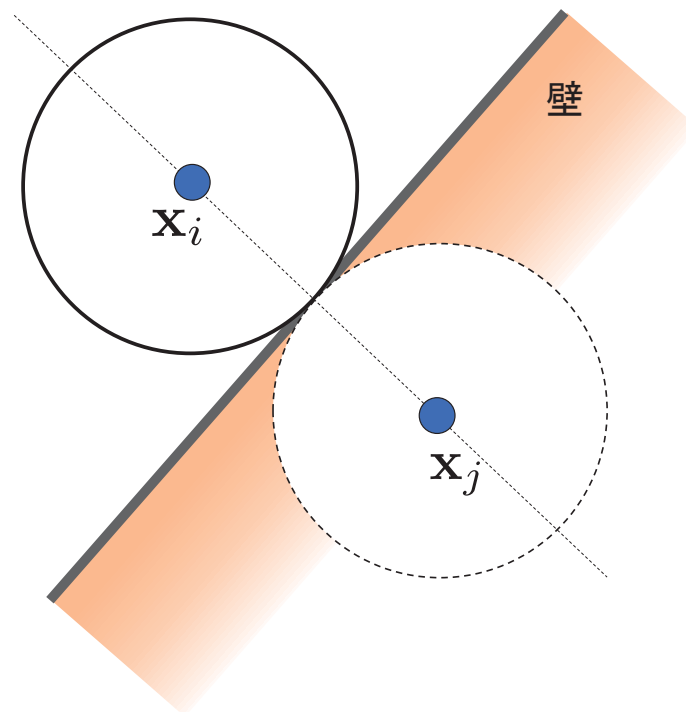


図 2.2 仮想粒子の配置

2.1.5 初期値の生成方法

DEM ではバネ及びダンパーに基づく物理モデルを用いるため、粒子を直交格子状に等間隔に隣接させて配置すると粒子同士がかみ合い、粒子間で連鎖的に力の伝搬が起こり数値発散の原因になる。粒子が積み重なった状態でも安定な初期値とするために粒子を上空から降らせて堆積させる方法 (堆積法) や、粒子の塊を落下させて安定状態に達するまで一定時間を経過させる方法 (重力落下法) がある [52]。後者は前者よりも初期値生成に要

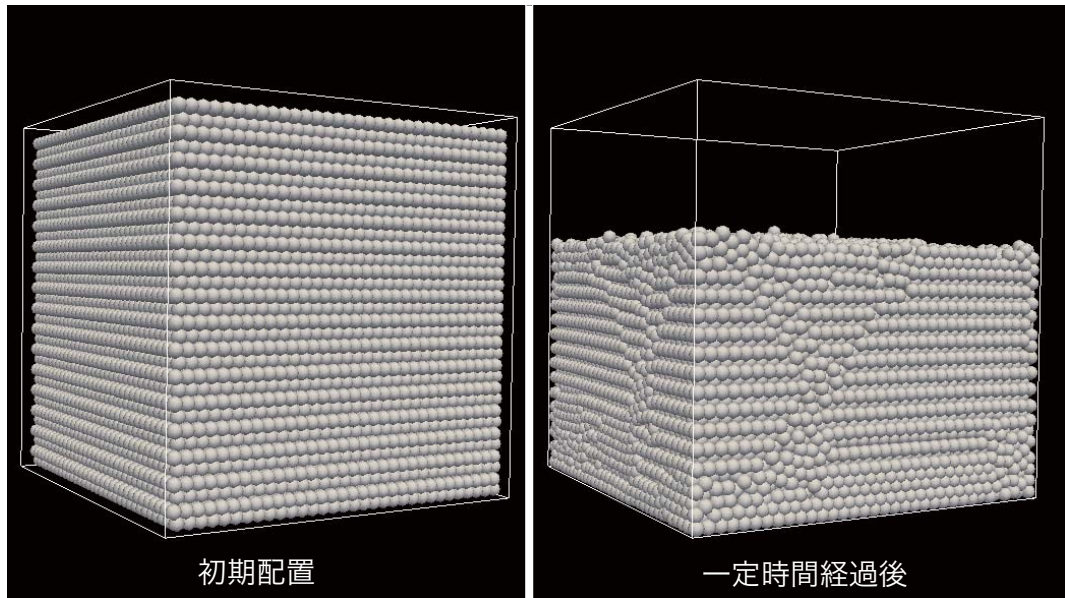


図 2.3 初期値の生成

する時間ステップが少なくすむという利点があり、本研究でも後者を用いる。この場合、図 2.3 の左側のように、初期に粒子を六方最密構造上に配置する。図 2.3 の右側は一定時間経過後の粒子分布であり、粒子の充填率の上昇や粉体特有の結晶化が確認できる。

2.1.6 符号付き距離関数法による任意形状の表現

実用的な DEM 計算では粉体と任意形状の物体との相互作用を取り扱う必要がある。物体は CAD で使用されるポリゴンによる形状データが利用できる。しかし、ポリゴンによる形状データとの粒子の衝突判定を直接計算しようとする、物体を構成するすべてのポリゴンの面や頂点との当たり判定を行う必要があり、計算コストが非常に高い。さらに、鋭角な形状をなすポリゴンとの相互作用計算は数値発散の原因にもなる。そこで、図 2.4 のように CAD データに対応する物体表面からの符号付距離関数 (Level Set 関数) を事前計算の段階で用意しておくことにより [53]、Level Set 関数を参照するだけで物体-粒子間の距離を判定することができ、粒子-物体間の相互作用を効率的に計算できる。

2.1.7 粉体計算の検証

粒子の物性値であるヤング率 E 及びポアソン比 ν を与えると上述のように DEM で必要な計算のパラメータは自動的に決定することができる。しかし、実際の物性値を用いると、時間刻み幅を非常に小さく (1×10^{-10} s 以下など) 設定する必要があるため、これら

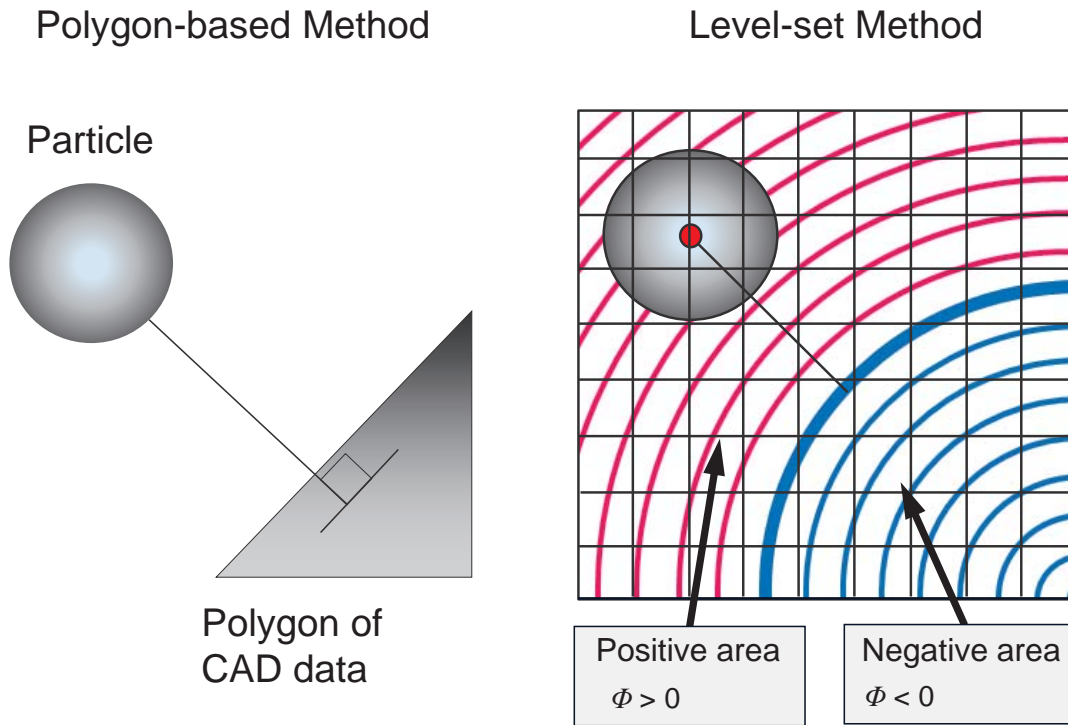


図 2.4 符号付き距離関数法による物体表現

を現実的な時間で問題を解くのは難しい。そこで、代替手法として粒子間にある程度のめり込みを許容し、実際のヤング率 E よりも $10^{-3} \sim 10^{-4}$ 程度小さな値を用いるのが一般的である。DEM における粒子間衝突が実際の粉体の衝突を再現しているかの目安として、エネルギー変化を確認する方法がある。粒子を自由落下させる場合、粒子の位置エネルギーと運動エネルギーが変化するが、落下の最中の位置エネルギーと運動エネルギーの和は一定に保たれる。ここで、誤差の指標として初期の粒子のエネルギーを E_0 、時間ステップ i における粒子のエネルギーを E_i として、以下で定義される $Error$ を導入する。

$$Error = \frac{|E_i - E_0|}{E_0} \quad (2.17)$$

DEM 計算では、反発力のモデルとしてバネを仮定しているため、粘性係数を 0 とした場合、衝突の前後でエネルギーが保存する。図 2.5 に粒子を壁面に 1 回衝突させた場合のエネルギーの変化を示す。位置エネルギーと運動エネルギーの和が衝突の前後で一致することがわかる。図 2.6 は粒子を複数回衝突させた例である。衝突の回数が増してもエネル

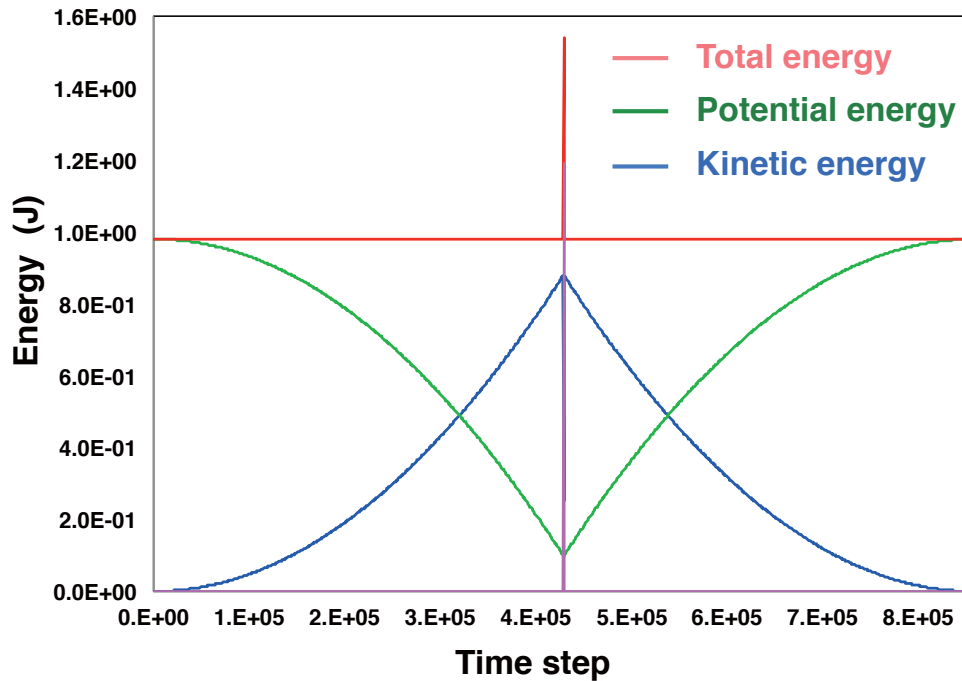


図 2.5 粒子同士の衝突前後でのエネルギーの保存

ギーが一定に保たれることが確認できる。図 2.6 で、粘性係数が 0 ではない場合のエネルギー変化を図 2.7 に示す。エネルギーの階段状の減衰が確認できる。

摩擦効果の検証

せん断方向の計算による摩擦の効果は図 2.8 に示すように粒子に一定方向に初速度を与えて転がしその挙動を調べることにより確認できる。粒子は回転しながら壁面を転がり摩擦の効果により静止するため、与えた初速の大きさと静止距離が対象とする粉体の物性と比較して同程度となるように変数を決定する。

実問題における摩擦効果の検証

実問題におけるせん断方向の計算の導入による摩擦の効果は、粉体のダムブレイク計算や図 2.9 に示すようなホッパー計算における粒子の揺らぎや、攪拌計算など、いくつかの粉体のベンチマーク問題によって確認することができる。図 2.10 と図 2.11 は 12,500 個の粉体粒子を用いた粉体のダムブレイク問題において、摩擦を考慮する場合としない場合の粉体挙動を比較した結果である。図 2.10 は摩擦を考慮せずに計算した結果であり、壁面を粒子が滑り慣性が支配的になっている。一方、粒子間の摩擦を考慮した図 2.11 は堆積した粉体の塊が崩れる様子が確認できる。

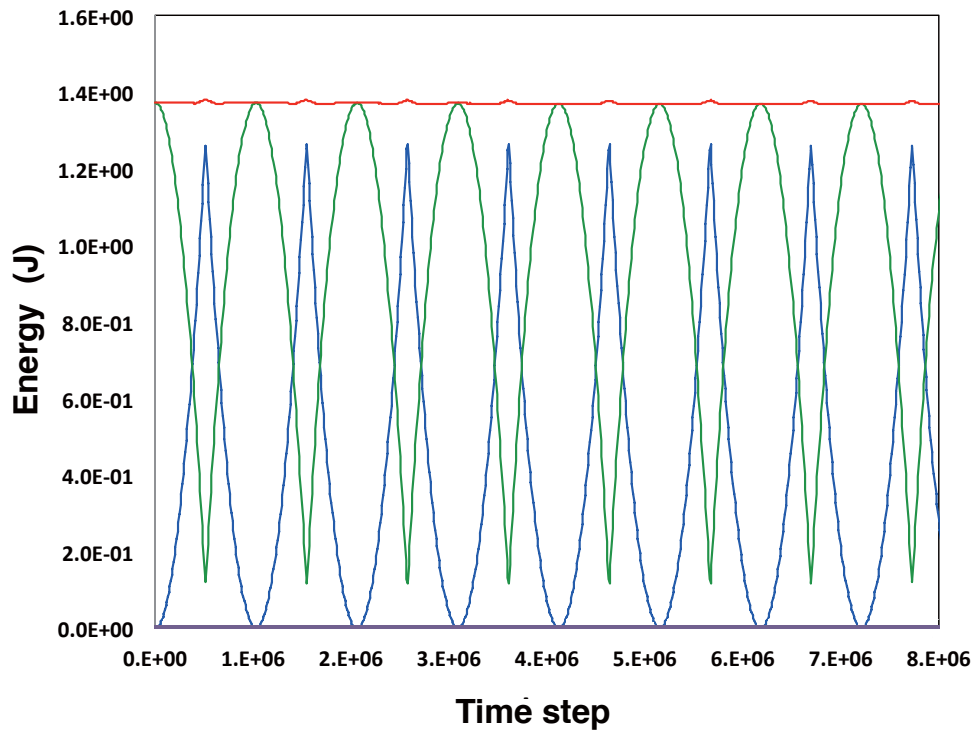


図 2.6 複数回の衝突前後でのエネルギーの保存

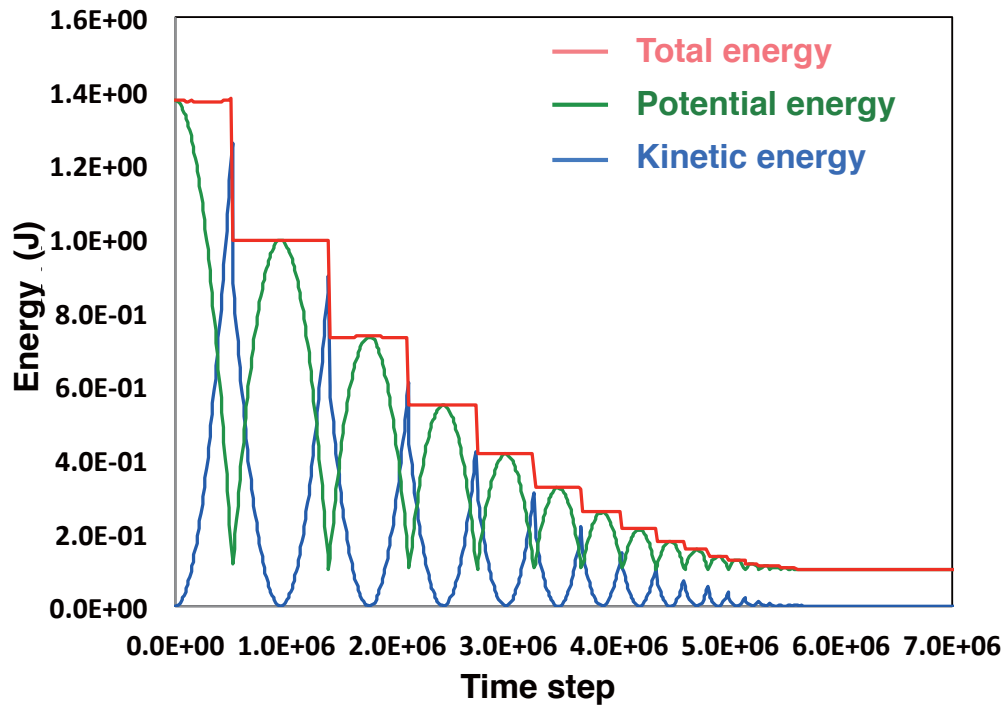


図 2.7 粘性を 0 としない場合のエネルギーの変化

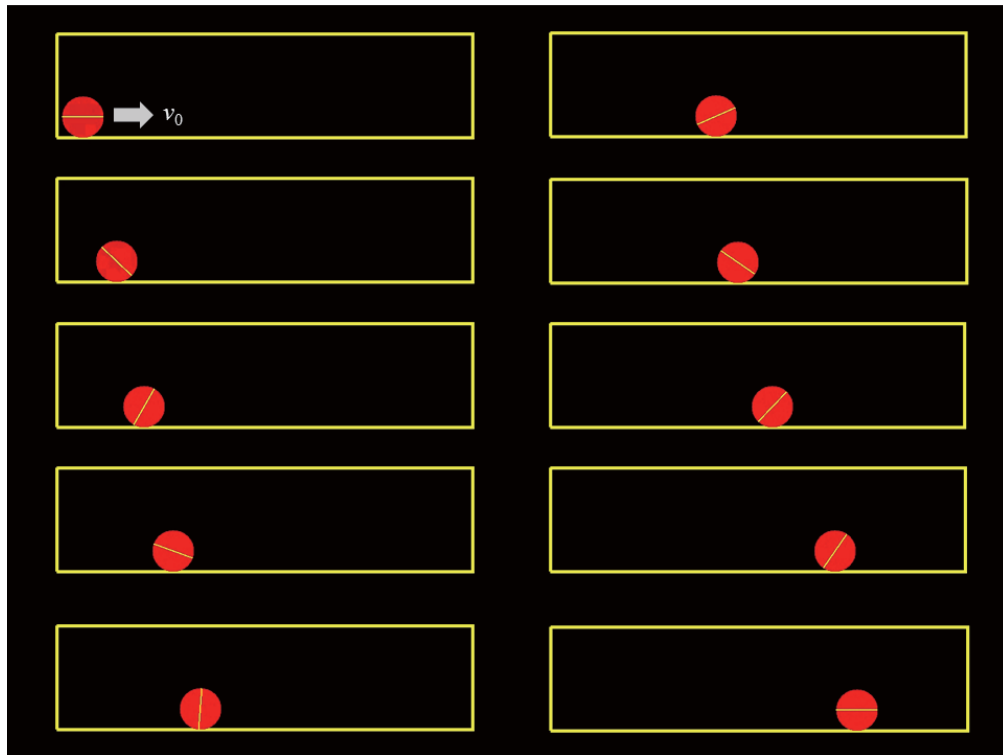


図 2.8 一定速度を与えた粉体の転がり試験

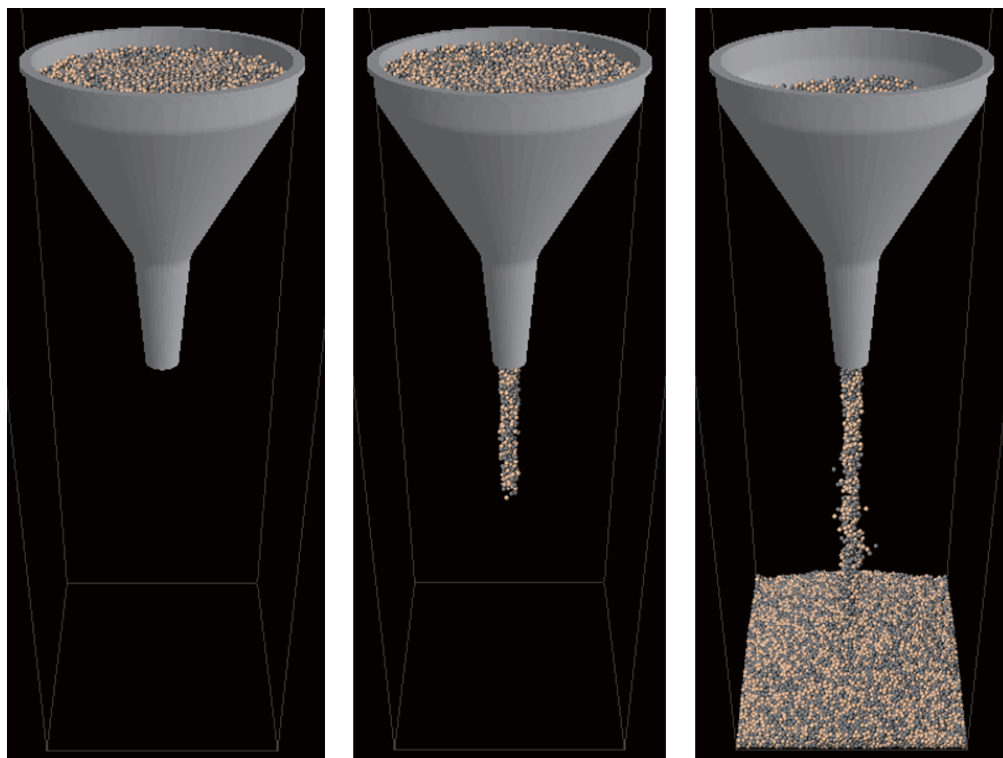


図 2.9 粉体のホッパー計算例

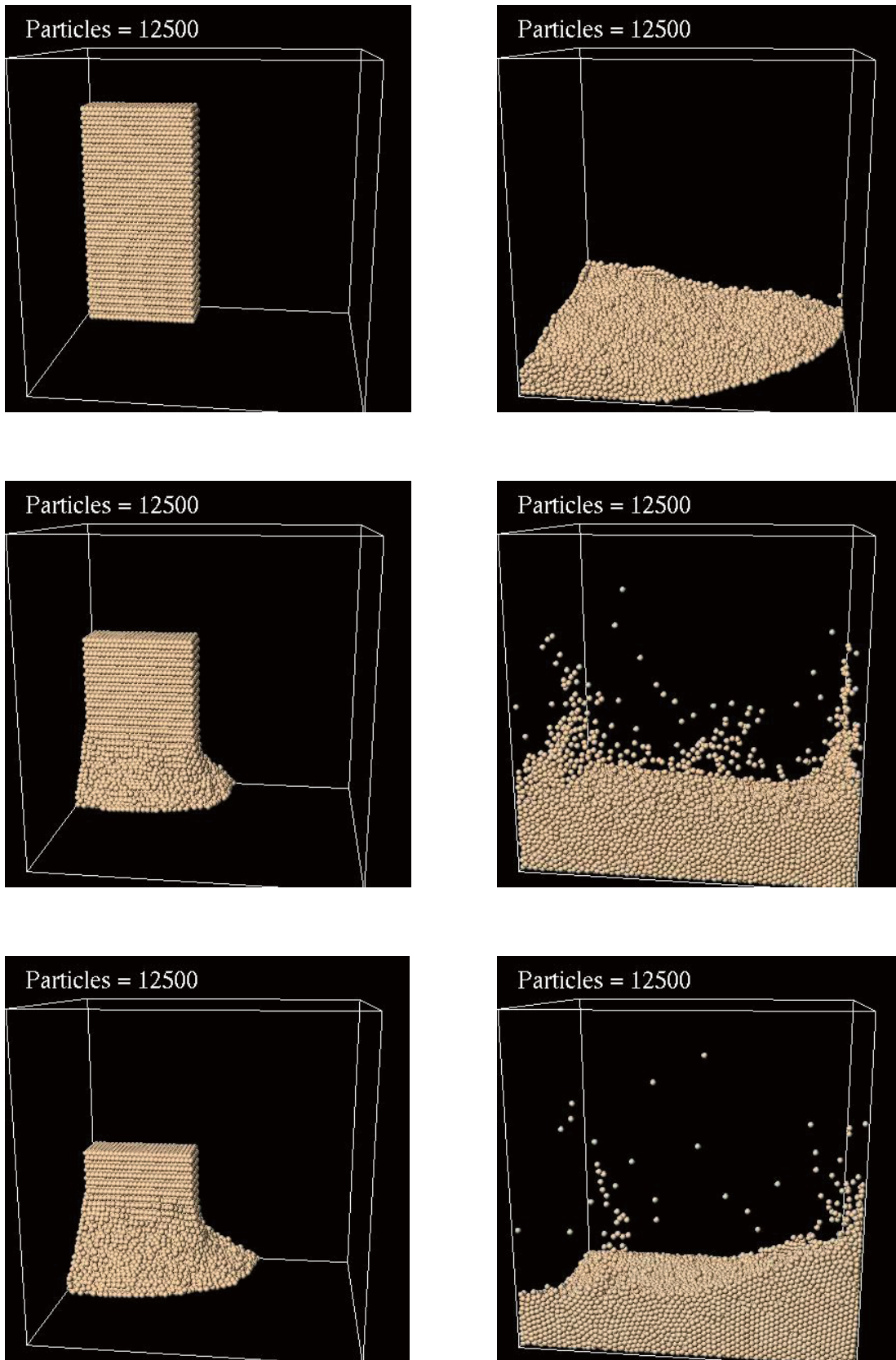


図 2.10 粉体のダムブレイク問題において法線方向のみを考慮して計算した結果

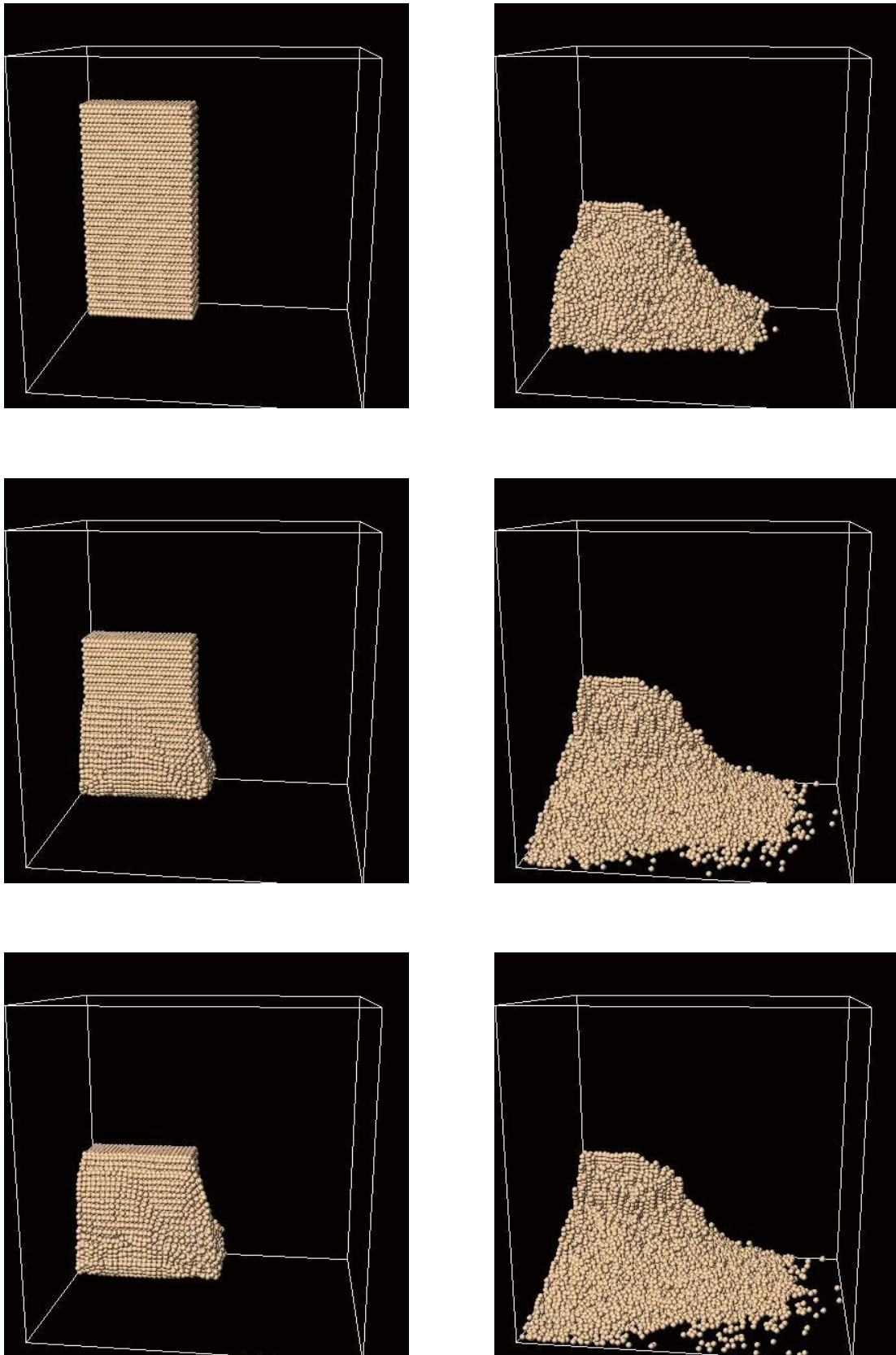


図 2.11 粉体のダムブレイク問題において接線方向を導入し摩擦を考慮して計算した結果

2.2 流体計算手法

2.2.1 改良型 SPH 法

本研究では、井元-田上 (2014) によって提案された流れ問題に対する粒子法を用いて流体計算を行う。本手法は陽解法 SPH 法 (Weakly Compressible Smoothed Particle Hydrodynamics)[20] や陽解法 MPS 法 (Explicit Moving Particle Simulation)[54] の空間近似と粒子の持つ物理量 (流速, 圧力) の計算方法を改良することによって、より連続的な流速場と圧力分布が得られる粒子法である。以後、改良型 SPH 法 (または改良型粒子法) と呼ぶ。

まず、改良型 SPH 法の近似微分作用素を導入する。 Ω を $d(\geq 1)$ 次元空間の領域とする。 Ω 内に N 個の粒子を配置し、その位置座標を x_i ($i = 1, 2, \dots, N$) とする。参照関数 $w : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ を、

$$w(r) \begin{cases} > 0, & 0 \leq r < 1, \\ = 0, & r \geq 1 \end{cases} \quad (2.18)$$

となる関数とする。 h を正のパラメータとし影響半径と呼ぶ。 w と h に対し、重み関数 $w_h : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ を、

$$w_h(r) := \frac{1}{h^d} w\left(\frac{r}{h}\right), \quad r \in \mathbb{R}_0^+ \quad (2.19)$$

と定める。このとき、関数 $\phi : \Omega \rightarrow \mathbb{R}$ に対する近似勾配作用素 ∇_h および近似 Laplace 作用素 Δ_h をそれぞれ、

$$\nabla_h \phi(x_i) := C_\nabla V_0 \sum_{j \neq i} \{\phi(x_i) - \phi(x_j)\} \frac{x_i - x_j}{|x_i - x_j|} w_h(|x_i - x_j|), \quad (2.20)$$

$$\Delta_h \phi(x_i) := C_\Delta V_0 \sum_{j \neq i} \{\phi(x_i) - \phi(x_j)\} w_h(|x_i - x_j|) \quad (2.21)$$

と定義する。ここに、 V_0 , C_∇ , C_Δ は、

$$V_0 := \frac{|\Omega|}{N}, \quad (2.22)$$

$$C_\nabla := \frac{d}{h \int_{\mathbb{R}^d} |x| w(|x|) dx}, \quad (2.23)$$

$$C_\Delta := -\frac{2d}{h^2 \int_{\mathbb{R}^d} |x|^2 w(|x|) dx} \quad (2.24)$$

である。本研究では、参照関数に spike 型の 2 次多項式

$$w(r) = \begin{cases} (1-r)^2, & 0 \leq r < 1, \\ 0, & r \geq 1 \end{cases} \quad (2.25)$$

を用いた。

次に、流れ問題の支配方程式である非圧縮性 Navier-Stokes 方程式と連続の式

$$\frac{Du}{Dt} = -\frac{1}{\rho} \nabla p + \nu \Delta u + g, \quad (x, t) \in \Omega \times (0, T), \quad (2.26)$$

$$\frac{D\rho}{Dt} = 0, \quad (x, t) \in \Omega \times (0, T) \quad (2.27)$$

の離散化を行う。ここに、 u , p , g , ρ , ν はそれぞれ、流速、圧力、重力、密度、動粘性係数である。 k ステップ目の時刻を t^k とし、 $\Delta t^k = t^{k+1} - t^k$ とする。時刻 t^k における i 番目の粒子の位置座標を x_i^k と書き、時刻 t^k 、位置 x_i^k の関数 v の値を v_i^k と書く。粒子はラグランジュ的に移動するものとし、物質微分を次のように近似する：

$$\frac{Dv}{Dt}(t^k) \approx \frac{v_i^{k+1} - v_i^k}{\Delta t^k}. \quad (2.28)$$

(2.27) を (2.28) で離散化することにより、密度 ρ_i^k を

$$\rho_i^{k+1} = \rho_i^k = \rho_0 \quad (2.29)$$

と離散化する。ここに ρ_0 は初期時刻の密度定数である。さらに、(2.28), (2.29) を用いて、(2.26) を次のように離散化する：

$$\frac{u_i^{k+1} - u_i^k}{\Delta t^k} = -\frac{1}{\rho_0} \nabla_h p_i^{k+1} + \nu \Delta_h u_i^k + g_i^k. \quad (2.30)$$

ここで、 t^k と t^{k+1} 間の仮想的な時刻 $t^{k+1/2}$ を導入し、(2.30) を

$$\frac{u_i^{k+1/2} - u_i^k}{\Delta t^k} = \nu \Delta_h u_i^k + g_i^k, \quad (2.31)$$

$$\frac{u_i^{k+1} - u_i^{k+1/2}}{\Delta t^k} = -\frac{1}{\rho_0} \nabla_h p_i^{k+1} \quad (2.32)$$

と分離する。時刻 $t^{k+1/2}$ における流速 $u_i^{k+1/2}$ は非圧縮性を満たさないので、時刻 $t^{k+1/2}$ における流体の微圧縮性を仮定する。改良型 SPH 法では圧縮性の液体に対する状態方程式の 1 つである Birch-Murnaghan の状態方程式 [55]

$$p = \frac{c_s^2 \rho}{\gamma} \left\{ \left(\frac{\rho + \Delta_t \rho}{\rho} \right)^\gamma - 1 \right\} \quad (2.33)$$

を離散化する. ここに, c_s は流体の音速, $\Delta t \rho$ は密度の時間変化, $\gamma (\geq 1)$ は流体に依存するパラメータである. ここで, 粒子数密度 n_i^k と粒子数密度の基準となる定数 n_0 を

$$n_i^k := \sum_{j \neq i} w_h(|x_i^k - x_j^k|), \quad (2.34)$$

$$n_0 := \sum_{m \in \mathbb{Z}^d \setminus \{0\}} w_h(l_0|m|) \quad (2.35)$$

と定める. ここに, l_0 は平均粒子間隔であり,

$$l_0 := \left(\frac{|\Omega|}{N} \right)^{1/d} \quad (2.36)$$

で与えられる. さらに, 時刻 $t^{k+1/2}$ における粒子の位置 $x_i^{k+1/2}$ を

$$x_i^{k+1/2} := x_i^k + \Delta t^k u_i^{k+1/2} \quad (2.37)$$

と定める. これらを用いて, (2.33) を

$$\hat{p}_i^{k+1} = C_w \frac{(C_s^k)^2 \rho_0}{\gamma} \left\{ \left(\frac{n_i^{k+1/2}}{n_0} \right)^\gamma - 1 \right\} \quad (2.38)$$

と離散化する. ここに, C_w は参照関数 w に依存するパラメータ, C_s^k は離散的な音速を示すパラメータである. また, (2.38) は粒子数密度によって離散化しているため, 流体の圧力 p_i^{k+1} とは区別し, \hat{p}_i^{k+1} を粒子圧力と呼ぶ. 粒子圧力 \hat{p}_i^{k+1} を用いて, 粒子を

$$x_i^{k+1} = x_i^{k+1/2} - \frac{(\Delta t^k)^2}{\rho_0} \hat{\nabla}_h^+ \hat{p}_i^{k+1} \quad (2.39)$$

で更新する. ここに, $\hat{\nabla}_h^+$ は

$$\hat{\nabla}_h^+ \hat{p}_i^{k+1} := -C_\nabla V_0 \sum_{j=1}^N (\hat{p}_i^{k+1} + \hat{p}_j^{k+1}) \frac{x_i^{k+1/2} - x_j^{k+1/2}}{|x_i^{k+1/2} - x_j^{k+1/2}|} w_h(|x_i^{k+1/2} - x_j^{k+1/2}|) \quad (2.40)$$

であり, 近似勾配作用素 (2.20) における相互作用 $\{\phi(x_i) - \phi(x_j)\}$ を, 粒子同士の衝突を避けるために $-\{\phi(x_i) + \phi(x_j)\}$ と置き換えたものである. 粒子圧力 \hat{p}_i^{k+1} を用いて, 圧力 p_i^{k+1} を

$$p_i^{k+1} = \Pi_h \hat{p}_i^{k+1} \quad (2.41)$$

で求める. ここに, Π_h は補間作用素で

$$\Pi_h \phi_i^{k+1} := \frac{\sum_{j=1}^N \phi_j^{k+1} w_h(|x_i^{k+1} - x_j^{k+1}|)}{\sum_{j=1}^N w_h(|x_i^{k+1} - x_j^{k+1}|)} \quad (2.42)$$

と定義する. この圧力 p_i^{k+1} を用いて, (2.32) によって流速を更新する. 本手法は (2.31) によって流速 $u_i^{k+1/2}$ を予測し, その流速 $u_i^{k+1/2}$ を用いて圧力を求め, (2.32) によって流速を修正する予測子修正子法となっているので, 流速 $u_i^{k+1/2}$ を予測子と呼び, (2.32) で予測子を修正する圧力勾配

$$-\frac{\Delta t^k}{\rho_0} \nabla_h p_i^{k+1} \quad (2.43)$$

を修正子と呼ぶ.

各種パラメータは以下のように与える. v_{\max}^k を

$$v_{\max}^k := \max_{i=1,2,\dots,N} |u_i^k| \quad (2.44)$$

とし, C_s^k は

$$C_s^k = \frac{v_{\max}^k}{C_M} \quad (2.45)$$

で与える. ここに, C_M は 1 以下の正定数でマッハ数と呼ばれる (マッハ数は経験的に 0.2 以下で安定した数値計算結果が得られる). あるいは, $v_{\max} := \max_k v_{\max}^k$ が予測できる場合は,

$$C_s = \frac{v_{\max}}{C_M} \quad (2.46)$$

と定数で与える. 時間刻み Δt^k は,

$$\Delta t^k = C_t \frac{l_0}{C_s^k} \quad (2.47)$$

と与える. ここに, C_t は 1 以下の正定数でクーラン数と呼ばれる. また, C_w は, 参照関数が (2.25) の場合に,

$$C_w = \gamma \quad (2.48)$$

と与えている. また, 影響半径 h は, $C_h := h/l_0$ としたときに, C_h が 2.1 以上となるように与える.

ここで, 式 (2.41) について説明を補足しておく. 粒子圧力 \hat{p}_i^{k+1} は, 粒子数密度による状態方程式を用いているため, 時刻 t^{k+1} における粒子数密度 n_i^{k+1} が $n_i^{k+1} \approx n_0$ となるように補正するポテンシャル量となる. したがって, この粒子圧力 \hat{p}_i^{k+1} は流体の圧力 p_i^{k+1} とは異なるものであり, この粒子圧力を実際の流体の圧力として解析している従来の陽解法の粒子法は非物理的な圧力振動や非常に小さい時間刻みが必要といった問題が発生する. しかしながら, 従来の粒子法は, このような圧力を用いているのにも関わらず,

ダムブレイク問題の先端速度や波高などが実験値とよく一致しており，流速および領域形状の妥当性が確認されている．そこで，粒子圧力 \hat{p}_i^{k+1} は流体の圧力 p_i^{k+1} に粒子数密度を補正するポテンシャル \check{p}_i^{k+1} が付加されていると考える：

$$\hat{p}_i^{k+1} = p_i^{k+1} + \check{p}_i^{k+1} \quad (2.49)$$

\check{p}_i^{k+1} は粒子数密度を補正するポテンシャルで均等な部分では 0 となるので，粒子数密度が疎な部分では正，密な部分では負となり，粒子数密度と比例関係にある．また，粒子数密度は均等な分布の場合 n_0 となり，空間平均が常に n_0 と近似できるので， \check{p}_i^{k+1} を空間平均をすると 0 と近似できる．したがって，補間作用素 Π_h による空間の重み付き平均を行うことで

$$\Pi_h \check{p}_i^{k+1} \approx 0 \quad (2.50)$$

と近似する．また，補間作用素 Π_h は作用する関数自身の近似

$$p_i^{k+1} \approx \Pi_h p_i^{k+1} \quad (2.51)$$

を表すので，(2.49)，(2.50)，(2.51) より

$$\begin{aligned} p_i^{k+1} &\approx \Pi_h p_i^{k+1} \\ &\approx \Pi_h p_i^{k+1} + \Pi_h \check{p}_i^{k+1} \\ &= \Pi_h \hat{p}_i^{k+1} \end{aligned} \quad (2.52)$$

と近似できる．粒子の平滑化は状態方程式によって決まるため，式 (2.38) で示した Birch-Murnaghan の状態方程式のパラメータ (γ など) や，他の状態方程式を用いた場合には弱圧縮を許容した範囲内で計算結果に影響する．本研究ではダムブレイク問題によって γ の選択による結果の違いを確認し，最も解析が安定であった $\gamma = 1$ または $\gamma = 2$ を採用している．

このような圧力値の補間を行うことが，改良型 SPH 法と従来の陽解法の粒子法との大きな違いである．また，空間離散化に用いる近似作用素や重み関数も従来のものとは異なっている．従来は格子点分布のような理想的な粒子分布における解析により近似作用素の重み関数を決定していたが [7]，改良型 SPH 法に用いる近似作用素は，打ち切り誤差評価の観点から一般の分布でも有効な重み関数を用いている [56]．以上の改良により，従来手法と比較して，改良型 SPH 法は空間の近似精度の向上し，安定した圧力分布が得られ，より大きな時間刻みをとることができるといった特徴を持つ．本手法の妥当性は第 2.2.4 節でいくらかの数値実験により示す．

2.2.2 境界処理および例外処理

本手法は粒子数密度を利用して物理量を計算するため、近傍粒子数が不足すると良い近似が得られない。そこで、壁境界では境界上に壁粒子、境界外に仮想粒子を分布させる。仮想粒子は、 λ を C_h の整数部分とすると、壁境界の外側法線方向に 2λ 層だけ分布させる。また、 λ 層内の仮想粒子に関しては、 $u_i^{k+1/2}$ を 0 とし、流体と同じように粒子数密度および粒子圧力および圧力を計算する。

粒子圧力は粒子の衝突を防ぐために

$$n_i^{k+1/2} < C_F n_0 \quad (2.53)$$

を満たす粒子 $x_i^{k+1/2}$ に対する粒子圧力を

$$\hat{p}_i^{k+1} = 0 \quad (2.54)$$

とする。ここに、 C_F は 1 以下の定数で、通常は負の粒子圧力が発生しないように $C_F = 1$ と与える。

また、自由表面を含む場合は、前述の粒子圧力の例外処理により、自由表面付近で粒子圧力が常に 0 となるので、自由表面付近で粒子分布が補正されずに非均一な分布となる。そこで、本研究では運動量を保存しながら粒子の位置を補正する衝突条件 [57] を用いて粒子分布の補正を行っている。また、通常の勾配作用素 (2.20) で圧力勾配を計算すると、自由表面付近では、粒子数の不足から自由表面外側に力が発生してしまうので、粒子圧力の勾配 (2.40) と同様に、 $(p_i^{k+1} - p_j^{k+1})$ を $-(p_i^{k+1} + p_j^{k+1})$ に置き換えた近似勾配作用素

$$\nabla_h^+ p_i^{k+1} := -C_{\nabla} V_0 \sum_{j=1}^N (p_i^{k+1} + p_j^{k+1}) \frac{x_i^{k+1} - x_j^{k+1}}{|x_i^{k+1} - x_j^{k+1}|} w_h(|x_i^{k+1} - x_j^{k+1}|) \quad (2.55)$$

によって計算を行う。

なお、粒子の少ない自由表面では粒子圧力に対して式 (2.53) と式 (2.54) に従い処理を行っているが、流体の流速・圧力に関しては特別な処理は行っていない。粒子圧力は粒子数の粗密を解消するようなポテンシャル量なので、粒子分布が疎な自由表面上も同じアルゴリズムを適用すると、自由表面の外側に粒子が飛び出る問題が起こるため、粒子密度が基準値より少ない粒子上の粒子圧力は 0 とし、この問題を防いでいる。この処理に対応するのが論文の自由表面判定の式 (2.54) である。一方、流速と圧力に関しては、低粘性の流れの場合、自由表面の境界条件は流速・圧力ともに 0 であると仮定できるので、自由表面付近で粒子数が不足していても問題が起きにくい (自由表面外側に仮想粒子を置いたとしても、その仮想粒子上の物理量を 0 とするため、参照する必要が無い、または参照したことによる影響が少ないからである)。

2.2.3 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する．計算量域内には図 2.12 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－流体粒子間の相互作用** 導入した改良型 SPH 法によって流体間の相互作用を計算する．
2. **流体粒子－物体構成粒子・壁粒子との相互作用** 流体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－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－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

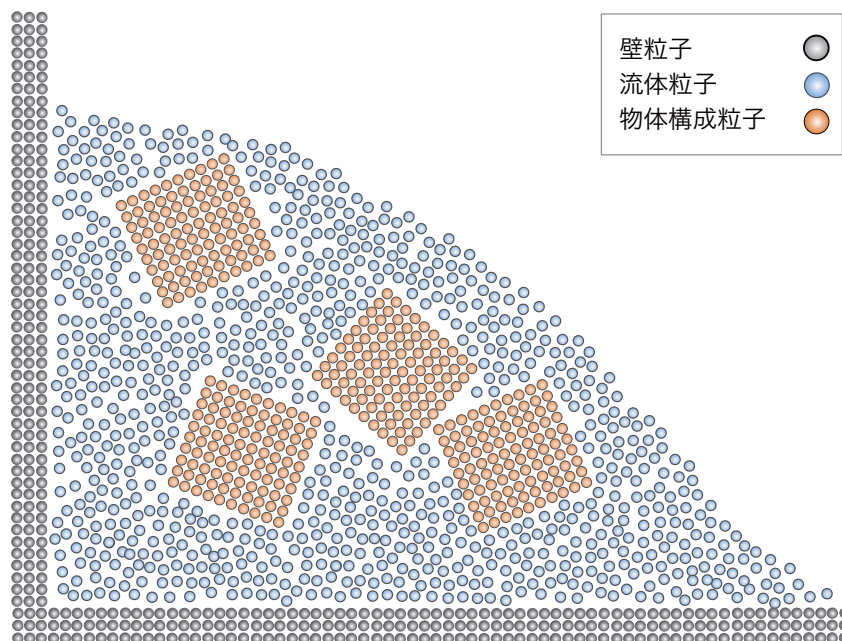


図 2.12 粒子法による流体構造連成計算の模式図

ベクトルに分解する.

1. と 2. の時間発展では, 予測子修正子法に基づく 2 次のオイラー法 (2.30) によって時間積分を行う. 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

$$M \frac{d\boldsymbol{v}}{dt} = \sum_{i \in \text{solid}} \boldsymbol{f}_i \quad (2.56)$$

$$\frac{d\boldsymbol{L}}{dt} = \sum_{i \in \text{solid}} \tilde{\boldsymbol{r}}_i \times \boldsymbol{f}_i \quad (2.57)$$

$$\boldsymbol{\omega} = I^{-1} \boldsymbol{L} \quad (2.58)$$

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ォ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.

2.2.4 流体・流体構造連成計算の検証

キャビティ・フロー問題

流体計算の基本的な検証問題にキャビティ・フロー問題がある. キャビティ・フロー問題は直方体領域の上面に, その境界に沿った一定の流速の境界条件を与え, その他の境界では速度 0 の境界条件を与える. これにより, 領域内部に渦が発生し, その渦の形状が領域形状, 境界の流速, 密度, 粘性率によって変化するような流れである (図 2.13 参照). 一定時間経過したキャビティ・フロー問題の解は定常・または時間周期的な解となるため, 全体の流速分布を検証することで, エネルギー収支が妥当であるかを確認することができる. また, レイノルズ数

$$Re := \frac{LU}{\nu} \quad (L; \text{代表長さ}, U; \text{代表速度}) \quad (2.59)$$

が高くなるにつれて, 領域の各角に微小な 2 次渦, 3 次渦が発生するので, 渦の再現性を確認することで, 高解像度な解析ができていのかどうかを確認することができる.

本研究では 2 次元の正方領域におけるキャビティ・フロー問題を考える. 領域 Ω を 2

次元の単位正方領域 $(0, 1) \times (0, 1)$ とし，次のように境界条件を与える：

$$u(x, 0) = v(x, 0) = u(0, y) = v(0, y) = u(1, y) = v(1, y) = 0 \quad (2.60)$$

$$u(x, 1) = 1, v(x, 1) = 0 \quad (2.61)$$

ただし，位置座標を (x, y) とし， x 方向と y 方向の流速をそれぞれ， u, v で記述している．なお，重力 g は考慮しない．

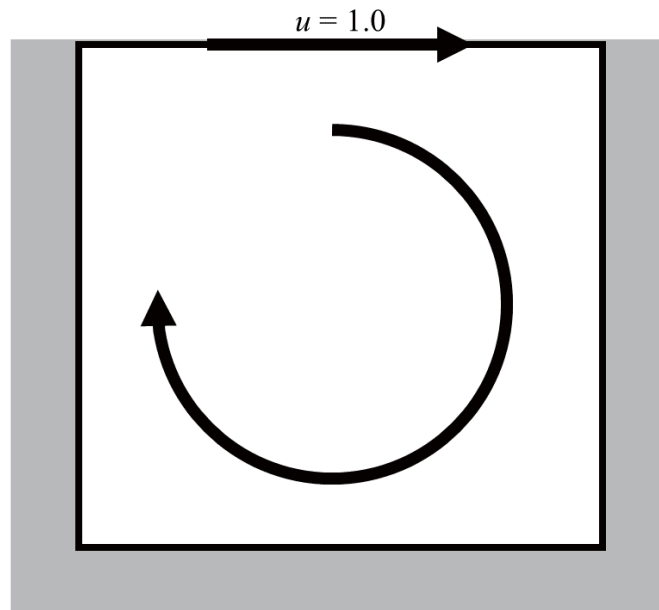


図 2.13 2次元の正方領域におけるキャビティ・フロー問題の概念図

改良型 SPH 法を 2次元キャビティ・フロー問題に適用し，Ghia らによる高次の差分法の数値計算結果 [58] と比較を行った．表 2.2 に改良型 SPH 法の計算条件を示し，図 2.14 に $Re = 1,000$ の定常状態における流速図（流れ関数の等高線図：左側）と速度分布のグラフ（右側）を示す．ただし，図 2.14 の流線図内のアルファベットおよび数字は [58] に対応している．また，図 2.14 の速度分布のグラフは，緑線が $x = 0.5$ における v の分布，赤線が $y = 0.5$ における u の分布，青点がそれらに対応する [58] の速度分布を示している．図 2.14 の流線図より，領域下部の両端に [58] の結果とよく一致した 2次渦が確認できる．さらに，図 2.14 の速度分布のグラフより，[58] の結果とよく一致した流速分布が得られている．以上より，2次元キャビティ・フロー問題による検証で，改良型 SPH 法は妥当なエネルギー収支と高解像度な計算が得られていることを確認できた．

ダムブレイク問題

自由表面を含む流れ問題の検証問題の 1 つとしてダムブレイク問題がある．ダムブレイク問題は，ダムのように水槽の一端で壁によって堰き止められている流体を用意し，堰き

表 2.1 2次元キャビティ・フロー問題の計算及び物理条件

ν	$[m^2/s]$	1.0×10^{-3}
ρ	$[kg/m^3]$	1.0
v_{max}	$[m/s]$	1
C_M		0.1
C_t		0.5
C_h		3.1
γ		1
<i>particles</i>		16,000
<i>physical time</i> [s]		30

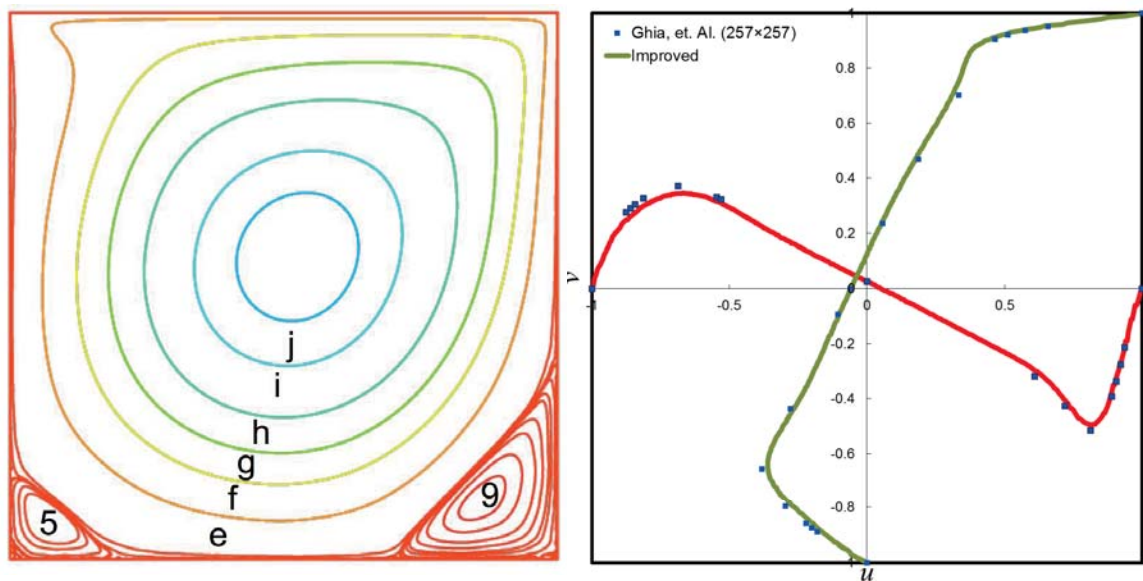


図 2.14 2次元キャビティ・フロー計算 (Re=1,000)

止めている壁を取り除くことで、流体が重力によって崩壊していく流れである。ダムブレイク問題は、流れの先端速度、自由表面の変化、対岸における圧力分布などのデータが物理実験により得られている [59] ので、実験値と数値実験を比較することによって数値実験で得られた領域形状や圧力分布の妥当性を確認することができる。

本研究では改良型 SPH 法による数値シミュレーションを Lobovsky らの物理実験

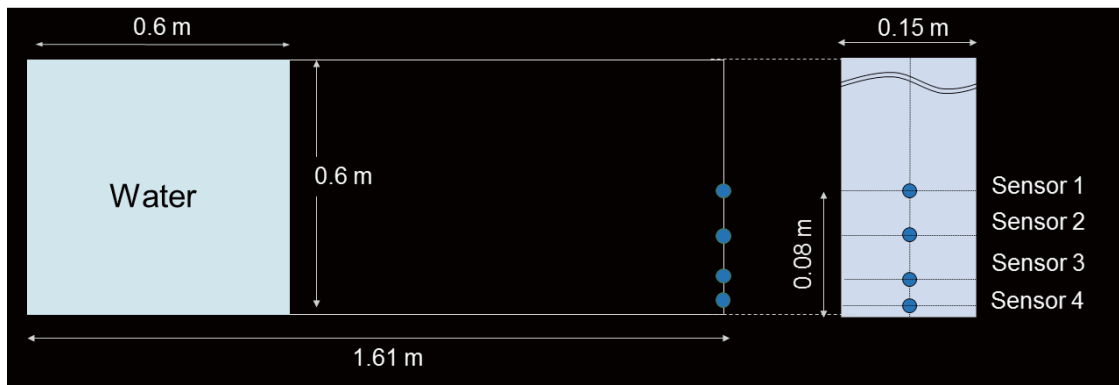


図 2.15 実験配置図

[59] と同一の物理条件で実施し，両者の圧力値を比較した．このときの実験配置図の概略図と計算条件を図 2.15 と表 2.2 にそれぞれ示す．図 2.15 の壁面に取り付けられた Sensor 1~Sensor 4 上における圧力値の時間変化を図 2.16 に示す．Lobovský らによる実験の測定値を緑で示しており，改良型 SPH 法による数値シミュレーション結果を赤線で示している．図 2.16 では横軸は時間スケールであり，Lobovský らの論文に合わせて $(H/g)^{0.5}$ で規格化されている．縦軸は圧力値であり $(\rho g H)$ で規格化されている．Sensor 1~Sensor 4 の場合も実験値と良い傾向で一致しており，特に Sensor 1~Sensor 3 においてはピークも実験値と近い値が得られていることが確認できる．

表 2.2 ダムブレイク問題の計算及び物理条件

g	[m/s^2]	9.8
ν	[m^2/s]	8.9×10^{-7}
ρ	[kg/m^3]	1.0×10^3
v_{max}	[m/s]	4
C_s	[m/s]	20
l_0	[m]	0.0025
C_t		1.0
C_h		2.6
γ		2
<i>particles</i>		5,866,267
<i>physical time</i> [s]		0.98

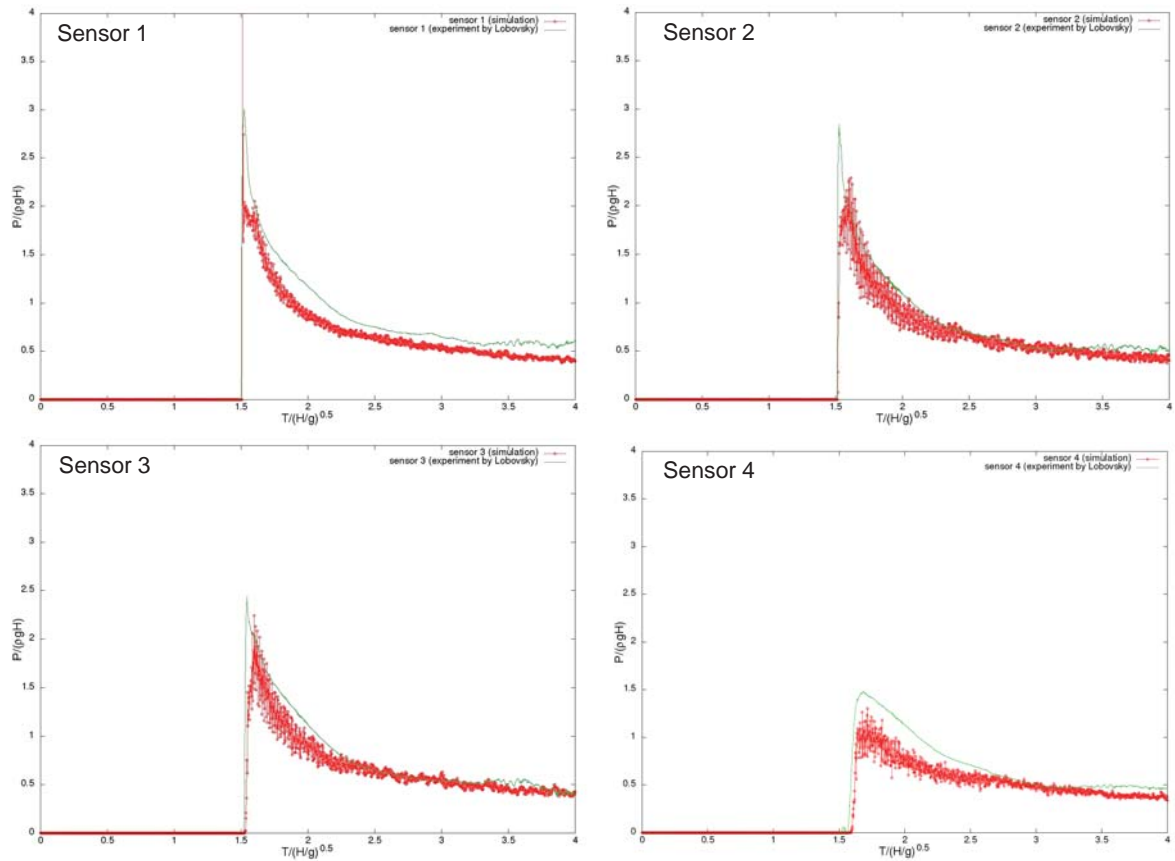


図 2.16 Lobovský らの実験値との比較

物体の沈降問題

流体中における球状物体の沈降問題の数値実験を行った．球状物体の沈降問題はその沈降する速度に対する理論解が存在するので，この数値実験により流体-物体間の相互作用の妥当性が確認できる．

球状物体の直径 0.75 m とし，数値実験では $1,736$ 個の粒子で充填して物体を構成し，図 2.17 のように初期に配置した．表 2.3 に示す計算条件 (ρ_f は流体の密度， ρ_{obj} は物体構成粒子の密度， D は球の直径を表す)のもと，初速度 0 で自由落下し，各時刻の球の重心位置における鉛直方向の速度 U を測定した．図 2.18 に測定結果を赤線で示す．図 2.18 では経過時間を横軸に，終端速度 U_t で規格化した速度 U/U_t を縦軸に表示している．重心位置における鉛直方向の速度が一定に収束するのが確認できる．また，図 2.19 に示す圧力分布からは安定した静水圧場が確認できる．

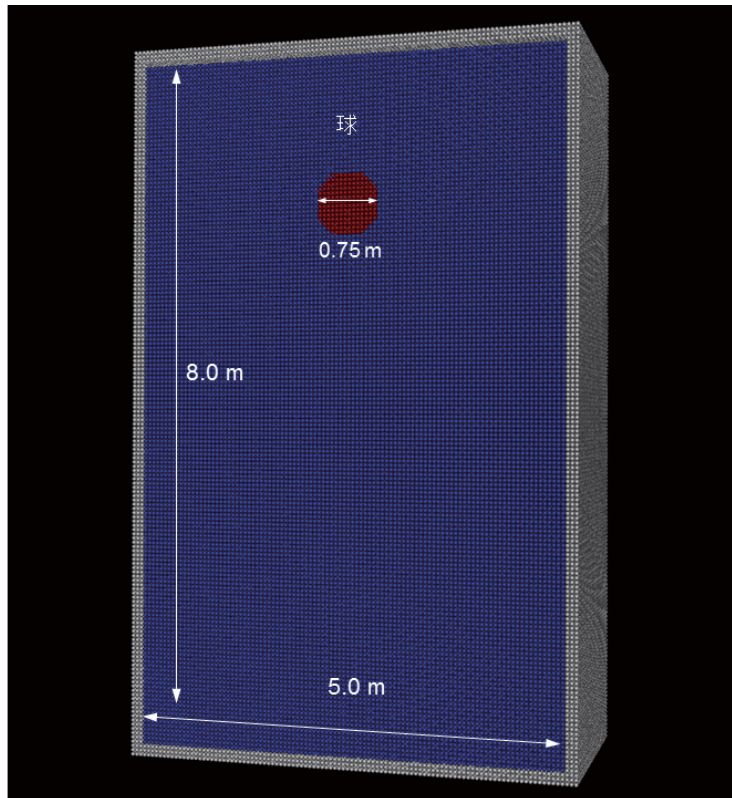


図 2.17 初期の配置図

理論解の鉛直方向の速度を v とすると，慣性抵抗を受けながら運動する物体の鉛直方向の運動方程式は，球の質量を m ，重力加速度を g として，

$$m \frac{dv}{dt} = mg - \kappa v^2 \quad (2.62)$$

となり，非線形微分方程式で表される．初期条件として $v(0) = 0$ とすることにより，終端速度を v_t ，時定数を τ として，式 (2.62) から，

$$v(t)/v_t = \tanh(t/\tau) \quad (2.63)$$

を得る．図 2.18 中の丸印の実線 (黒色) は式 (2.63) によるフィッティング曲線を示しており，数値実験とよく一致していることが確認できる．

浮力によるつり合い問題

物体が流体の密度より低い場合は，アルキメデスの原理より，物体にかかる重力と浮力に釣り合うことで物体は流体表面上で静止する．この浮力によるつり合いを確認するために浮力によるつり合い問題の数値実験を行った．

本研究では，図 2.20 に示すように直方体形状の物体を水面に落下させ，一定時間経過

表 2.3 球の沈降問題の計算及び物理条件

g	$[m/s^2]$	9.8
ν	$[m^2/s]$	1.0×10^{-6}
ρ_f	$[kg/m^3]$	1.0×10^3
ρ_{obj}	$[kg/m^3]$	2.5×10^3
v_{max}	$[m/s]$	15
D	$[m]$	0.75
C_s	$[m/s]$	200
l_0	$[m]$	0.05
C_t		1.0
C_h		2.6
γ		1
<i>particles</i>		1,866,412
<i>physical time</i> [s]		1.4

して静止状態に達するまでの物体の重心座標の変化を測定した。表 2.4 に浮力のつり合い問題の計算及び物理条件を示す。

表 2.4 浮力のつり合い問題の計算及び物理条件

g	$[m/s^2]$	9.8
ν	$[m^2/s]$	1.0×10^{-6}
ρ_{water}	$[kg/m^3]$	1.0×10^3
ρ_{obj}	$[kg/m^3]$	0.5×10^3
v_{max}	$[m/s]$	15
C_s	$[m/s]$	200
l_0	$[m]$	0.05
C_t		1.0
C_h		2.6
γ		2
<i>particles</i>		599,603
<i>physical time</i> [s]		12.4

この物理条件では、流体と物体の密度比が 2 倍なので、直方体の底面積を S 、高さを H 、静止状態における沈み込みの深さを L 、重力加速度を g 、水の密度を ρ_{water} 、直方体

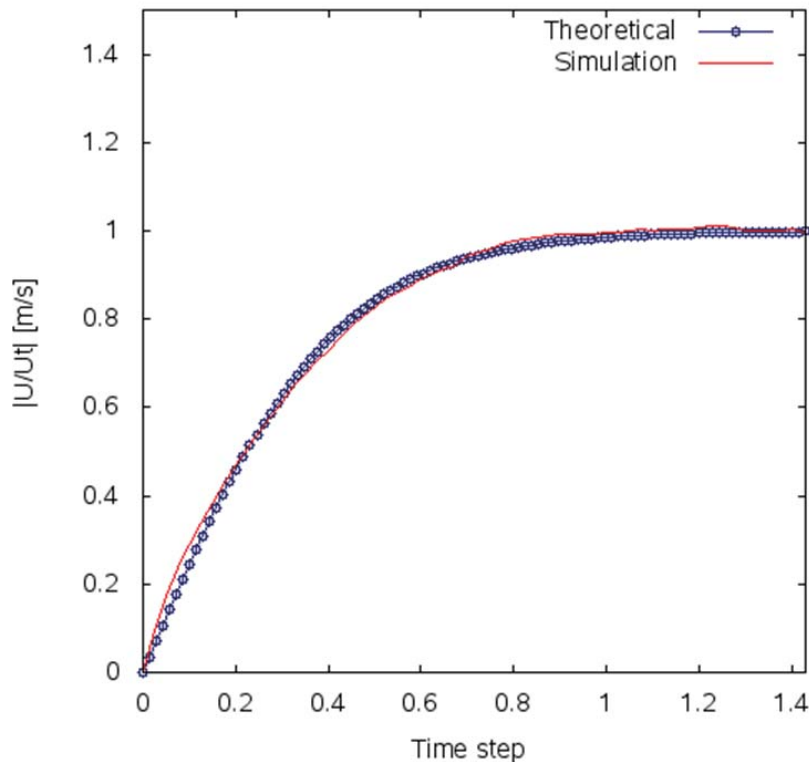


図 2.18 重心速度の変化

の密度を ρ_{obj} とした場合に，アルキメデスの原理より，

$$(SH\rho_{obj})g = (SL\rho_{water})g \quad (2.64)$$

であり， L について解けば，

$$L = \frac{\rho_{obj}}{\rho_{water}}H \quad (2.65)$$

となるので，静止状態において物体重心は水面の位置に一致する。

物理時間が 12.4 秒経過するまでの重心の変化を図 2.21 に示す．赤線は測定結果を表している．中心の実線はアルキメデスの原理から導出した静止状態における重心位置であり，35,000 ステップ以降の静止状態ではほぼ一致することが確認できる．落下直後から静止状態に至るまでの運動は，重力と浮力，及び速度に比例した粘性抵抗力による減衰振動で表すことができる．印付きの実線は減衰振動の特殊解として $f(x) = Ae^{Bx}\cos(Cx) + D$ でフィッティングした結果であり，測定値と非常に良く一致することが確認できる（フィッティングパラメータは $A=0.50834$, $B=0.00013$, $C=3.499915$, $D=4.166666$ ）。

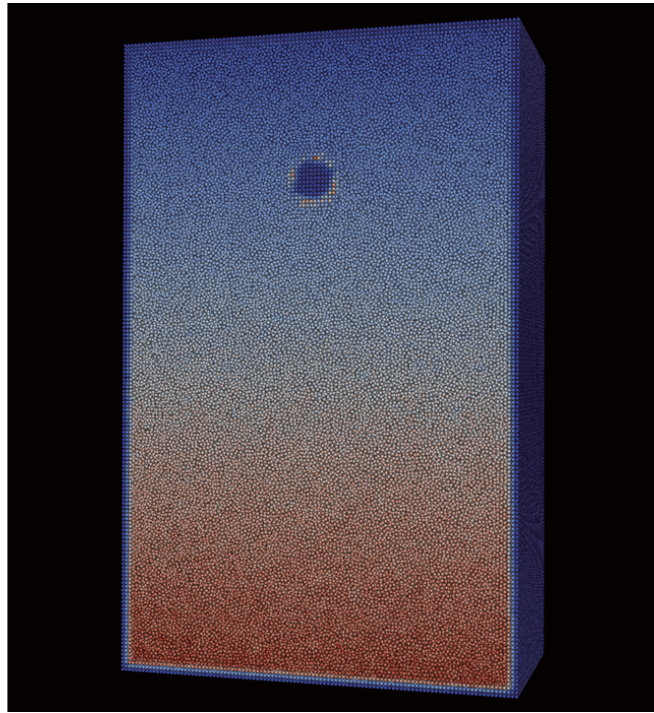


図 2.19 初期の圧力分布

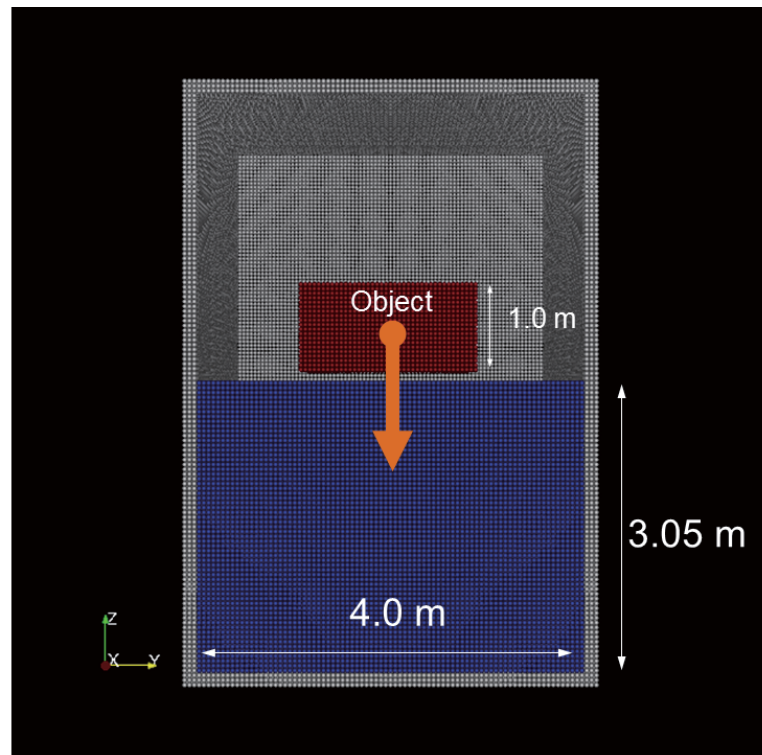


図 2.20 浮力の検証計算における実験配置図

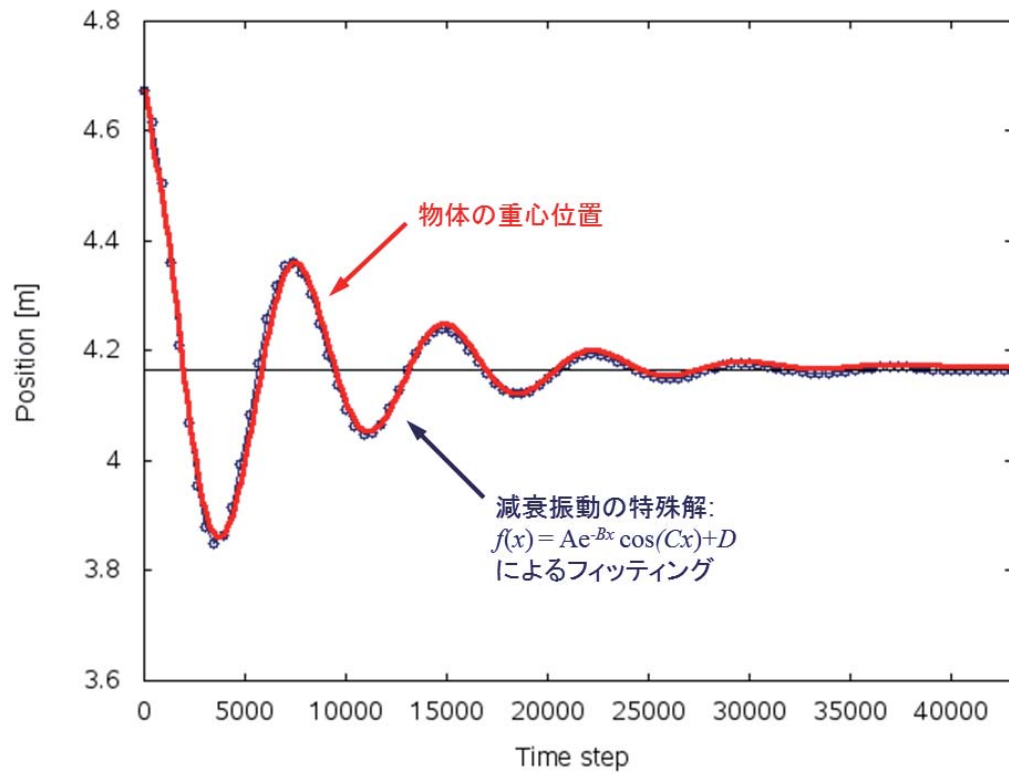


図 2.21 直方体を落下させた場合の重心位置の変化

第 3 章

粒子法シミュレーションの GPU コンピューティングによる実装

3.1 GPU コンピューティングの概要

3.1.1 GPU のアーキテクチャ

パソコンのグラフィクス・ボードには画像処理プロセッサである GPU (Graphics Processing Unit) が搭載されている。元々は高速な画像表示を目的として開発された GPU を汎用計算に使う GPU コンピューティング (General-Purpose computing on GPUs) への取り組みが 2000 年頃から始まった。NVIDIA 社が 2006 年に GPU コンピューティングの統合開発環境である CUDA (Compute Unified Device Architecture)[26][27] をリリースしたことにより通常の C/C++ 言語の拡張プログラミングが可能になり、GPU を汎用計算に利用する GPU コンピューティングが広く普及した。GPU のアーキテクチャについて、本研究で使用する NVIDIA 社の Tesla K20X (図 3.1 参照, 学術国際情報センター TSUBAME2.5 ハードウェア・ソフトウェア仕様 <http://www.gsic.titech.ac.jp/sites/default/files/spec25j1.pdf> より抜粋) を用いて説明する [60]。

Kepler コアの GPU では、192 個の演算ユニット (CUDA コア) が 1 つのストリーミング・マルチプロセッサ (SMX と呼ばれている) を構成している。Tesla K20X は 14 個の SMX を搭載しており、全部で 2,688 個の CUDA コアがある。1 つの SMX の中には 64 k 個の 32 bit レジスタファイルがある。また、非常に高速な L1 キャッシュと共有メモリ (合計で 64 kB), \sin や \cos などの数学関数を高速に計算する SFU (Super Function Unit) などがある。1 つの SMX 内の共有メモリは 192 個の CUDA コアからのみアクセスを許しており、他の SMX 内の CUDA コアからはアクセスできない。K20X を搭載したグラフィクス・ボード上には 6 GB のグローバル・メモリがあり、GPU からグローバル・メモ

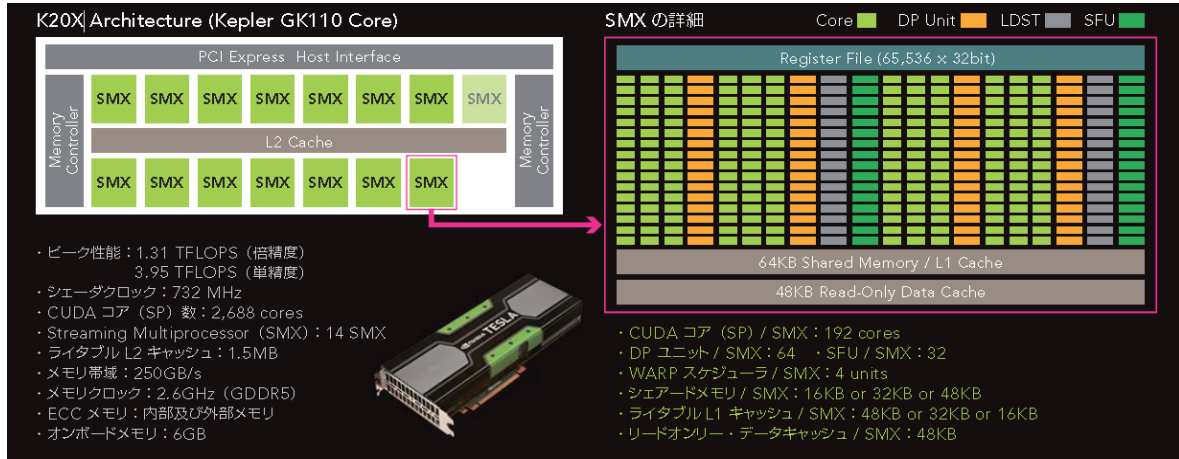


図 3.1 Kepler 世代の GPU アーキテクチャ

リへは 250 GB/sec というメインメモリに比べて数倍以上高速なメモリアクセスが可能である。グローバル・メモリには 2,688 個の CUDA コアから共有でメモリアクセスが可能である。通常は CPU からグローバル・メモリには直接アクセスすることができず、メインメモリとグローバル・メモリ間で PCI Express バス (Gen2 の帯域は 8 GB/sec, Gen3 では 16 GB/sec) を介してデータ通信を行うことができる。

3.1.2 GPU を用いたスレッド並列計算

GPU には数千個 (NVIDIA Tesla K20X の場合は 2,688 個) の演算コアが搭載されており、それらを並列計算によって効率的に利用することにより GPU の持つ高い演算性能を引き出すことができる。各 CUDA コアでは「スレッド」という単位で計算を実行する。32 個の CUDA コアは同一の演算命令を実行する必要があり、SIMD (Single Instruction Multiple Data) の計算となる。従って、32 個の CUDA コアが一斉にメモリアクセスを行い (アクセスするデータの数値は違ってよい)、レジスタにそれらのデータを保存し、同一の演算を処理する。NVIDIA Tesla K20X ではひとつの SMX に対して 2,048 個のスレッドを同時に実行命令することができ、投入されたスレッドをスケジューラが 32 個単位に分割し、メモリアクセスと演算処理を同時に実行させるなど効率的に処理が行われる。GPU 全体では 2,688 個の物理的なコア数を大幅に上回る $1024 \times 1024 \times 64 \times 1024$ 個のスレッドが投入可能である。各スレッドの内容は条件分岐などを含んで良く、一つの GPU に対して同一のプログラムが動く SPMD (Single Program Multiple Data) となっている。

3.1.3 CUDA プログラミング

GPU を利用するには CUDA を用いるのが主流になっている [26][27]. CUDA は通常の C 言語 または Fortran 言語に GPU コンピューティングのための言語拡張や API (Application Programming Interface) やコンパイラ等を含めた統合開発環境である. CUDA プログラミングでは, 計算コードの全体は 1) CPU での実行を記述するホストコード, 2) GPU の実行を記述するデバイスコードから構成される. ホストコードには通常の C/C++ や Fortran による記述に加えて, GPU のビデオメモリ上への配列確保やビデオメモリ間のデータ転送などのメモリ操作のための CUDA の API, GPU の実行内容を記述したカーネル関数のコール部分が記述される. 一方, デバイスコードには GPU カーネル関数の中身が記載され, その内容がスレッドとして各 CUDA コアで実行される.

CPU のホストメモリ上の配列 A, B, C に対して配列 B と配列 C の N 個の要素を加算した結果を配列 A にコピーする C/C++ 言語のプログラムは, 以下の様に見える.

プログラム 3.1 CPU の逐次コード

```

1 for (int i = 0; i < n; i++)
2 {
3     A[i] = B[i]+C[i];
4 }

```

GPU のデバイスメモリに確保された配列 A, B, C に対して, 配列 B と配列 C の N 個の要素を加算した結果を配列 A にコピーする GPU のカーネル関数に対するプログラムは, 以下の様になる.

プログラム 3.2 GPU のカーネル関数コード

```

1 __global__ void add_array (double* A, const double* B, const double* C)
2 {
3     int i = blockDim.x*blockIdx.x + threadIdx.x;
4     A[i] = B[i]+C[i];
5 }

```

CPU で動作する逐次プログラム 3.1 と同様に CUDA によるプログラム 3.2 においてもポインタや配列を利用できる.

GPU ではプログラム 3.2 の内容がスレッドとして 1 つの CUDA コアで実行される. ホストコードでは, カーネル関数のコール (実行指令) は以下のように記述される.

$$\text{add_array} \lll N/256, 256 \ggg (\text{A}, \text{B}, \text{C}); \quad (3.1)$$

” $\lll a, b \ggg$ ” の中の 2 つの引数 a, b が, GPU に投入されるスレッドの数を階層的に表しており, 第 2 引数の b はひとつのブロック内のスレッド数を, 第 1 引数の a はブロック数を指定し, 全部で $a \times b$ 個のスレッドが GPU で実行される (ブロック内のス

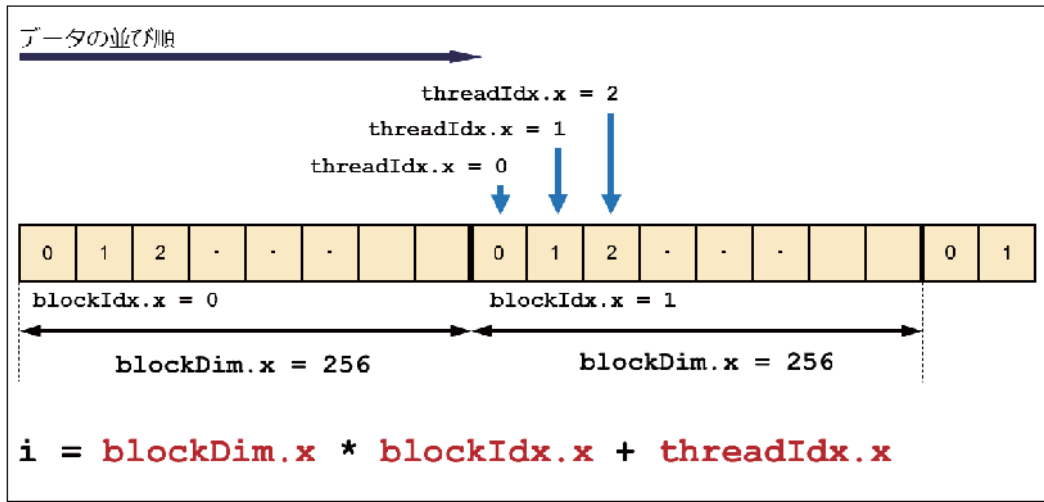


図 3.2 ビルトイン変数と配列要素の対応関係

レッドは同一の SMX 内で処理されるが、各ブロックを担当する SMX や、各ブロック内のスレッドを処理する CUDA コアを明示的には指定できない。(3.1) の例では 1 ブロックあたり 256 個のスレッドが配置され、 $N/256$ 個のブロックが投入されるため全体で $N/256 \times 256 = N$ 個のスレッドが実行される。カーネル関数内ではスレッド毎に違う値が入るビルトイン変数を宣言なしで使うことができる。プログラム 3.2 の中で使われている `blockDim.x` にはブロック中のスレッドの数が入っており、(3.1) の例の場合は `blockDim.x = 256` となる。ビルトイン変数 `blockIdx.x` にはそのスレッドが何番目のブロックの中で実行されるかの値が入る。そのため (3.1) では $0 \sim N/256 - 1$ のどれかの整数になる。また、`threadIdx.x` にはそのスレッドがブロック内の何番目であるかが入るため (3.1) では $0 \sim 255$ の値になる。図 3.2 にビルトイン変数と配列要素の関係を示す。これらのビルトイン変数を利用し、 $i = \text{blockDim.x} * \text{blockIdx.x} + \text{threadIdx.x}$ は $0 \sim N - 1$ の N 個の配列要素に対して異なる値を取らせることができる。従って、 N 個のスレッドが全て異なる配列要素にアクセスすることができ、プログラム 3.1 と同じ内容を GPU で実行することができる。

3.2 単一 GPU による粒子法計算

3.2.1 粒子のデータ格納形式

粒子法計算では各粒子において速度や座標、圧力など多数の従属変数が定義される。C/C++ 及び CUDA によるプログラミングではこれらの従属変数をメンバ変数として保持する構造体 (粒子構造体) を用いるのが一般的である。粒子構造体には Array of Structure (AOS), 及び Structure of Array (SOA) の 2 種類のデータ格納形式が考えられる。AOS 型では各粒子についてプログラム 3.3 のような粒子構造体が定義され、それらがメモリ上に連続的に確保される。一方、SOA 型ではプログラム 3.4 のように各従属変数の配列を指すポインタをメンバ変数として保持する粒子構造体が全体でひとつだけ定義され、各ポインタに対してそれぞれの従属変数の配列が要素数 (=粒子の個数) だけ連続的に確保される。AOS 型に基づく実装ではある粒子の従属変数の一要素にアクセスする場合にも、その従属変数をメンバ変数に保持する粒子構造体の全体がレジスタにコピーされてしまうため、しばしばレジスタ溢れの原因となる。本研究では SOA 型のデータ格納形式に基づく実装を行う。プログラム 3.3 とプログラム 3.4 では改良型 SPH 法の単一 GPU 計算で使用する粒子構造体を例として示している。第 3.3 節以降で説明する複数 GPU 計算の場合には、これらの従属変数のほかに領域番号を格納する整数型の変数などが付加される。

プログラム 3.3 AOS (Array of Structure) 型のデータ格納形式の粒子構造体

```

1  struct Particle {
2      double x, y, z;           // 座標
3      double tmp_x, tmp_y, tmp_z; // 仮の座標
4      double u, v, w;         // 速度
5      double tmp_u, tmp_v, tmp_w; // 仮の速度
6      double pres;           // 圧力
7      double tmp_pres;      // 仮の圧力
8      double dens;          // 密度
9      int p_type;           // 粒子の種類 (壁, 流体, 物体)
10     int index;            // Linked-list で使用する同一セル内の粒子の番号を格納する変数
11 }

```

プログラム 3.4 SOA (Structure of Array) 型のデータ格納形式の粒子構造体

```

1  struct Particle {
2      double *x, *y, *z;       // 座標
3      double *tmp_x, *tmp_y, *tmp_z; // 仮の座標
4      double *u, *v, *w;     // 速度
5      double *tmp_u, *tmp_v, *tmp_w; // 仮の速度
6      double *pres;         // 圧力
7      double *tmp_pres;    // 仮の圧力
8      double *dens;        // 密度
9      int *p_type;         // 粒子の種類 (壁, 流体, 物体)
10     int *index;          // Linked-list で使用する同一セル内の粒子の番号を格納する変数
11 }

```


3.2.2 近傍粒子探索

各粒子の持つ速度や座標，圧力などの時間積分により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づいた演算を行う．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では接触判定の後に反発力と摩擦力としてバネとダッシュポットを仮定するモデルに基づき相互作用を行い，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，図3.3のように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導入する[45]．これにより相互作用計算における演算量を $O(N)$ に抑えることができる．

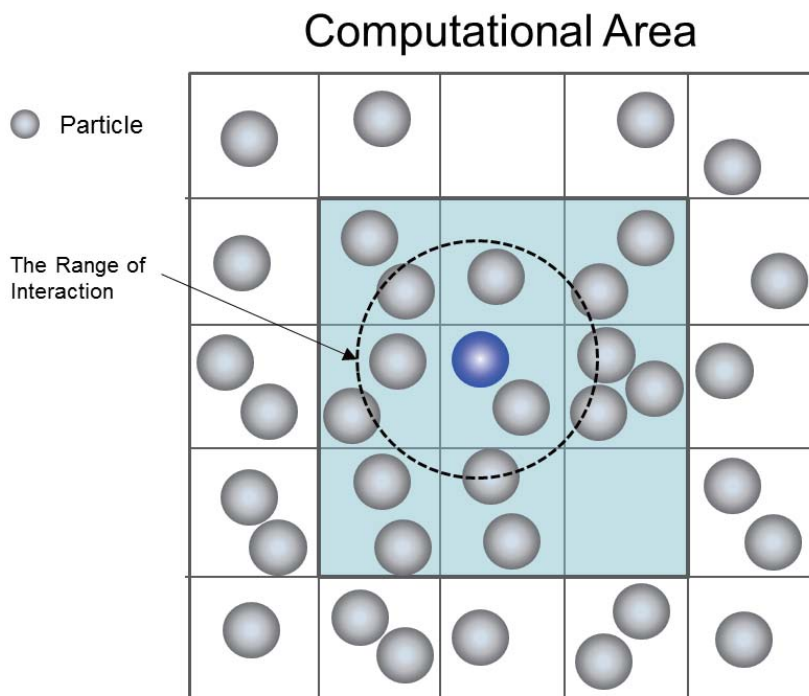


図 3.3 均一空間格子を用いた近傍粒子探索

3.2.3 Linked-list 法

DEM を用いた粉体計算では接触する粒子の間で相互作用が働くので、セルの幅が粒子直径よりも小さい場合、実際には隣接するセルのもう一つ隣のセル内の粒子とも接触する可能性が出てしまう。しかし、セル幅を直径よりも大きく設定するにつれてセル内の粒子数が増えて相互作用コストが増加するので、セル幅はできるだけ粒子直径に近い方が良い。したがって、DEM による粉体計算ではセル分割法の一セルの辺の長さを粒子直径に設定するのが一般的となっている。2次元計算ではセル幅が粒子直径の正方形形状の格子に、3次元計算ではセル幅が粒子直径の立方体形状の格子に格納できる粒子数を考えるので、ひとつのセルに収容される最大の粒子数は2次元計算で4個、3次元計算で8個となる。一方、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ので、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、各セルは最大収容数を想定して粒子番号を格納するメモリ領域を持っておく必要がある。そのため2次元計算では格子数の4倍、3次元計算では8倍の数の粒子番号を登録するメモリ領域が必要になり、粒子数と無関係に計算領域を広くとる場合などにメモリ消費量が極端に多くなる [54]。図 3.4 に示すように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がり保持する Linked-list を GPU 上で導入する [61][62][46]。これにより、空間格子に使用するメモリ使用量を粉体計算で 1/8、改良型 SPH 法の計算で 1/27~1/64 程度に抑えることができる (問題によって異なり、Linked-list 法を用いることで、空間格子の消費するメモリ容量は (セルの最大収容数)⁻¹ となる)。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。

3.2.4 セル番号のソートによるデータロードの高効率化

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 3.5 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 3.5 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

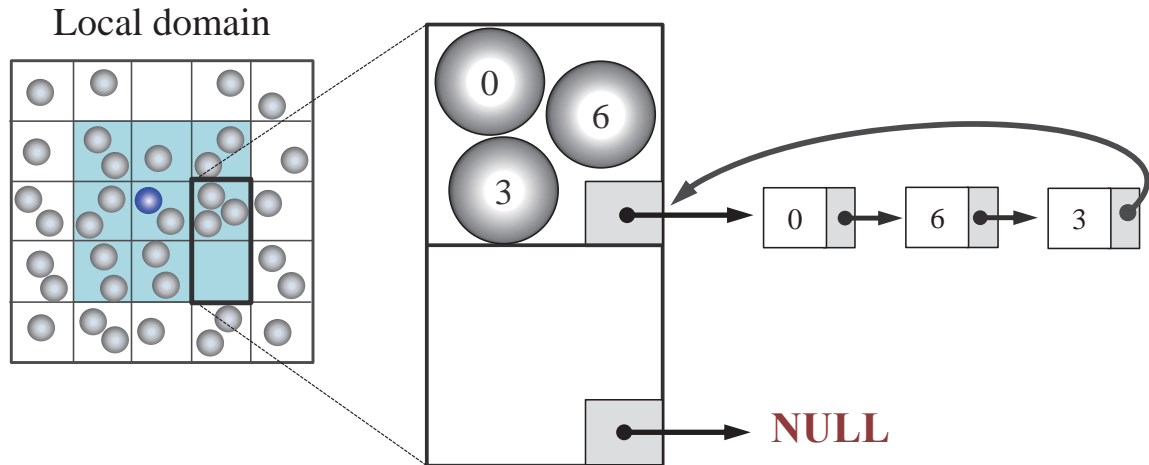


図 3.4 セル分割法への Linked-list 法の適用

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 3.5 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行う。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も多く報告されているが [62][30][28]、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 側から定期的に行う本手法は本研究による新しい提案である。

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 3.6 に検証例として示す。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 3.6 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 回の並び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。

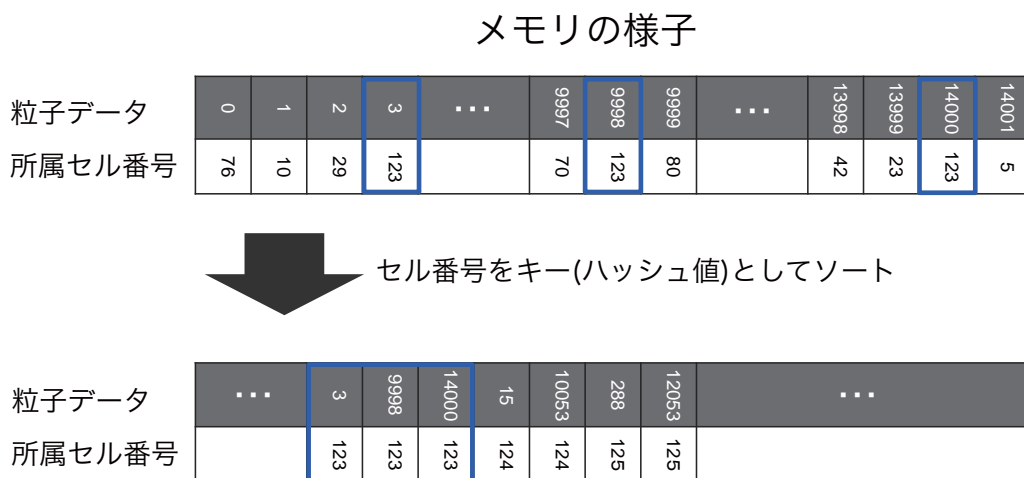


図 3.5 Linked-list に合わせて定期的に行われるセル番号をハッシュ値とした粒子データのソート

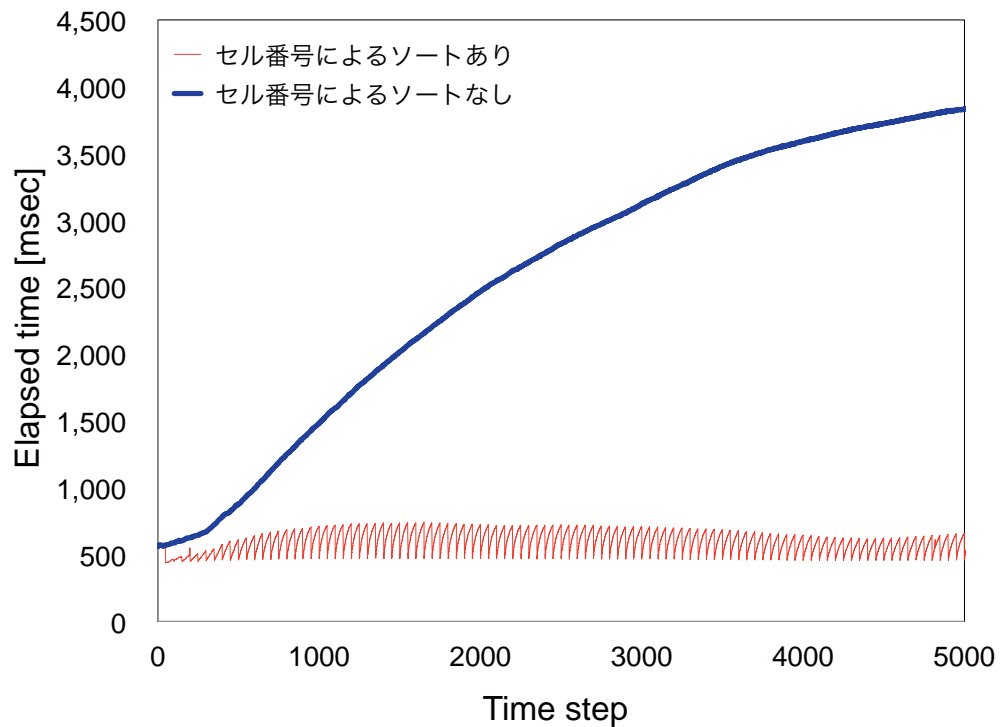


図 3.6 GPU における Linked-list 法を用いた近傍粒子探索にセル番号による粒子データのソートを行う場合と、行わない場合の計算時間の比較

Algorithm 1 The time integration for objects

```

for  $k = 0, 1, 2 \dots$  to  $N_{obj}$  do
  for  $j = 0, 1, 2 \dots$  to  $N_{particle}$  do
    if  $particle[j] \in object[k]$  then
       $object[k].force + = particle.force[j]$ 
       $object[k].torque + = particle.torque[j]$ 
    end if
  end for
  update  $object[k].quaternion$ 
  update  $object[k].position, velocity$ 
end for

```

3.2.5 流体構造連成の GPU 計算

第 3.2.4 節で提案する方法は、第 2.2.3 章で述べた流体構造連成の 4 種類の相互作用計算における流体計算部分 (1. 及び 2.) にそのまま適用できる。一方、物体粒子の時間発展は、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、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。

140 万粒子を用いた改良型 SPH 法の 3 次元ダム崩壊問題に対して計算領域内に一つあたり 1,000 個の物体構成粒子からなる 100~400 個の立方体を加えて配置し、(A) と (B) のそれぞれの実装について 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 3.7 と図 3.8 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。

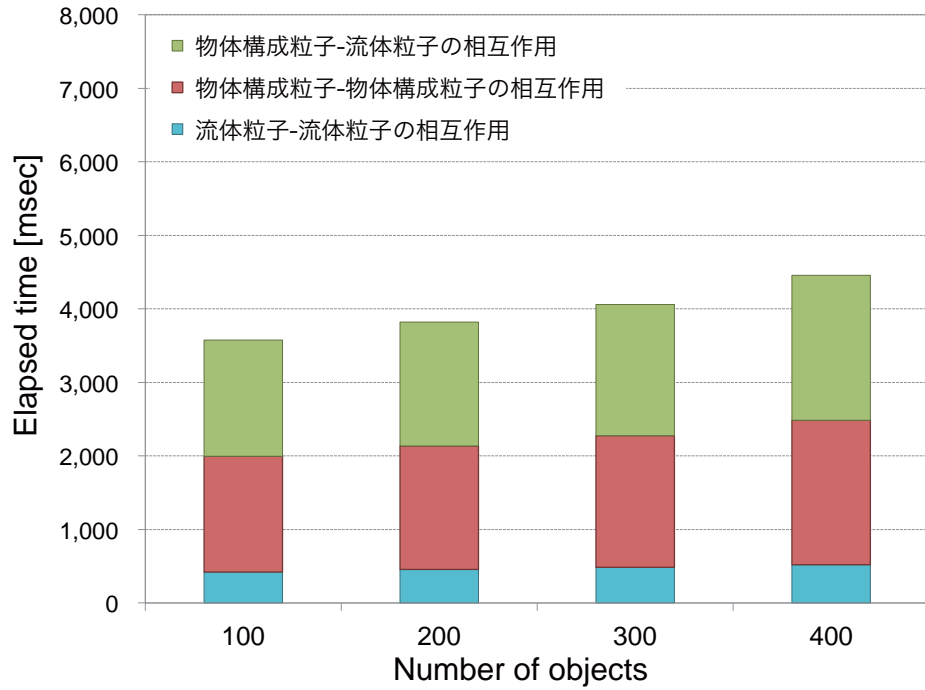


図 3.7 実装 (A) を用いた場合の計算時間の内訳

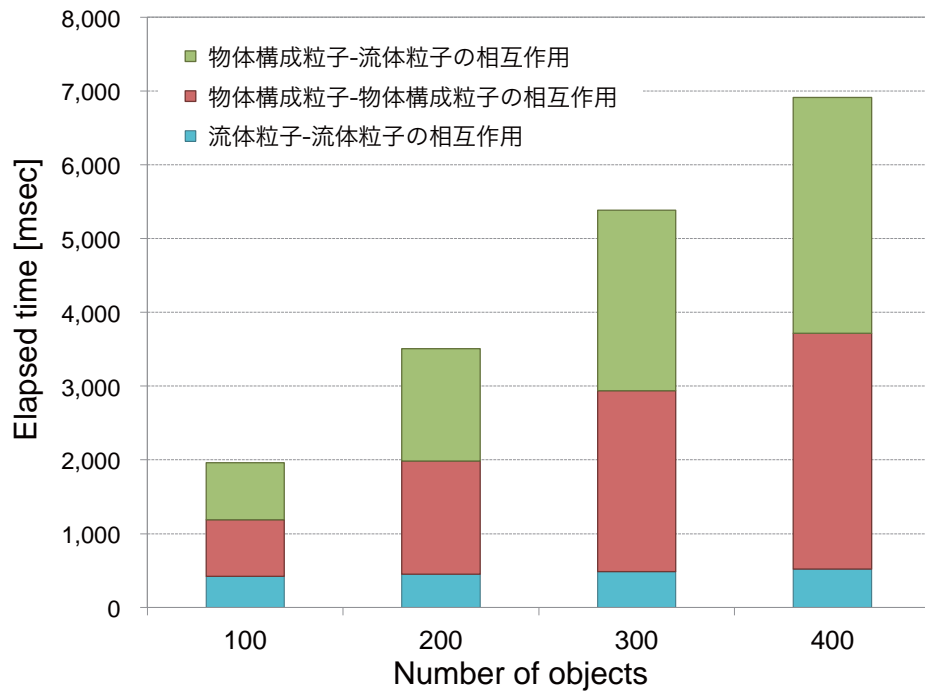


図 3.8 実装 (B) を用いた場合の計算時間の内訳

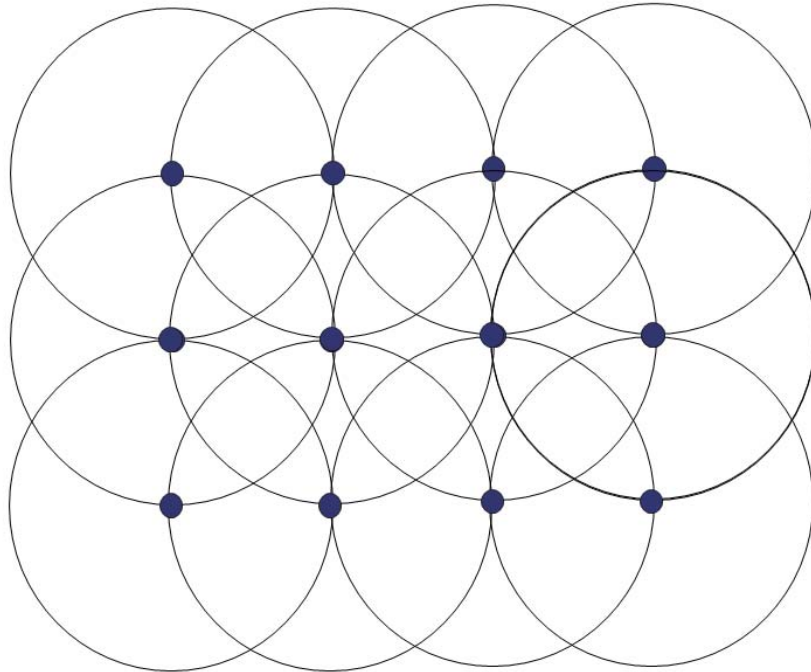


図 3.9 演算密度を一定にする場合の DEM 計算の問題設定

3.2.6 単体 GPU での実行性能

DEM 計算における演算密度 (FLOP/Byte) の算出

粒子を図 3.9 のように体心立方構造状にオーバーラップさせて 3 次元空間に配置し、粒子位置を固定して演算密度 (FLOP/Byte) を一定とした理想的な場合の DEM 計算を考える。本問題では粒子位置を固定しているので、浮動小数点演算回数 (FLOP) とメモリアクセス量 (Byte) を算出することができる。例えば、粒子は図 3.9 のように配置されているので、近傍探索のための空間格子のセル幅を粒子直径としたとき、一つのセルあたりに 3 次元空間では 8 個格納されていると考えられる。自身と隣接するセル内の粒子とのみ接触判定を行うため、接触判定を行う粒子の個数は次のようになる。

$$8 \text{ (個)} \times 27 \text{ (セル)} - 1 \text{ 個} = 215 \text{ (個)} \quad (3.2)$$

式 (3.2) で接触判定を行う粒子のうち、実際に相互作用する粒子の個数はオーバーラップしている粒子の個数であるので、次のようになる。

$$27 \text{ (個)} - 1 \text{ 個} = 26 \text{ (個)} \quad (3.3)$$

単一 CPU で動作する DEM の逐次計算コードに対して、式 (3.2), 式 (3.3) に従い浮動小数点演算回数を算出した。はじめに、粒子間の相互作用計算を行う計算コードはプログラム 3.5 の様に見える。ここで、real は単精度計算では float, 倍精度計算では double を表す。また numPrtcs は総粒子数を表す。それぞれの関数における具体的な処理の中身はコメントで説明している。各関数の FLOP と Byte については、浮動小数点演算回数 (メモリアクセス回数) の形でコロンの後に記している。

プログラム 3.5 DEM の相互作用計算部分の計算コード

```

1 for (int id = 0, id < numPrtcs; id++) {           // 粒子のイタレーション
2   init_force (...);                               // 力の初期化                      : 2 (3)
3   int cell_id [27];                               // 隣接セル番号リスト
4   set_cell_id_list (cell_id [27], ...);          // 周期境界の隣接セル番号の設定      : 0 (3)
5   for (int j = 0; j < 27; j++) {
6     while (true) {                                // cell_id [j] の linked-list をたどる
7       real length = get_overlap_length (...);     // 2 粒子間の距離の計算接触判定 ()   : 29 (6)
8       if (length > 0){
9         calculate_force (...);                  // 力の計算                          : 26 (9)
10      }
11    }
12  }
13 }

```

式 (3.2) より 215 個の粒子と接触判定を行い、26 個の粒子と相互作用計算を行うので、1 粒子あたりの浮動小数点演算回数は以下の様に算出できる。

$$29 \times 215 + 26 \times 26 + 2 = 6,913 \quad (3.4)$$

同様に、メモリアクセス量についても以下の様に算出できる。

$$(6 \times 215 + 9 \times 26 + 6) \times 4 \text{ Byte} = 6,120 \quad (3.5)$$

プログラム 3.5 を実行する段階では、あらかじめ近傍粒子探索リストの各セルに対して Linked-list 構造が構築されている必要がある。これは以下のプログラム 3.6 の手順で行うことができる。プログラム 3.6 から、粒子当たりの浮動小数点演算回数は 9 回、メモリアクセス量は $(3 + 6 + 5) \times 4 \text{ Byte} = 56 \text{ Byte}$ と求めることができる。

プログラム 3.6 DEM の Linked-list 構築部分の計算コード

```

1 for (int id = 0, id < numPrtcs; id++) {           // 粒子のイタレーション
2   set_cellid_to_prtcs (...);                     // 粒子の属するセル番号の決定      : 9 (3)
3 }
4 for (int id = 0, id < numPrtcs; id++) {           // 粒子のイタレーション
5   generate_linear_list (...);                    // 線形リストの生成                  : 0 (6)
6 }
7 for (int id = 0, id < numPrtcs; id++) {           // 粒子のイタレーション
8   connect_linear_list (...);                     // 線形リストの先頭と最後尾をつなげる : 0 (5)
9 }

```

時間積分には、ここでは簡単のため前進オイラー法を用い、1 粒子当たりの浮動小数点演算回数は 21 回、メモリアクセス量は 84 Byte であった。以上から表 3.1 の様になり、DEM 計算コードについて FLOP/Byte=1.1 と算出できた。表 3.1 からは浮動小数点演算回数、メモリアクセス量ともに粒子間相互作用に要することも確認できる。

表 3.1 各計算項目における浮動小数点演算回数と、メモリアクセス量

計算項目	FLOP	Byte	FLOP/Byte
Linked-list の構築:	9	56	-
粒子間相互作用:	6,913	6,120	-
時間積分:	21	84	-
合計:	6,943	6,260	1.1

ダムブレイク問題に対する DEM 計算の実行性能

PAPI (Performance Application Programming Interface)[63] は CPU での逐次計算コードに対して、実行時間、浮動小数点演算回数 (FLOP)、それらの比である浮動小数点演算性能 (Flops : Floating-point Operations Per Second) を測定することのできる性能解析ツールのひとつである。z 方向を重力方向とし、計算領域 20 m × 32 m × 30 m の隅に 4 m × 8 m × 10 m の粉体の柱を 2,560,000 個の粒子を用いて設置した粉体のダムブレイク問題に対する実行性能を、CPU (Xeon 5670) の 1 core と GPU (NVIDIA Tesla K20X) で比較する。簡単のため時間刻みを 1×10^{-20} 以下に設定して粉体の移動がほぼ無視できるようにした理想的な状態で実行性能を測定する。

CPU (Xeon 5670) の 1 core で計算を行い、1 ステップあたりの実行時間、浮動小数点演算回数 (FLOP)、及び浮動小数点演算性能 (Flops) を PAPI により測定した。次に同一の計算を GPU (NVIDIA K20X) を用いて行い実行時間を測定した。CPU の 1 core で PAPI により測定された浮動小数点演算回数 (FLOP) を GPU でも同様に行われた演算回数とみなして GPU での実行時間で除することにより CPU と GPU の両方に対して Flops (Floating-point Operations Per Second) を得た。測定結果をそれぞれ図 3.10 に示す。その結果、CPU では 1.35 GFlops、GPU では 199 GFlops となり、GPU を用いることで CPU の 1 core に対して最大 135 倍程度高速化されることが分かった。

ダムブレイク問題において得られた GFlops 値の妥当性を検討するため、Roofline Model[64] による評価を行った。Roofline モデルはあるプロセッサの理論演算性能と理論バンド幅から算出した、ある演算密度 (Flop/Byte) のアプリケーションが達成しうる最大性能の曲線を示すモデルであり、以下のように定義される。

$$P = \frac{F/B}{F/B + F_{peak}/B_{peak}} F_{peak} \quad (3.6)$$

ここで、 F はそのアプリケーションにおける浮動小数点演算回数、 B はメモリアクセスによる通信量 [GByte]、 F_{peak} は理論演算性能 [Gflops]、 B_{peak} は理論バンド幅である。本研究で用いる NVIDIA Tesla K20X の場合、 $F_{peak} = 3950$ Gflops であり、 $B_{peak} = 250$ GByte

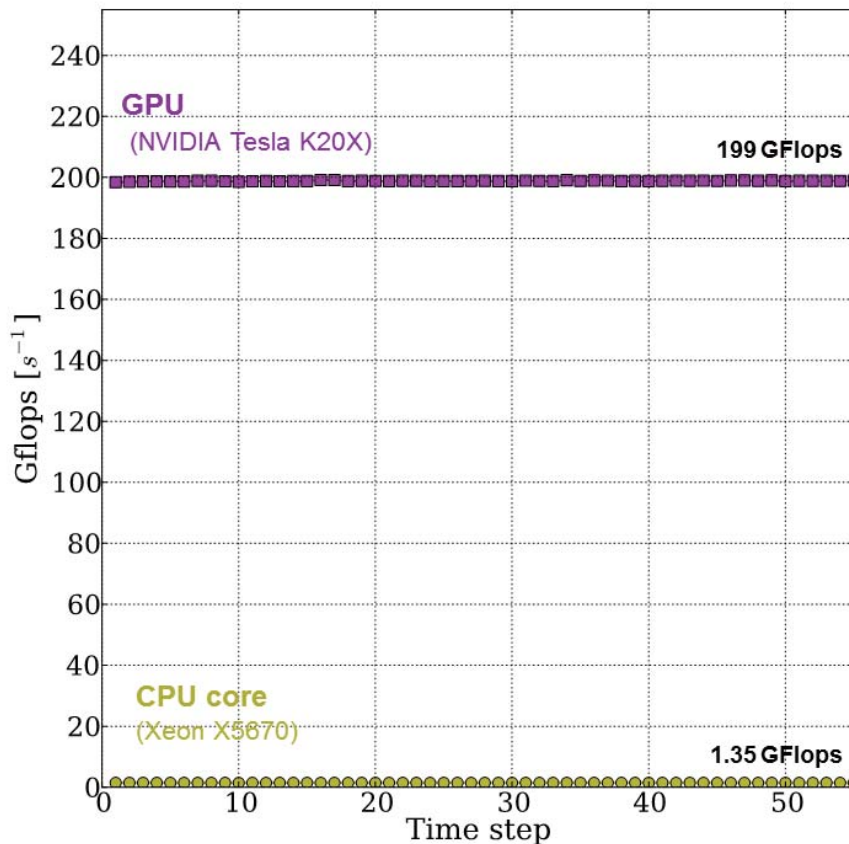


図 3.10 ダムブレイク問題に対する DEM 計算の実行性能の比較

であるので、Roofline Model による曲線は以下のように表される。

$$P = \frac{3950 x}{x + 15.8} \quad (x = F/B) \quad (3.7)$$

図 3.11 に示すように、FLOP/Byte が 1.1 の場合には Roofline Model から算出される最大性能は 257.1 Gflops と確認できる。ベンチマーク問題であるダムブレイク問題に対する実行性能の測定値は 199 Gflops であり、Roofline モデルに対して 77.4% の達成度であることがわかる。また、Roofline Model から算出される最大性能の 257.1 Gflops という値は、NVIDIA K20X の理論ピーク性能の 3950 Gflop に対しては 6.6% 程度であり、DEM 計算の実行性能の低さが確認できる。

流体計算の実行性能

単体 GPU の 3 次元流体計算について、NVIDIA 社の提供する性能解析ツールである Visual Profiler[65] 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。図 3.7 の一番右と同一問題 (140 万個の流体粒子に対して、1,000

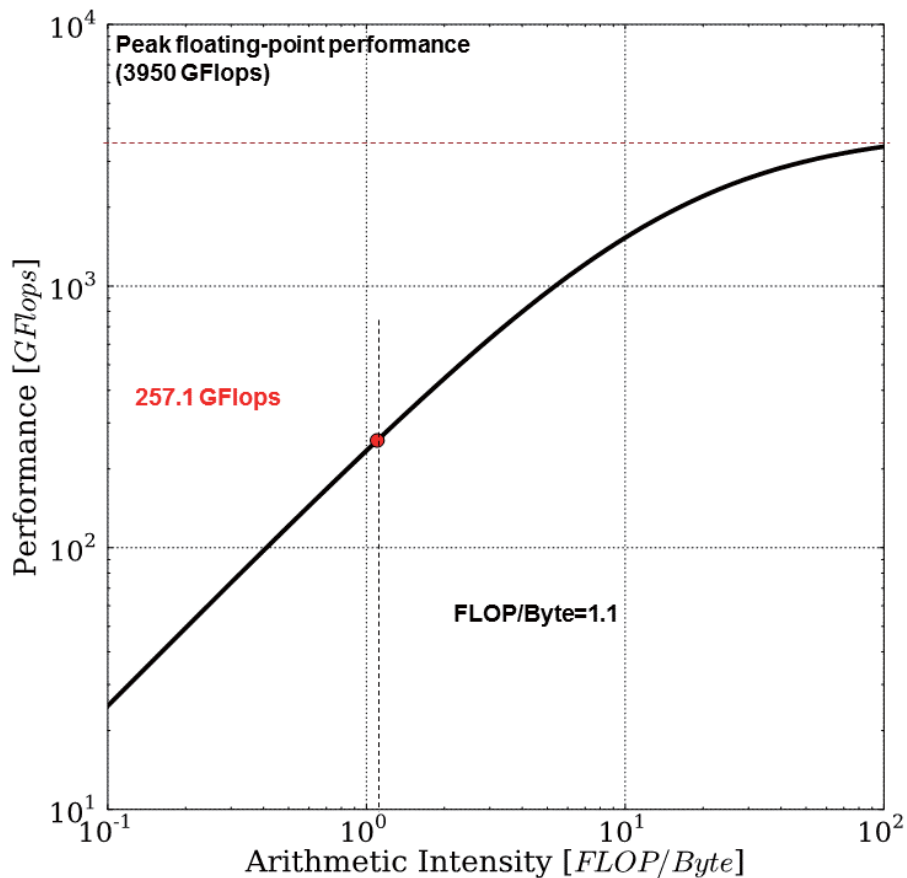


図 3.11 Roofline Model による実行性能の妥当性の検証

個の粒子から構成される立方体を 400 個配置したダムブレイク問題) に対する実装 (A) を用いた場合の実行性能の内訳を図 3.12 に示す。

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 3.7 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 (2.57) の \tilde{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 3.12 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 3.2 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

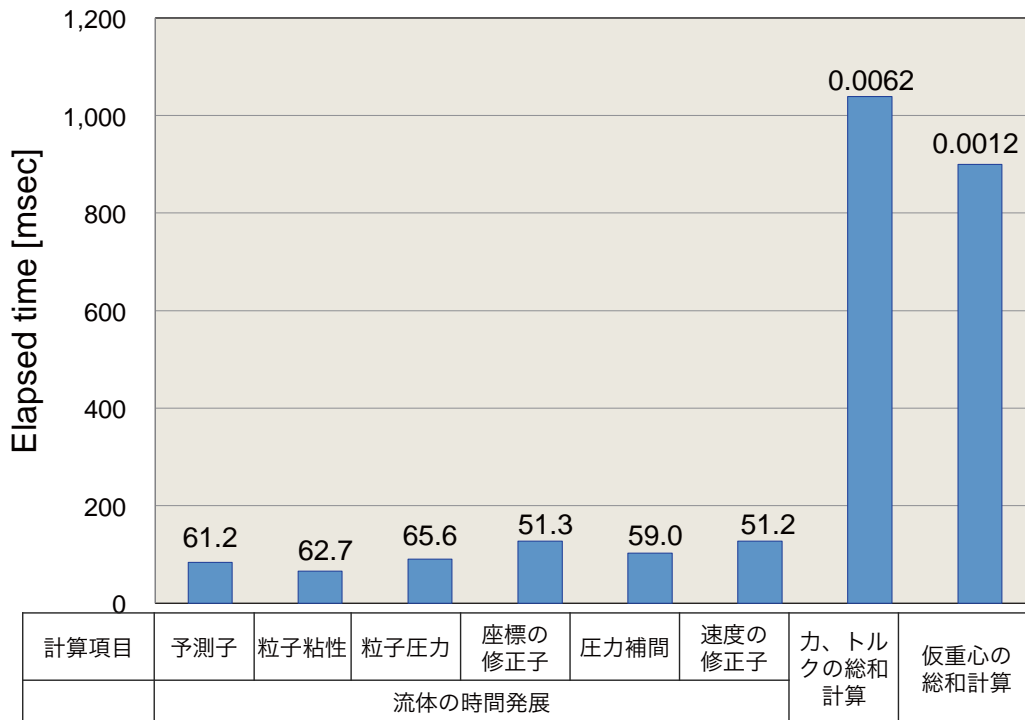


図 3.12 単体 GPU での実行性能 (棒グラフ上部の数値は GFlops 値)

表 3.2 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

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 (%))

3.12 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、流体計算でも Roofline Model[64] などを用いた検討が必要になる。

3.3 複数 GPU を用いた粒子移動の計算

計算領域を小領域に空間分割し、分割された各小領域に GPU を割り当てて計算する。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、動的負荷分散により領域形状が変更されたことにより粒子の所属する小領域が異なった場合、小領域同士が正しく接続されるように袖 (ハロー) 領域 (= 分割領域の境界から、相互作用のレンジ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。

異なるノードに搭載された GPU 間のデータ通信は図 3.13 のように CPU を介して行うことができる。GPU のデバイスメモリ上にある粒子データのうち領域外に出た粒子 (領域外粒子, 図 3.13 における赤枠内の粒子) のみを GPU 上でパッキングしてバッファに集め、

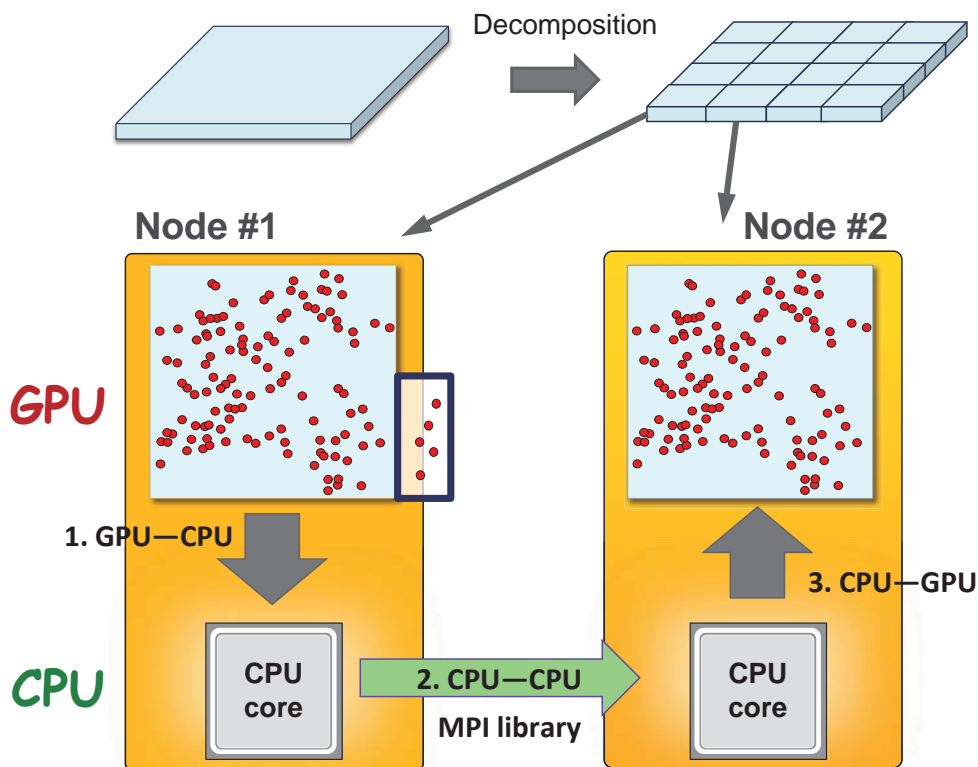


図 3.13 粒子移動の GPU 間通信

PCI Express バスを通して CPU のホストメモリにコピーする。CPU 間では MPI ライブラリを用いて転送する粒子データを相互に通信する。各ノードでは、受信された粒子データを再び PCI Express バスを通じて GPU のデバイスメモリ側にコピーする。

3.3.1 パッシブ・スカラー粒子の複数 GPU 計算

GPU 間での粒子データの通信について、パッシブ・スカラー粒子計算を用いて説明する。パッシブ・スカラー粒子計算とは、与えられた速度場のもとで粒子を移動させる計算であり、粒子法の複数 GPU 計算における粒子が運動したことにより小領域間を移動する場合に相当する。各小領域で行う計算手順を図 3.14 に示す。まずホスト (CPU) 側で、1. **初期条件の設定**を行う。粒子の情報として座標と粒子の所属する小領域の番号を保持するために、各小領域でメモリを確保する。メモリの不参照領域には、参照領域との区別のため、小領域の最大番号より大きな番号をフラグとして格納する。これらをデバイス (GPU) 側のメモリにコピーする。2. **時間発展**では、以下に示す 4 段階ルンゲクッタ法を用いて時

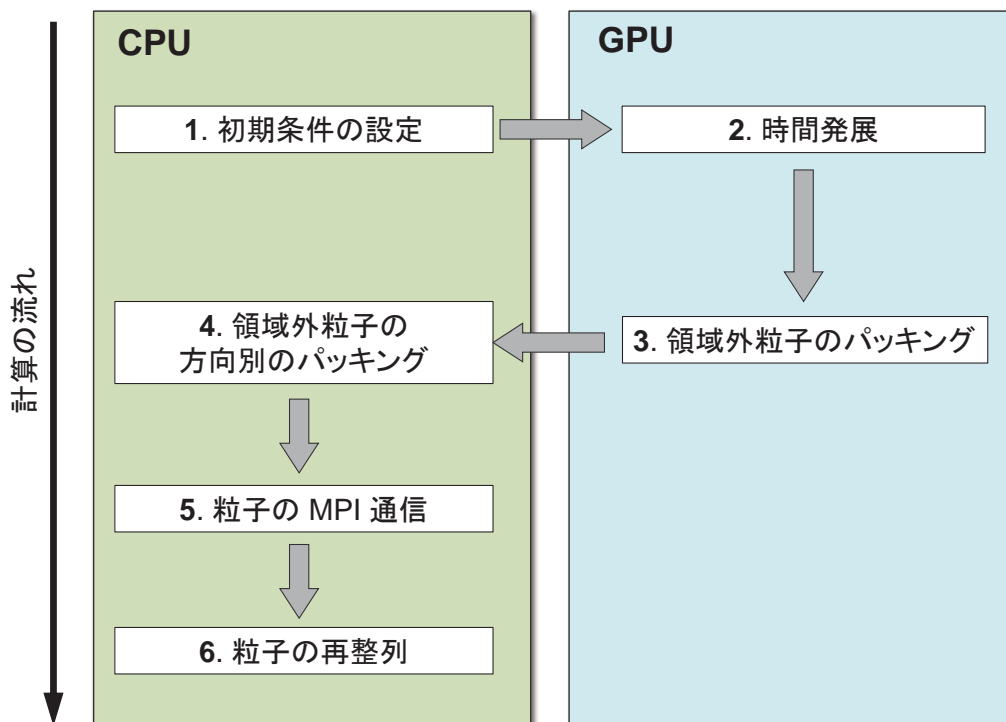


図 3.14 時間積分の手順

間積分する.

$$k_1 = hf(t_n, y_n) \quad (3.8)$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \quad (3.9)$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \quad (3.10)$$

$$k_4 = hf(t_n + h, y_n + k_3) \quad (3.11)$$

$$t_{n+1} = t_n + h \quad (3.12)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (3.13)$$

式 (3.8)~ 式 (3.13) は従属変数 y についての 4 段ルンゲクッタ法の各段を表す. h が時刻 t_n が n ステップ目における時刻である. 時間発展により, 小領域境界を横切り外に移動する粒子 (領域外粒子) が発生すると, 移動先の GPU に転送する必要がある. GPU 間のデータ転送は CPU を介して行うため, 粒子の情報をホスト (CPU) のメモリにコピーする必要がある. 全粒子の情報を GPU のメモリから CPU のメモリにコピーし CPU で領域外粒子の選別を行うと, GPU と CPU 間の通信が大きなオーバーヘッドとなる. 3. 領域外粒子のパッキングでは, GPU - CPU 間の通信回数, 通信量を可能な限り抑えるため, GPU 上で粒子の移動する方向に関係なく全ての領域外粒子を 1 つのバッファに集める. このバッファを CPU へ一度に転送する. ホスト側で受け取ったデータは, 粒子の横切った方向に関係なく領域外粒子すべてを含むので, 4. 粒子の方向別のパッキングにより隣接する小領域の各方向に送信する粒子の選別をホスト側で行う. 隣接する各小領域に 5. 粒子の MPI 通信を行う. 受信したデータをデバイス側にコピーし, メモリの使用領域の最後尾部に加えて通信が完了する.

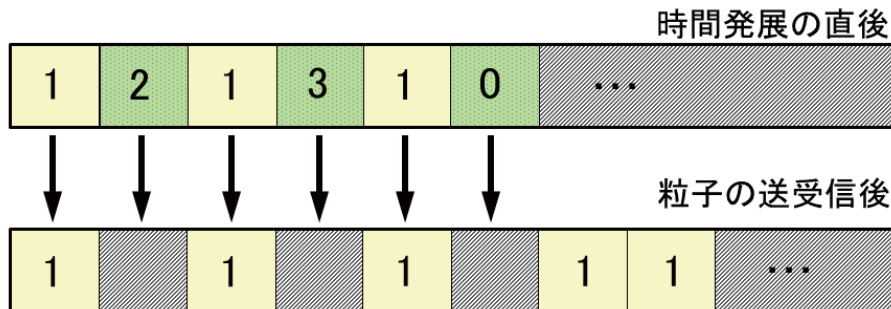


図 3.15 GPU メモリの断片化

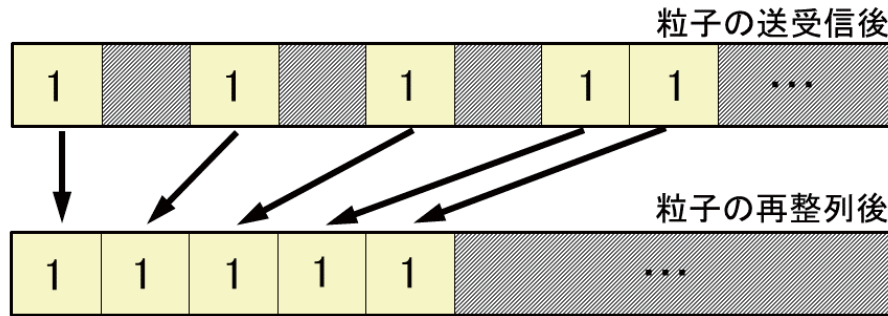


図 3.16 粒子番号の再整列

粒子の再整列

小領域から外に移動した粒子が使用していたメモリは、不参照メモリとなる。計算ステップが進行するにつれて、不参照メモリは増加していく。図 3.15 は 1 番の小領域の断片化を模式的に表したものである。図中の数字は小領域の番号を表す。黄色が自分の小領域内の粒子が使用しているメモリを、灰色が不参照メモリを表す。緑色が次のステップで他の小領域に移動する粒子を表し、粒子の送受信後に不参照メモリとなる。また、隣接小領域からの粒子の送受信後のメモリでは、バッファの最後尾が新たに流入した粒子が使用する領域になる。GPU を用いる場合、効率的に計算するためスレッドは連続したメモリを参照することになる。このとき、粒子データが有効か無効かは関係なくメモリ参照は行われるため、不参照メモリの部分には不要なデータの読み込みとそれに伴う条件分岐が入る。このため、使用メモリの断片化は GPU 計算の性能を低下させる。メモリの断片化を防ぐため、6. 粒子の再整列をホスト側で行う。図 3.16 はそれを模式的に表したものである。粒子の属する小領域の番号表すバッファにおいて、不参照メモリには小領域の最大番号より大きな数字が書き込まれていて、ソートを行うことにより不参照メモリはバッファの最後尾に集められる。

ソートは逐次処理を多く含むため、GPU よりも CPU の方が適している場合が多い。図 3.17 は粒子を粒子構造体のメンバのひとつである所属領域の番号でソートする場合の計算時間を、CPU と GPU とで比較したものである。(A) は C++ の STL (Standard Template Library) による CPU でのソートの時間を、(B) は (A) に CPU - GPU 間通信を合わせた時間を。(C) は THRUST ライブラリ [66] による GPU でのソートの時間を表す。CUDA は Version 4.0 を、CPU によるソートに対しては Intel コンパイラ (Version 11.1) を用いている。GPU のソートは、CPU 上でのソートと比較して 15 倍近く計算時間がかかっており、CPU でのソート自体にかかる時間と CPU-GPU 間通信の時間を合わせても、GPU でのソートにかかる時間より短いことが分かる。本論文では粒子の再整列のためのソートを

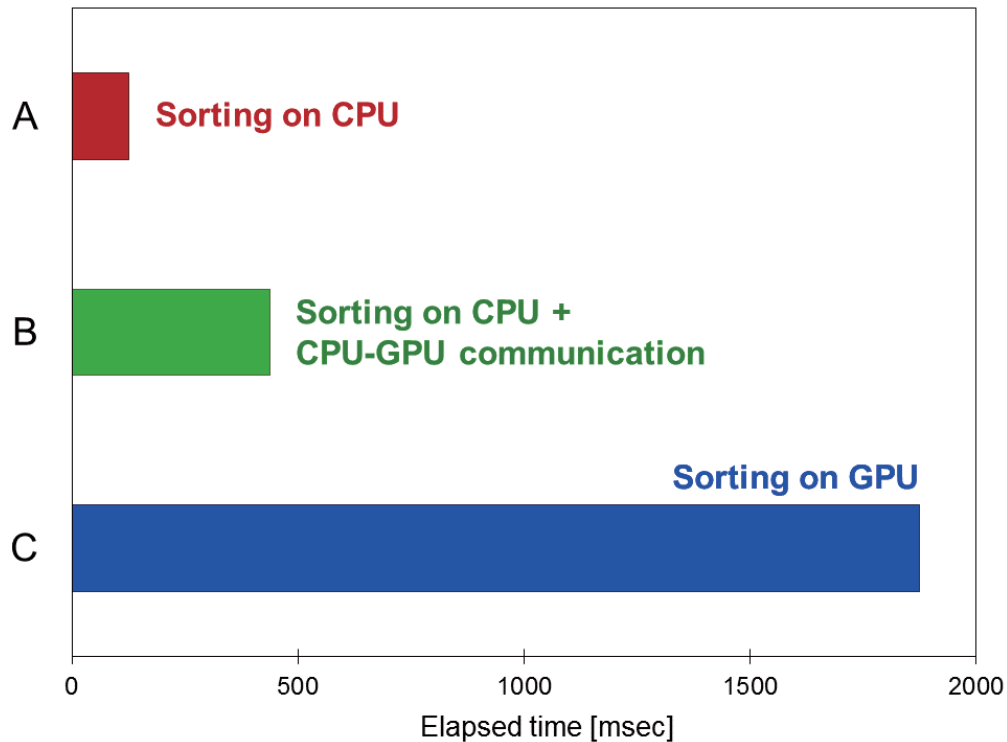


図 3.17 4,194,304 個の粒子を粒子構造体のメンバ (所属領域の番号) でソートする場合の CPU と GPU の計算時間の比較

CPU で行う。GPU-CPU 間で全粒子の配列をコピーする必要があり計算のオーバーヘッドになるため、ソートの回数を制御する必要がある。

3.3.2 均一分布・一様速度場を用いた性能評価

計算領域全体に渡り粒子が均一に分布し、一様な速度場で移動する場合は、時間が経過しても粒子分布が変化せず、各小領域の計算負荷は均一に保たれる。そのような計算では、小領域の境界を横切り隣接小領域へ移動する粒子についての GPU 間データ転送とメモリの断片化が計算の顕著なオーバーヘッドとなる。本節ではそのような動的負荷分散を必要としない計算条件のパッシブスカラー粒子計算に対して、弱スケーリングと強スケーリングによる性能評価を行う。強スケーリングでは、粒子の再整列を行う場合と行わない場合のそれぞれについて計算時間を測定する。さらにソートの頻度を変えて計算時間を測定し両者の関係を調べる。

弱スケーリング

1 プロセスあたりの粒子数を固定し、プロセス数に比例して粒子数を増加させる。全粒子数はプロセス数に比例して増加する。計算規模が拡大しても並列化効率が保たれることを確認する。

1. **実行環境** 東京工業大学学術国際情報センターの TSUBAME 2.0 を用いる。1 ノードあたり、GPU は NVIDIA 社の Tesla M2050 が 3 基、CPU は Intel 社の Xeon X5670 が 2 個搭載されている。1 ノードにつき 2 core, 2 GPU を利用する。CUDA は Version 3.2 を用いる。
2. **計算条件** 2 次元の計算領域に均一に分布させた粒子を一定速度場 ($u = v = 1.0$) のもと 4 段ルンゲクッタ法により時間発展させる。周期境界条件を用いて計算負荷を絶えず均等にする。1 GPU の担当する小領域は、 x 方向の長さを $Lx = 1.0$ 、 y 方向の長さを $Ly = 1.0$ とする。1 GPU あたりの粒子数は $4,194,304 (= 2^{22})$ 個とする。初期条件では全計算領域に均一に粒子を分布させる。計算領域の分割方法は、 x 方向、 y 方向の分割数の等しい 2 次元領域分割とする。時間ステップは、 $\Delta t = 0.001$ とする。500 ステップの計算時間を測定する。ソートは 500 ステップのうち、250 ステップ目に 1 回行う。初期条件の設定は測定に含めない。

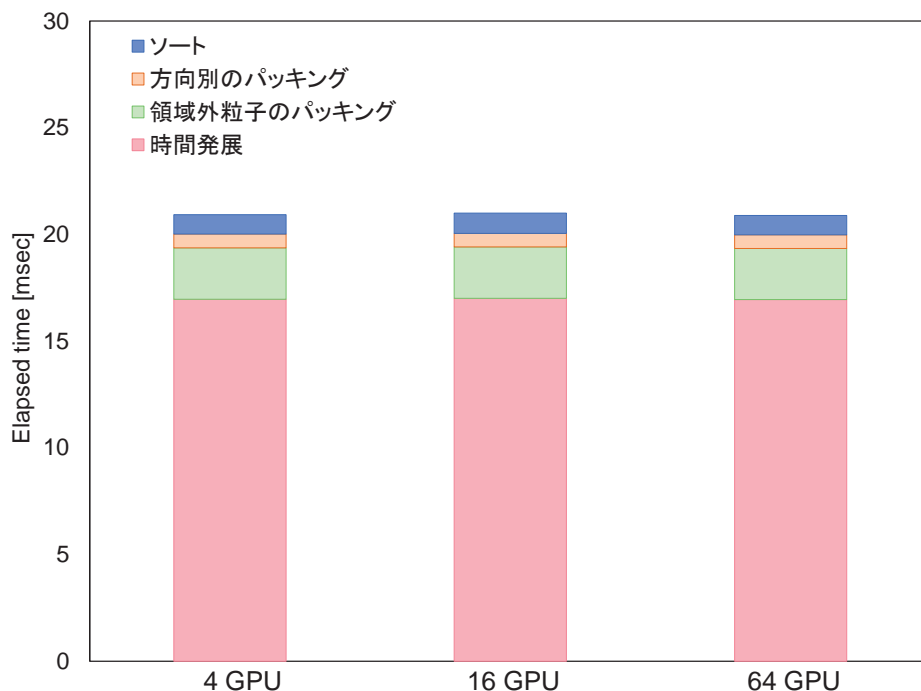


図 3.18 均一分布・一定速度場を用いた弱スケーリング

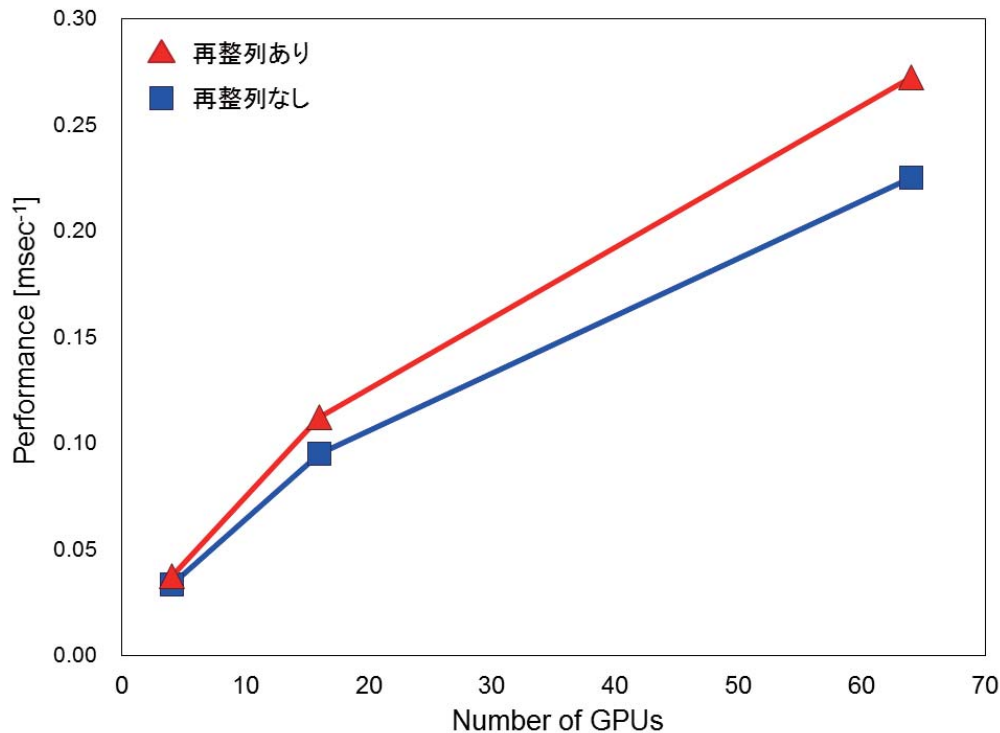


図 3.19 均一分布・一定速度場を用いた強スケーリング

3. **計算時間** 4 GPU から 64 GPU まで変化させたときの計算時間を図 3.18 に示す。赤色は 4 段ルンゲクッタ法による時間発展に要する時間を、緑色は GPU での領域外粒子のパッキングに要する時間を、橙色は CPU での転送方向別のパッキングに要する時間を示している。青色は粒子の再整理処理のためのソートを 1 回実行するのにかかる時間である。領域外粒子のパッキング (緑色) に要する時間は全体の 1 割程度であり、複数 GPU 化により必要となる処理のオーバーヘッドは小さいことがわかる。この検証問題では 64 GPU を用いた時点で粒子数は 2 億 5000 万個以上になる。並列計算の最小規模である 4 GPU の計算時間に対する、複数 GPU を用いた場合の計算時間の割合を効率と定義すると、64 GPU を用いた場合に並列化効率 97% 以上となり、良好な弱スケーリングが達成されている。

強スケーリング

全体の粒子数を一定にし、GPU 数を変化させてそれぞれの場合の計算時間を測定する。1 台あたりの GPU が計算する粒子数は GPU 数が増えるにつれて減少する。

1. **計算条件** 2 次元の計算領域に均一に分布させた粒子を一定速度場 ($u = v = 1.0$) のもと 4 段ルンゲクッタ法により時間発展させる。周期境界条件を用いて計算負荷

を絶えず均等にする．1ステップの間に，1GPUあたりの粒子の移動量は16,000個程度となるように Δt を設定している．粒子数は総粒子数を16,777,216 ($= 2^{44}$)個で固定する，1500ステップにかかる計算時間を測定する．計算領域の分割方法は， x 方向， y 方向の分割数の等しい2次元領域分割とする．測定時間を全ステップ数で割り1ステップあたりの計算時間を算出し，逆数の値を実行性能とする．測定対象には初期条件の設定は含めない．

2. **性能** 測定結果を図3.19に示す．赤線が100回に1度ソートを行う場合の性能を，青線がソートを行わない場合の性能を表す．ソートを行う場合に性能が向上することが確認できる．1GPUに対して，4GPUで2.0倍，16GPUで6.2倍，64GPUで15.6倍の性能向上を達成している．

4GPUの性能を測定する時の，1ステップあたりの計算時間の変化を図3.20に示す．計算が進むにつれて増加する1ステップあたりの計算時間が，ソートをかける毎に元に戻ることが確認できる．図3.20の0ステップ目の計算時間が長い理由は，0ステップ目でカーネル関数を立ち上げる際，GPUの電力がまだアイドル状態にあり，立ち上がるのに時間がかかるためである．図3.20では，CPUでのソート実行のためのGPU-CPU間通信により生じる270msec前後のピークを，見やす

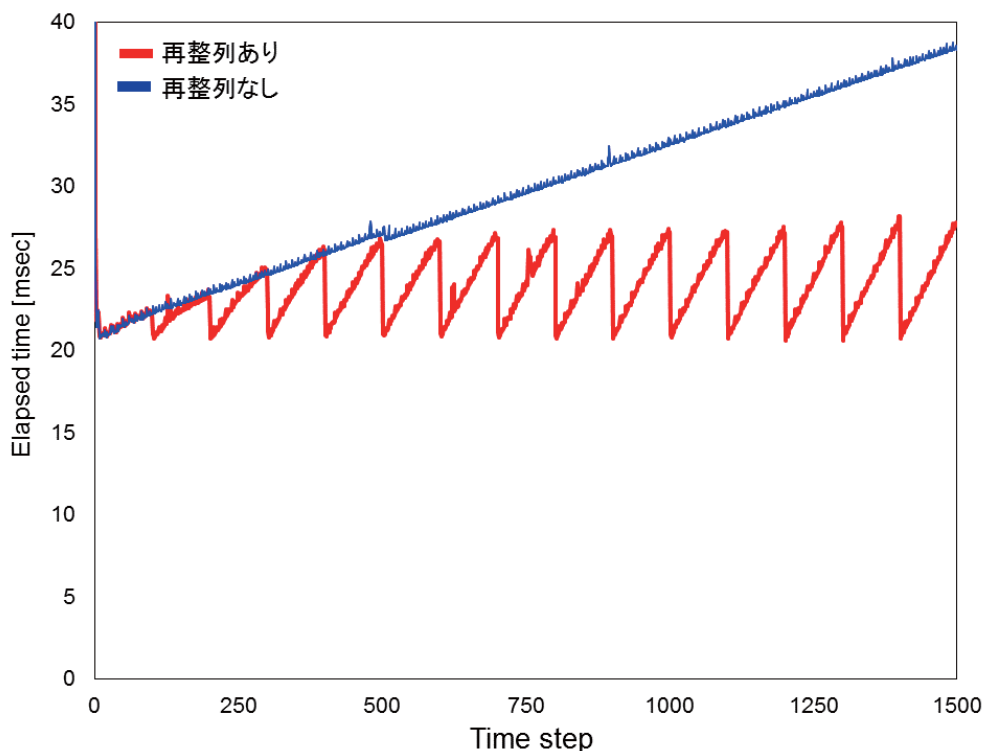


図3.20 4GPUを用いて100回に1回ソートを行う場合(赤)と，ソートを行わない場合(青)の1ステップの計算時間の変化

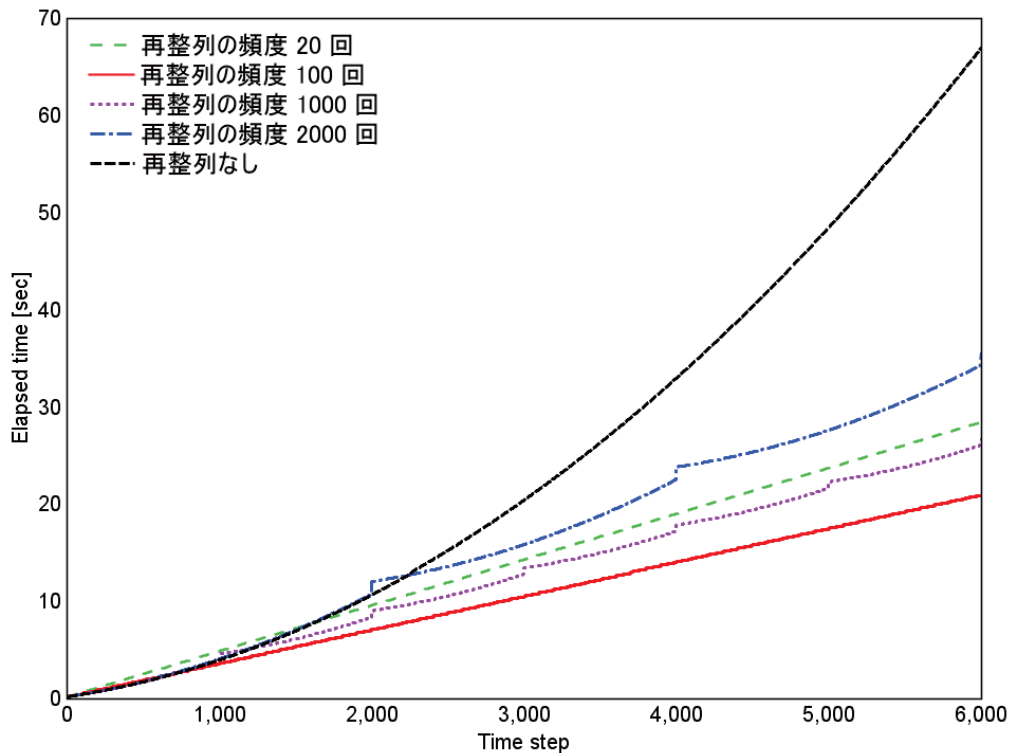


図 3.21 64 GPU を用いた場合のソート頻度の違いによる積算計算時間の変化の比較

さのため図中には示していない。

ソートの頻度と計算時間の関係

図 3.21 に、ソートの頻度が異なる 5 通りの場合について、64 GPU を用いる場合の積算計算時間の変化を示す。それぞれ、ソートを行わない場合 (黒線)、20 回に 1 回行う場合 (緑線)、100 回に 1 回行う場合 (赤線)、1000 回に 1 回行う場合 (紫線)、2000 回に 1 回行う場合 (青線) を表す。計算条件は図 3.20 と同様である。ソートの頻度を上げるにつれて計算時間は線形変化に近づき、頻度が 100 回に 1 回の場合に、行わない場合の $1/3$ 以下となる。頻度が 20 回の場合、頻度が 100 回に 1 回の場合よりも計算時間がかかる。この中では頻度が 100 回に 1 回の場合が、最適解の近傍と予想できる。

3.3.3 再整理の頻度のモデル化の提案

第 3.3.2 節では、粒子の再整理を定期的に行うことにより積算計算時間が線形変化となることを示した。本節では再整理の頻度と計算時間の関係のモデル化を行い、計算時間を最小化するための再整理の最適頻度について考察する。はじめに、ソートを行わない場合

と行う場合のそれぞれについて計算時間のモデル化を行う。

再整列を行わない場合

i ステップ目の計算時間を T_i とする。各スレッドで同様の計算を行う場合、GPUの計算時間は、スレッド数に比例する。0ステップ目では、スレッド数は粒子数に N_0 に等しいので、比例定数を α として T_0 は次のように書ける。

$$T_0 = \alpha N_0 \quad (3.14)$$

1ステップ目の粒子数 N_1 は、流入した粒子数を ΔN_{in} 、流出した粒子数を ΔN_{out} とすると、 $N_1 = N_0 + \Delta N_{in} - \Delta N_{out}$ である。 ΔN_{out} はメモリに離散的に存在する不参照メモリであるため、スレッドは、 $N_0 + \Delta N_{in}$ だけ起動する。よって第一次近似的には計算時間は $N_0 + \Delta N_{in}$ に比例する。しかし、不参照メモリは粒子データの読み込みのメモリアクセスだけで、書き込みのメモリアクセスは行わないため、GPUにかかる全体の負荷は、すべてのスレッドが粒子の実データを扱う場合よりは軽くなる。そこで、計算時間は、不参照メモリ数 ΔN_{out} に比例する量だけ短縮されると仮定する補正項を導入する。 T_1 は、 β を補正項にかかる係数として次のように表せる。

$$T_1 = \alpha(N_0 + \Delta N_{in}) - \beta \Delta N_{out} \quad (3.15)$$

一定速度場より $\Delta N_{in} = \Delta N_{out} = \Delta N$ であるから、式 (3.15) は次のようになる。

$$T_1 = \alpha(N_0 + \gamma \Delta N) \quad (\gamma = \frac{\alpha - \beta}{\alpha}) \quad (3.16)$$

0ステップ目と比較して、2ステップ目は1ステップ目と2ステップ目の合計の流入量 ($=2\Delta N_{in}$) だけ粒子の実データが増加し、1ステップ目と2ステップ目の合計の流出量 ($=2\Delta N_{out}$) だけ不参照メモリが発生している。計算時間 T_2 は、式 (3.15) で、 ΔN_{in} 、 ΔN_{out} がそれぞれ2倍になるため、次のように書ける。

$$T_2 = \alpha(N_0 + (2\Delta N_{in})) - \beta(2\Delta N_{out}) \quad (3.17)$$

式 (3.16) と同様に整理すると、式 (3.17) は次のようになる。

$$T_2 = \alpha(N_0 + 2\gamma \Delta N) \quad (3.18)$$

3ステップ目以降についても同様であるので、 i 番目のステップ1回の計算時間 T_i は次のようになる。

$$T_i = \alpha(N_0 + i\gamma \Delta N) \quad (3.19)$$

m 番目のステップまでの計算時間の積算値 $T_{sum}(m)$ は、式 (3.19) の m 番目までの和であり、次のようになる。

$$T_{sum}(m) = \sum_{j=0}^m T_j = \sum_{j=0}^m (A + j2B) \quad (3.20)$$

ここで、 $\alpha N_0 = A$ 、 $\alpha\gamma\Delta N/2 = B$ とした。式 (3.20) は等差数列である。 m 項までの和を求めると、次のようになる。

$$T_{sum}(m) = A + (A + B)m + Bm^2 \quad (3.21)$$

再整列を行う場合

再整列を S 回に 1 回 行う場合を考える。再整列の際に発生する GPU - CPU 間通信や再整列 1 回にかかる計算時間の合計時間を、 T_{sort} とする。再整列をかける度に計算時間は 0 ステップの計算時間に戻るため、 m ステップ目までの積算の計算時間は、式 (3.20) の S ステップ目までの和 m/S 回の合計と、 m/S 回の T_{sort} の和であるため、以下のように表せる。

$$T_{sum}(m) = \left(\frac{m}{S}\right) \sum_{j=0}^S T_j + \left(\frac{m}{S}\right) T_{sort} \quad (3.22)$$

式 (3.19) を代入すると次のようになる。

$$T_{sum}(m) = \left(\frac{m}{S}\right) \sum_{j=0}^S \alpha(N_0 + j\gamma\Delta N) + \left(\frac{m}{S}\right) T_{sort} \quad (3.23)$$

$\alpha N_0 = A$ 、 $\alpha\gamma\Delta N/2 = B$ より、式 (3.23) は次のようになる。

$$T_{sum}(m) = \left(A + (S + 1)B + \frac{A + T_{sort}}{S}\right)m \quad (3.24)$$

係数 A, B の決定

α 、 γ は、粒子数やレジスタの占有率などの様々な要因に依存するパラメータであるため、未定係数の A, B を関係式 $A = \alpha N_0$ 、 $B = \alpha\gamma\Delta N/2$ から一意に決定することは難しい。そこで、ソートを行わない場合の計算時間の測定値に対して、式 (3.21) を用いて最小二乗法によるフィッティングを行い A, B を決定する。

図 3.22 は、図 3.21 のソートを行わない場合の測定値の初期の 500 ステップを、式 (3.21) によりフィッティングした結果である。赤線が測定値を、黒線がフィッティング曲線を表し、 $A = 2.3319$ 、 $B = 0.0015$ と求まる。

測定値とモデル値の比較

測定値と、式 (3.21), 式 (3.24) から決定される $T_{sum}(m)$ を比較する. ソートを行わない場合, 100 回に 1 回ソートを行った場合のそれぞれの測定結果と, $T_{sum}(m)$ の比較を図 3.23 に示す. 黒の実線が図 3.21 のソートを行わない場合の, 赤の実線が 100 回に 1 回ソートを行う場合の測定値である. 黒色の点線, 赤色の点線はそれぞれ式 (3.21), 式 (3.24) による予測結果である. 図 3.23 に示す通り, モデルの曲線と測定値の曲線の傾向はよい一致を示している. 再整列を行わない場合は式 (3.21) のようにステップ数 m の 2 乗に比例して計算時間が増加し, 再整列を行う場合は式 (3.24) のように, ステップ数 m に比例することがわかる.

最適解の決定

再整列の回数が最適値となる条件は, 式 (3.24) が最小値をとることである. 式 (3.24) が極小値をとる必要条件として,

$$\frac{\partial T_{sum}(m)}{\partial S} = Bm + (A + T_{sort})\left(\frac{-m}{S^2}\right) = 0 \quad (3.25)$$

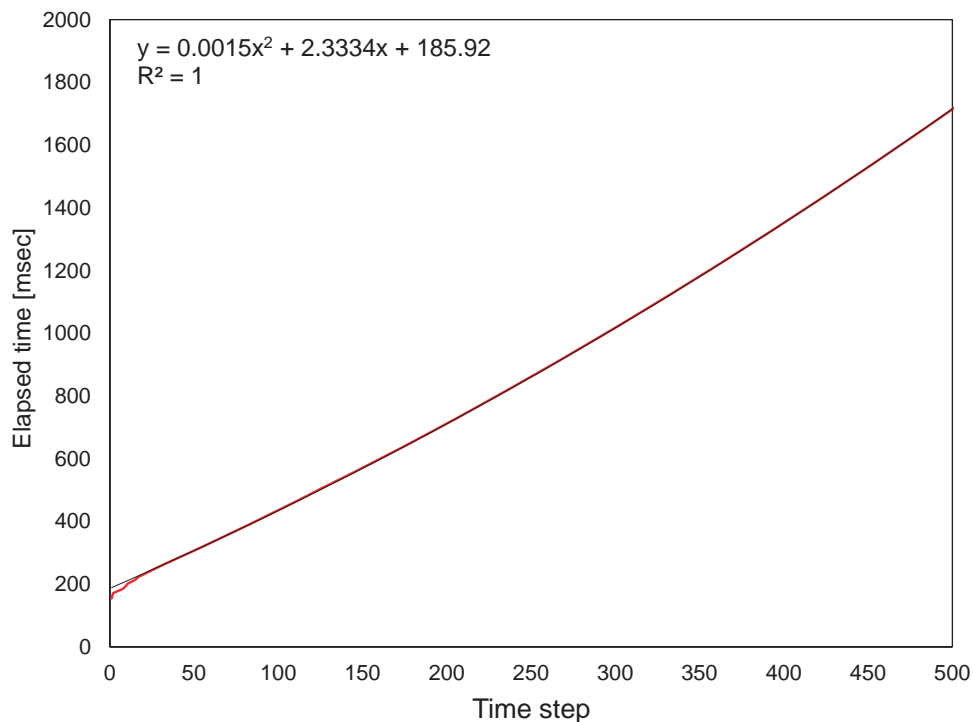


図 3.22 図 3.21 のソートを行わない場合の測定値を, 式 (3.21) によりフィッティングした結果

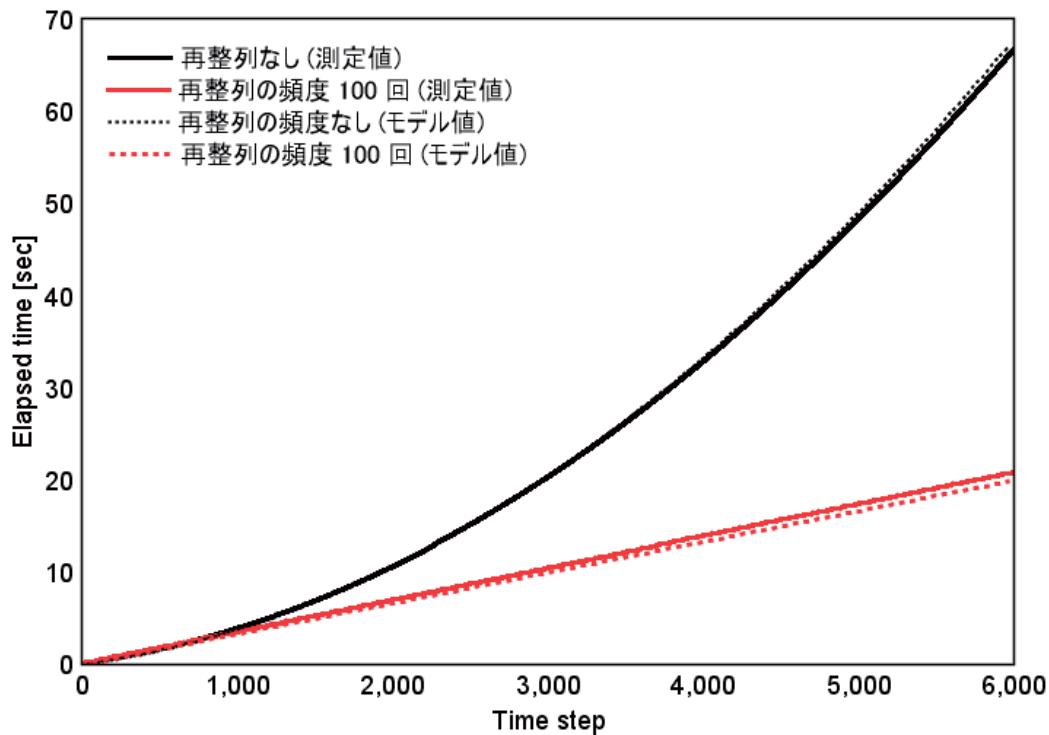


図 3.23 図 3.21 の測定値とモデル値の比較

を S について解くと、最適値 S_{opt} は、

$$S_{opt} = \sqrt{\frac{A + T_{sort}}{B}} \quad (3.26)$$

となる。式 (3.26) に、 A 、 B 、別途測定した $T_{sort} = 85.4$ msec を代入すると、 $S_{opt} = 238$ 回となる。図 3.21 から 238 回の頻度でソートを行った場合の積算の計算時間は、ソートの頻度が 1000 回の場合と 100 回の場合の間で、線形変化することが推定できる。測定結果の中の最小値である 100 回の場合に近い結果であり、妥当な値である。

式 (3.26) から決定された最適な実行頻度 (ここでは、238 回) の値の妥当性についてより検討するため、再整列の頻度を 100 から 50 回刻みで変化させて、6000 ステップにかかる計算時間をそれぞれ測定した結果を図 3.24 に示す。再整列の頻度の最適値は 250 回付近にあることが分かりモデル値の妥当性を示している。

今回のモデルは一定速度場に対するものであり、そのままでは他の計算の場合の再整列の頻度を直接決定できない。本研究では実際の複数 GPU を用いた流体計算の際には、初期の数ステップを計算し、粒子データの再整列のオーバーヘッドも調べて経験的に再整列の実行間隔を決めている。本節で提案したモデルを実際の流体計算にそのまま適用することは現状ではハードルが高いが、DEM や SPH 法でよく行われるような運動方向が決まっ

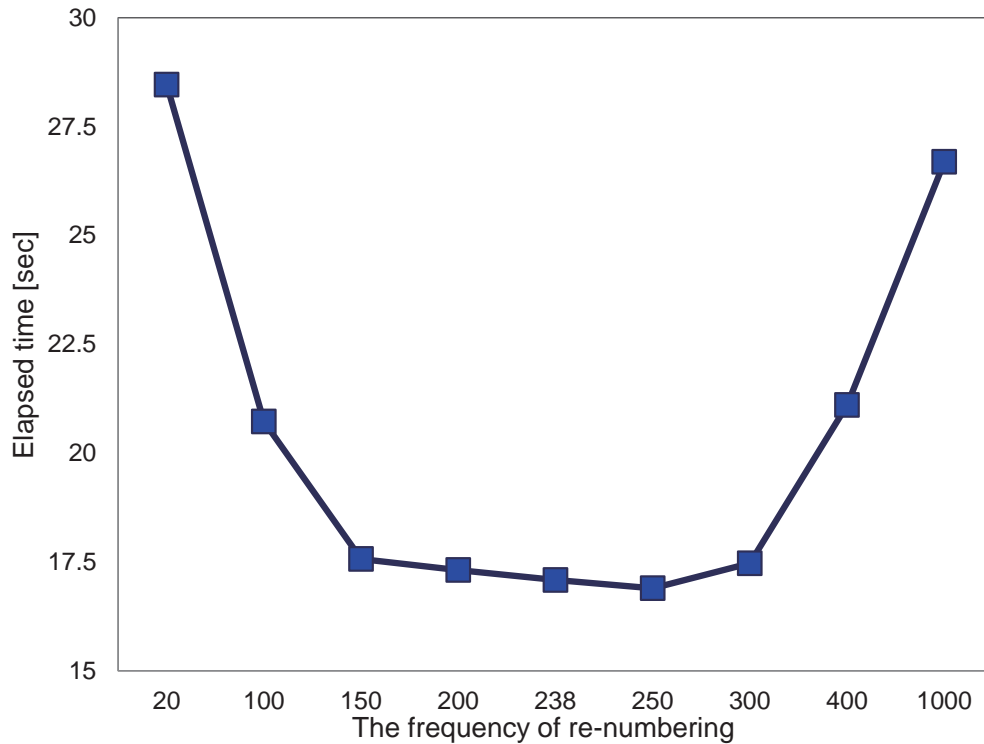


図 3.24 再整列の頻度の違いによる計算時間の変化

ていて慣性支配の現象であるダムブレイク問題や、粒子の速度の空間分布が比較的均一な問題に対しては次のような処理を行うことで利用可能になると考えられる。例えば、定期的に粒子の速度分布を調べその統計平均値や中央値をもとに時間刻み幅を掛けてその計算における平均的な流入量や流出量を算出して、これと測定された再整列のオーバーヘッドから最適な実行間隔をモデルから決定する方法などである。また、速度分布を調べることが高コストになる場合には、各 GPU 上で粒子の流入量と流出量を計測してそれらをマスター領域に転送して流入量と流出量の GPU 間の分布を作成し、それらから同様にして平均的な流入量と流出量を求め、再整列の最適な実行頻度をモデルから決定する方法も考えられる。これらについては、まずは第 4.1.3 節で用いるような渦度速度場のもと粒子を初期にガウス分布させたパッシブ・スカラー粒子計算などのベンチマーク問題で検証し、実問題へのモデルの適応を進めて行く必要がある。

第 4 章

複数 GPU を用いた粒子法シミュレーションの動的負荷分散法

4.1 スライスグリッド法による動的負荷分散

計算領域を小領域に空間分割し、分割された各小領域に GPU を割り当てて計算する。図 4.1 の左側のように各小領域の大きさを均等に分割した場合、時間発展とともに GPU 間の粒子分布の偏りが生じる。そこで、GPU 1 台あたりの粒子数を一定にすることにより計算負荷を均一に保つよう、図 4.1 の右側に示すようなスライスグリッド法 [33][34] による動的な 2 次元領域分割を GPU 間で実現する。

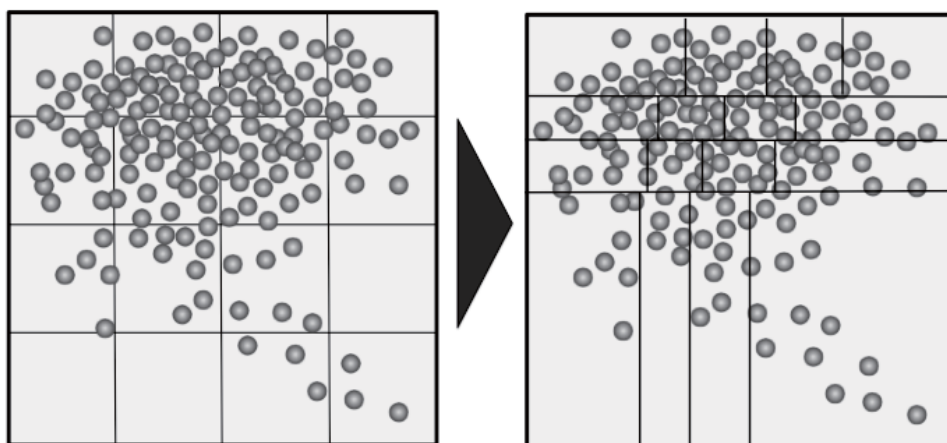


図 4.1 2次元スライスグリッド法による動的領域分割の概要

4.1.1 隣接小領域への転送粒子数の決定方法

i 番目の小領域内の粒子数を N_i としたとき、式 (4.1)、式 (4.2) に従い小領域間で転送する粒子数 ΔN_i を決定する.

$$\Delta N_0 = N_0 - N_{total}/p \quad (4.1)$$

$$\Delta N_i = \Delta N_{i-1} + N_i - N_{total}/p \quad (i \geq 1) \quad (4.2)$$

隣接領域への送信量, ΔN_{i-1} , ΔN_i の正負と粒子の送信方向の関係について図 4.2 に示す. $\Delta N_i > 0$ のとき, i 番目の小領域から $i+1$ 番目の小領域に ΔN_i 個の粒子を送信し, $\Delta N_i < 0$ のときは $i+1$ 番目の小領域から i 番目の小領域に $-\Delta N_i$ 個の粒子を送信する. 効率的な通信を行うため, 式 (4.1), 式 (4.2) で小領域間で転送する粒子数を先に決定し, 実際の粒子データの通信は全 GPU で並列に行う. ここで, 粒子を送信しても小領域内の粒子が 0 未満にならないことが条件であるため, 決定した送信粒子数の和は, 小領域内にある粒子数よりも小さいことが条件となる.

$$\max(\Delta N_i, 0) + \max(-\Delta N_{i-1}, 0) \leq N_i \quad (4.3)$$

粒子数を小領域間で一定にするためには, 境界線の変更にともない隣接小領域に移動す

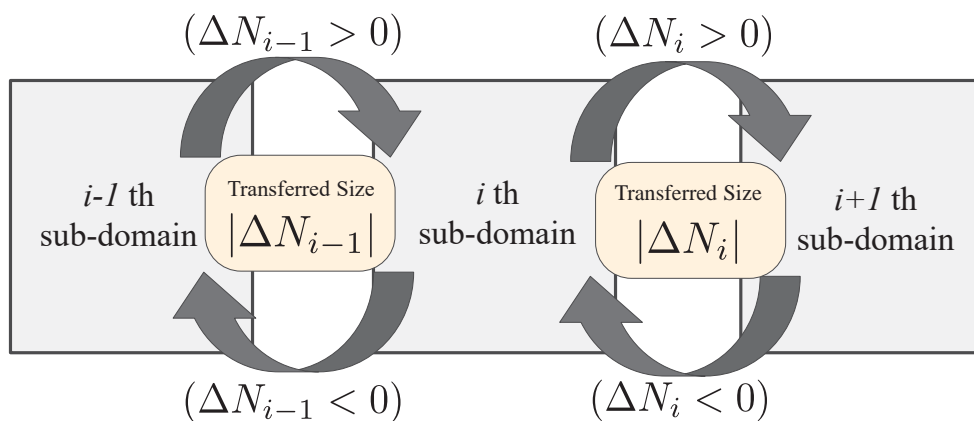


図 4.2 小領域の番号と粒子の通信量の関係

る粒子を GPU 上で探索する。粒子は空間にランダムに分布しているので図 4.3 のように自身の小領域を微小幅 Δl で分割し、各分割内の粒子を数え上げる。

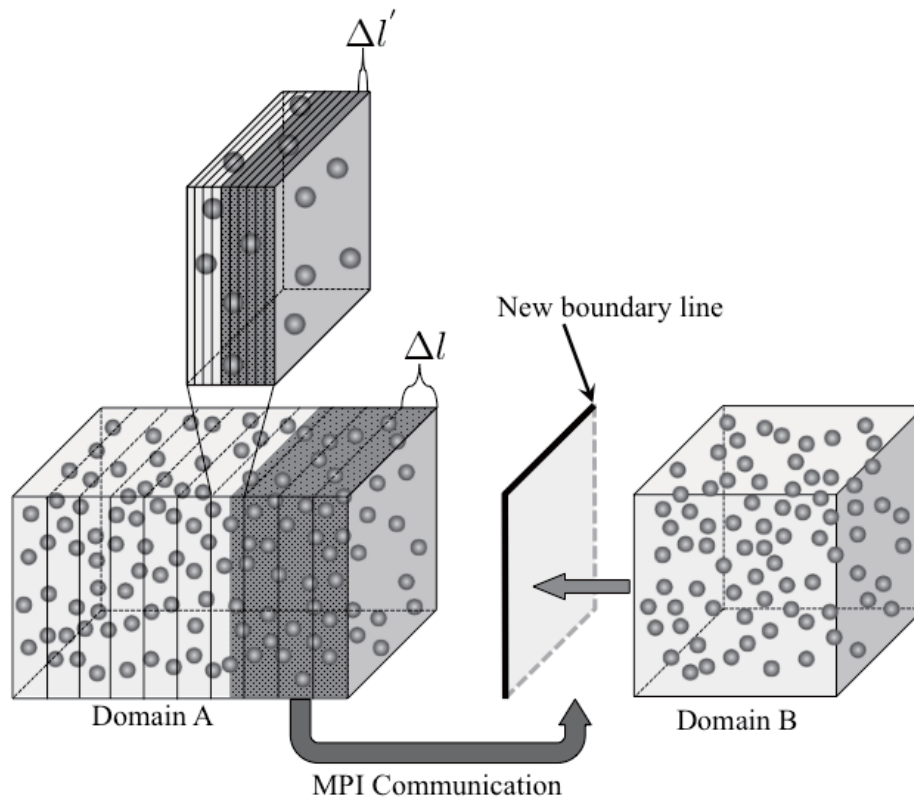


図 4.3 GPU 上での粒子の数え上げと領域境界線の移動

4.1.2 GPU における粒子の数え上げの実装

CUDA を用いた GPU による計算では、1 スレッドが 1 粒子を担当し、各スレッドで担当する粒子の属するセル ID を判定する。粒子に所属セルの ID を割り振り、ID を記した配列型のデータに対して、各 ID の数をそれぞれ数え上げる。GPU 上で排他処理を可能にする `atomic` 関数を利用する方法が考えられるが、GPU の実行性能を著しく低下させる。本論文では `atomic` 関数を使用せず、各 ID について `reduction` 操作のカーネル関数の 1 回の実行で数え上げる方法を提案する。GPU 上での `reduction` の効率的な方法は、Mark Harris らによって示されており [67]、本研究ではこれを各 ID について同時に行うよう変更する。

GPU のシェアードメモリには、セル ID と粒子番号を要素にもつ整数型の 2 次元配列を用意する。2 次元配列には、粒子のセル ID に該当する要素は 1 を、それ以外は 0 を格納する。各 ID について、それぞれブロック内でデータの加算を行う。シェアードメモ

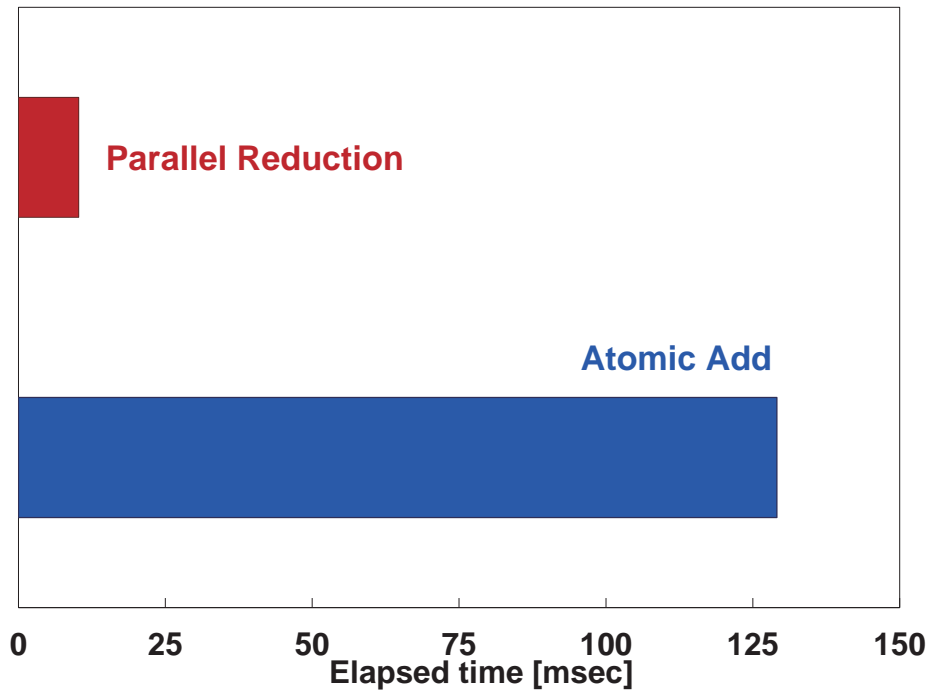


図 4.4 16,777,216 個の粒子に割り振られた 10 種類のセル ID の GPU 上での数え上げにかかる計算時間の比較

りの計算結果を、グローバルメモリに書き込み、CPU のメモリに転送する。シェアードメモリからグローバルメモリに書き込む際、1 次元配列で確保された領域に書き込むことで 1 回の転送回数にできる。CPU 上でブロック毎の計算結果を各セルについて足し合わせ、**reduction** が完了する。ブロック内のシェアードメモリは 16 KB が上限であるため、CUDA の API である `cudaFuncSetCacheConfig` を用いてシェアードメモリを 48 KB まで拡張している。図 4.4 に、16,777,216 個の粒子に割り振られた 10 種類の ID を GPU (NVIDIA Tesla M2050) 上で数え上げるのに要する計算時間を、**atomic** 関数と比較した結果を示す。本実装により 10 倍以上の高速化が確認できる。

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。複数ノードに分散する GPU 間では直接のデータ通信は出来ないため、CPU を介して通信する。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 [32] を用いたデータ通信を行い、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。

隣接する小領域間ではこの他、時間積分の後に境界を横切り領域外に出た粒子の通信や、相互作用計算のため、隣接領域との境界部分にある粒子の隣接領域へのコピーを行う。これらはすべて CPU を介した MPI によるノンブロッキング通信によって、隣接する小領域の間で一斉に通信される。このような頻繁な小領域間でのデータ通信が起こると、粒子

が境界を横切り隣接領域へ移動することによりメモリの断片化が生じるため、第 3.3.1 節で述べたように粒子の再整列を行いメモリの断片化を解消する。

2 次元領域分割への拡張

2 次元領域分割の動的負荷分散では、スライスグリッド法による 1 次元領域分割を垂直方向に行い、次に水平方向に行う。垂直方向の再領域分割では、水平方向に並んだ小領域をひとつのグループとし、グループの粒子数の総和が一定になるよう、垂直方向の境界線を変更する。各グループの左端の小領域を小マスターと呼ぶことにし、小マスターがグループの総粒子数を管理する。隣接する垂直方向の小マスター間で通信を行い、隣接グループに送信する粒子数を決定する。次に、水平方向の再領域分割は、グループ内で各小領域の粒子数が一定になるように、隣接小領域の境界線を変更する。

4.1.3 不均一粒子分布のパッシブ・スカラー粒子計算による性能評価

不均一な粒子分布のパッシブ・スカラー粒子計算に対し、2 次元領域分割に基づくスライスグリッド法による動的負荷分散を用いることで高速化を図る。非一様な速度場に対して不均一な粒子分布を仮定し、強スケーリングによる性能評価を行い、提案する負荷分散法の有用性を確認する。

渦速度場での検証

2 次元に配置したスカラー粒子を、4 段ルンゲクッタ法により時間発展させる。以下に示すような非一様な渦速度場を用いる。

$$u(r, t) = -2 \cos(\pi t/T) \sin(\pi x) \sin(\pi x) \cos(\pi y) \sin(\pi y) \quad (4.4)$$

$$v(r, t) = 2 \cos(\pi t/T) \cos(\pi x) \sin(\pi x) \sin(\pi y) \sin(\pi y) \quad (4.5)$$

T は速度場が時間的に変化する周期であり、 $T = 8.0$ とする。粒子の時間積分のステップは $\Delta t = 0.01$ とする。速度場の一周期は 800 ステップとなり、粒子分布も 800 ステップで初期分布に戻る。GPU は 64 台使用する。計算領域は、 x 方向の長さを $Lx = 1.0$ 、 y 方向の長さを $Ly = 1.0$ とする。 x 方向、 y 方向の分割数の等しい 2 次元領域分割を用いる。

初期条件では、疑似乱数を用いてガウス分布に配置する。粒子数 4096 ($= 2^{12}$) 個とする。各時刻での小領域間の粒子数の比 (最大粒子数 / 最小粒子数) と、そのときの各境界線を求める。動的負荷分散は各ステップで実行する。計算結果のスナップショットを図 4.6 に示す。半周期分に相当する 400 ステップ目までの 100 ステップ毎のスナップショットを示している。渦速度場は周期関数であり、後半の 400~800 ステップについては、図 4.6 の逆過程である。変化する粒子分布を追従して計算領域を再分割していることがわかる。

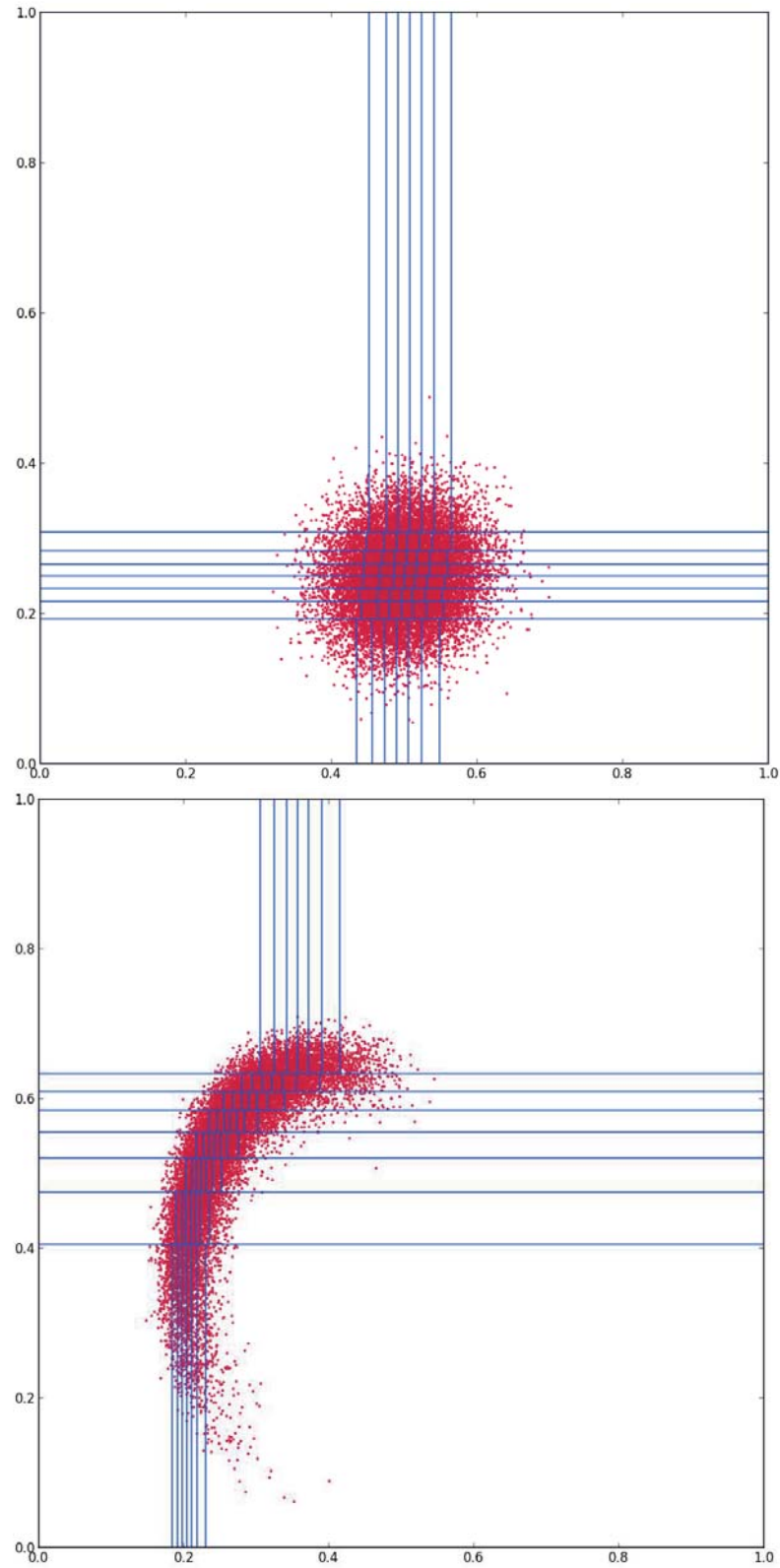


図 4.5 渦度速度場中のスライスグリッド法による動的負荷分散のスナップショット (前半)

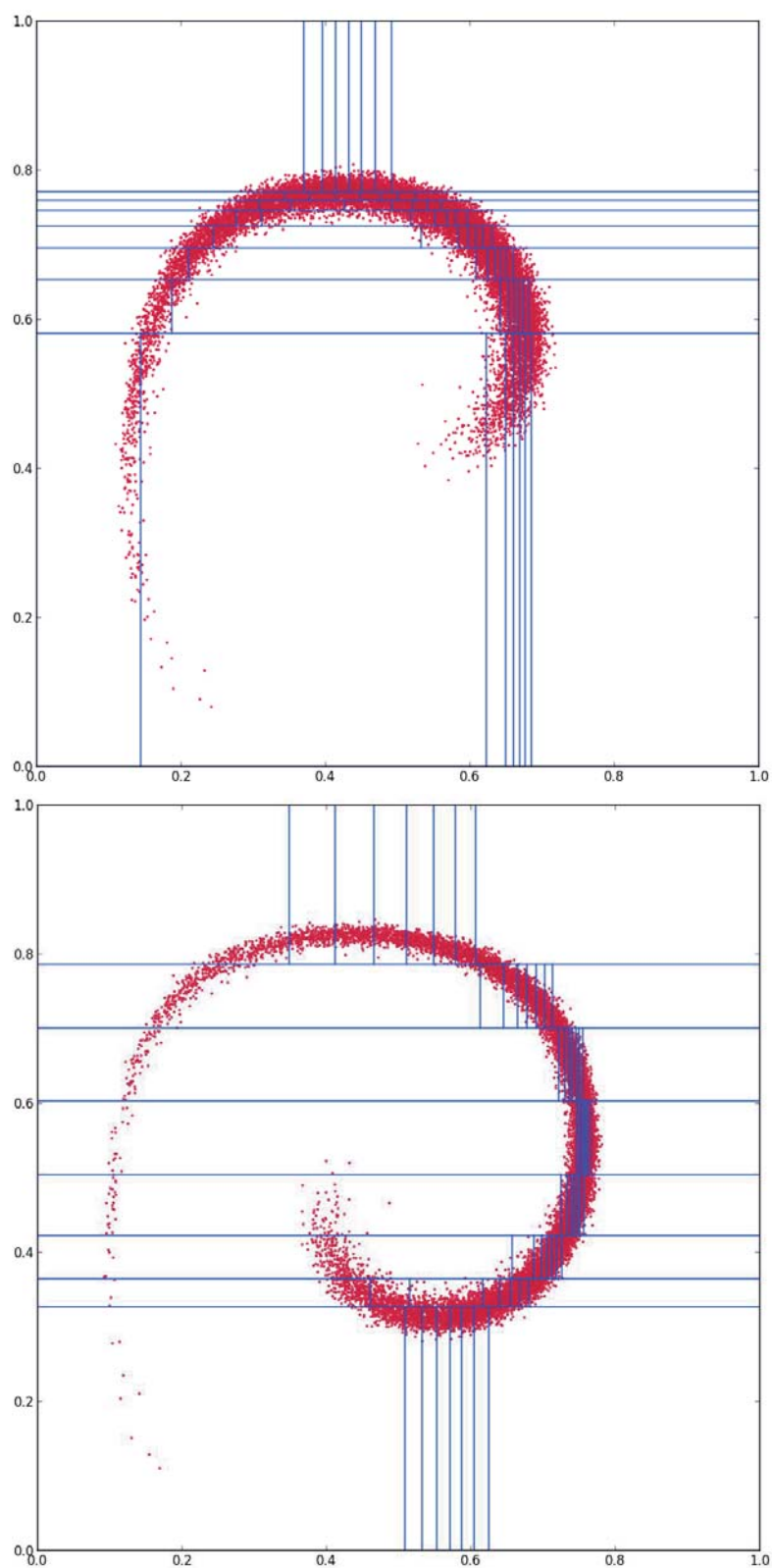


図 4.6 渦度速度場中のスライスグリッド法による動的負荷分散のスナップショット (後半)

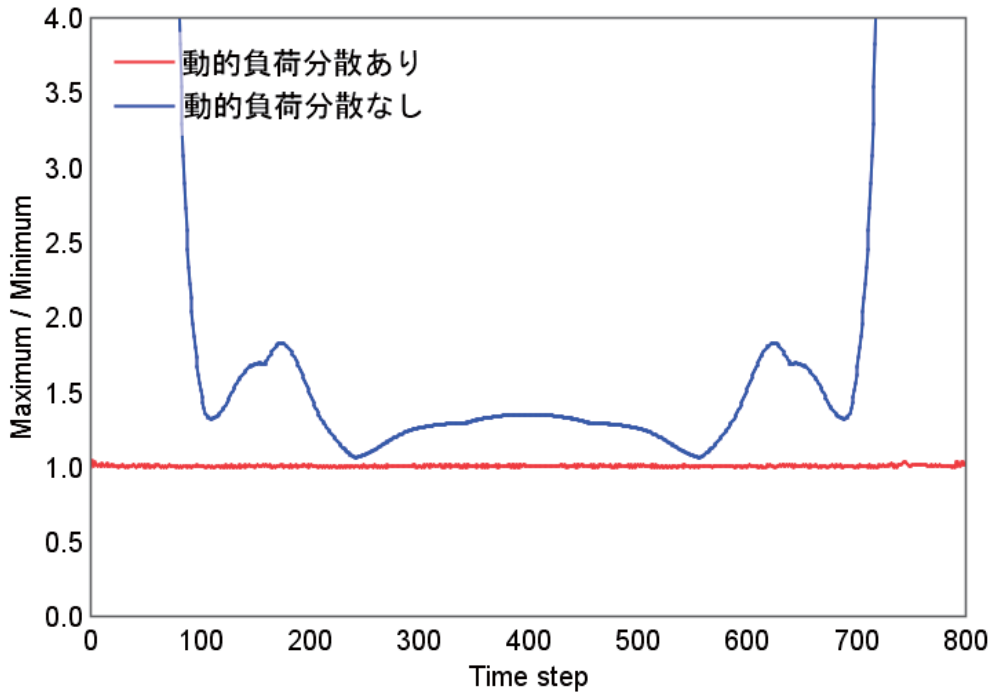


図 4.7 4 GPU を用いた場合の小領域間の最大粒子数と最小粒子数の比率の変化

図 4.7 は同一問題に対して 4 GPU を用いて計算した場合の、各時間ステップにおける小領域間の粒子数の比 (最大粒子数 / 最小粒子数) の測定結果である。動的負荷分散の実行により、赤線で示す通り GPU 間の粒子数の比はほぼ一定となっていることが分かる。

回転速度場での強スケーリング

渦速度場は、粒子分布の空間的非対称から検証には適しているが、後半に粒子分布が均一化するため、動的負荷分散の性能評価には不向きである。ここでは、以下の回転速度場を用いる。

$$u(t) = -1.0 \cos(\pi t/T) \quad (4.6)$$

$$v(t) = 1.0 \sin(\pi t/T) \quad (4.7)$$

T は周期であり、 $T = 1.0$ とする。DEM などの粒子法では、計算の安定性などのため、時間ステップは $\Delta t = 10^{-6}$ 程度の値であり、ここでも $\Delta t = 10^{-6}$ とする。総粒子数が 10^7 個と、 1.6×10^8 個の 2 通りの場合を測定する。総粒子数が 10^7 個の場合、GPU を 4 台から 128 台まで変化させる。総粒子数が 1.6×10^8 個の場合、GPU を 64 台から 512 台まで変化させる。計算領域の分割方法は、スライスグリッド法による 2 次元領域分割とする。10,000 ステップの計算時間を測定する。測定時間から 1 ステップの計算時間を算出

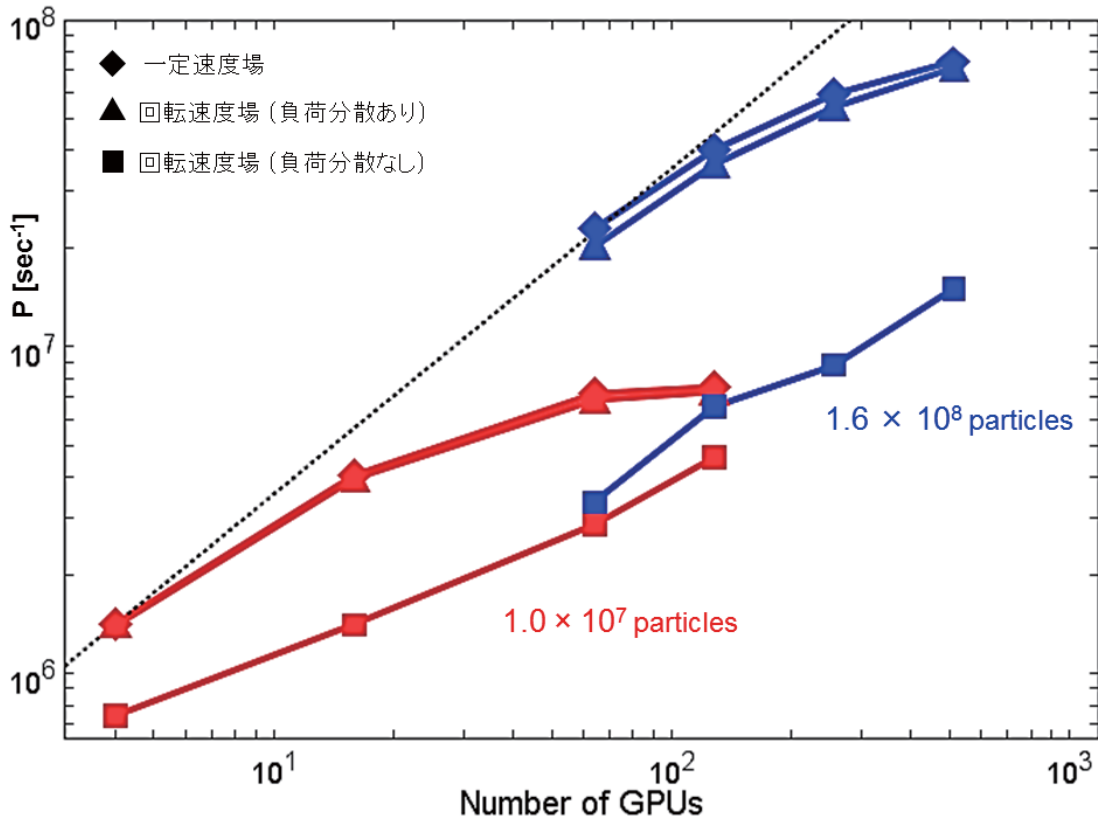


図 4.8 強スケーリングによる性能評価

し、その逆数の値に粒子数をかけたものを以下のような式で実行性能とする。

$$P = \frac{1}{T_{step}} \times N_{all}$$

1 ステップの計算時間を T_{step} 、総粒子数を N_{all} 、性能を P とした。動的負荷分散は以下に定義される均等粒子数 N_{std} からのずれ R が 20% を超えた場合に実行する。

$$R = \frac{N_i - N_{std}}{N_{std}} \times 100 \quad (4.8)$$

100 ステップ ~ 1000 ステップ間隔で再整列の頻度を変えて複数条件の性能値をそれぞれ測定する。すなわち、100 ステップに 1 回、200 ステップに 1 回...1,000 ステップに 1 回、のように 100 ステップ毎に再整列を実行する間隔を変え、それぞれの条件に対して性能値を測定し、その中で得られた最大の実行性能を測定値とする。

実行性能

強スケーリングの結果を図 4.8 に示す。図中赤色は粒子数は 10^7 個の測定結果を、青色は粒子数 1.6×10^8 個の測定結果を表す。四角が動的負荷分散を行わない場合を、三角が

表 4.1 図 4.8 の 512 GPU 使用時の計算時間の内訳

単位 : msec	時間発展	パッキング	MPI 通信	負荷分散	再整列
一定速度場	0.76	0.64	0.52	0.0	656.24
回転速度場 (負荷分散あり)	0.82	0.68	0.64	42.88	131.24
回転速度場 (負荷分散なし)	7.13	2.08	18.38	0.0	1576.89

行う場合を表す。ひし形は同じ粒子数の均一粒子分布、一定速度場の計算に対する性能で目標値を示す。黒の点線は、4 GPU での性能を GPU の台数に比例させて引いた、性能の理想直線である。 10^7 個の粒子を 4 GPU で計算させた場合の性能と、 1.6×10^8 個の粒子を 64 GPU で計算した場合の性能値は直線上の値となり、弱スケールが達成されている。図 4.8 から、動的負荷分散を行うことにより最大で 6 倍程度性能が向上しているとわかる。第 3.3.2 節の均一粒子分布で一樣速度場のもとでの計算は動的負荷分散が必要なく、この計算と比較すると、不均一粒子分布で回転速度場の場合でも、動的負荷分散を導入することにより 90 % 程度の性能を達成している。各ステップで 1 つの小領域の境界を横切る粒子の移動量は、512 GPU を用いて動的負荷分散を行う場合に全粒子数の 1~2 % 程度である。従って 10,000 ステップでの動的負荷分散の実行回数は 3 回となっている。

512 GPU を用いた場合の主要項目の計算時間の内訳を表 4.1 に示す。粒子分布の不均一により測定時間は各 GPU で異なり、全体の計算時間は最も遅い GPU の時間に律則されるため、各項目について測定された計算時間の最大値を表示している。動的負荷分散 1 回にかかる時間は、粒子の再整列の時間の 3 分の 1 程度である。実行回数を押さえたことで、動的負荷分散によるオーバーヘッドは無視できる程度である。

4.1.4 粉体・流体の複数 GPU 計算への適用

本節では 2 次元スライスグリッド法による GPU 間の動的負荷分散法を、DEM による粉体計算や改良型 SPH 法による流計算などの近接相互作用に基づく粒子法シミュレーションに適用した場合の計算手順を説明する。粒子間で相互作用計算を行わないパッシブ・スカラー粒子計算と異なり、DEM では接触する粉体の間でバネとダッシュポットに基づく物理モデルによる相互作用計算を行う。改良型 SPH 法による流体計算ではカーネル半径 (通常、初期粒子間隔の 3 倍 ~ 4 倍程度の長さ) に含まれる近傍粒子との相互作用計算を行う。各 GPU において近傍粒子探索を行うためのセル分割法の一つのセル幅だけ小領域の領域境界から内側の領域 (=DEM による粉体計算の場合は通常は粒子直径、改良型 SPH 法の場合はカーネル半径) の内部に含まれる粒子を袖 (ハロー) 領域として隣接領域にコピーして転送する必要がある。

複数 GPU を用いた DEM による粉体計算の手順を図 4.9 に示す。本研究では DEM による粉体計算では 1 ステップに 1 回だけハロー領域の通信を行う。ハロー領域の通信方法はパッシブ・スカラー粒子計算における領域外粒子の通信と同様である。GPU 上でハロー領域内に存在する粒子にフラグを立てて、これらを連続バッファにコピーしてパッキングする。それらを CPU 側に転送して、転送先別に粒子パッキングして送信する。受信した粒子をもとに今度は GPU 側にコピーしてハロー領域の GPU 間が完了する。

ハロー領域はデータの移動ではなくコピーであるため、ハロー粒子を GPU 上でコピーしてパッキングして CPU 側に転送したその後は、ハロー領域内に存在する粒子に先ほど割り振ったフラグを消してやる必要がある。一方、メモリの枯渇を防ぐため、受信側においてもハロー領域の通信で隣接する小領域から送られてきた粒子については、すべての相互作用の計算過程が完了して必要が無くなり次第、順次消去する必要がある。受信した粒子はパッシブ・スカラー粒子計算の「領域外粒子の GPU 通信」のときと同様に今現在使用しているメモリ領域の最後尾に連続的に書き込まれるので、あらかじめ隣接する小領域から受信した粒子の個数を数えておいて、相互作用計算が終わった後、使用メモリの最後尾の位置を示す変数をデータ通信を行う前の状態に戻し、ハロー粒子が使用していたメモリ領域には未参照を示す数字を格納しておくことで、バッファを元の状態に簡単に戻すことができる。

ハロー領域の通信が完了した後、相互作用計算における近傍粒子探索のためのリンクリストを構築する。ハロー粒子については相互作用計算を行う必要はないので、リンクリスト構築の際にはハロー粒子を除外するようにする。図 4.9 における相互作用計算の部分 (中央の青色部分) では、A~C の三種類の物体を含む場合を想定して示している。本研究で行った CUDA プログラミングでは中央の青色の部分が一つの CUDA のカーネル関数に

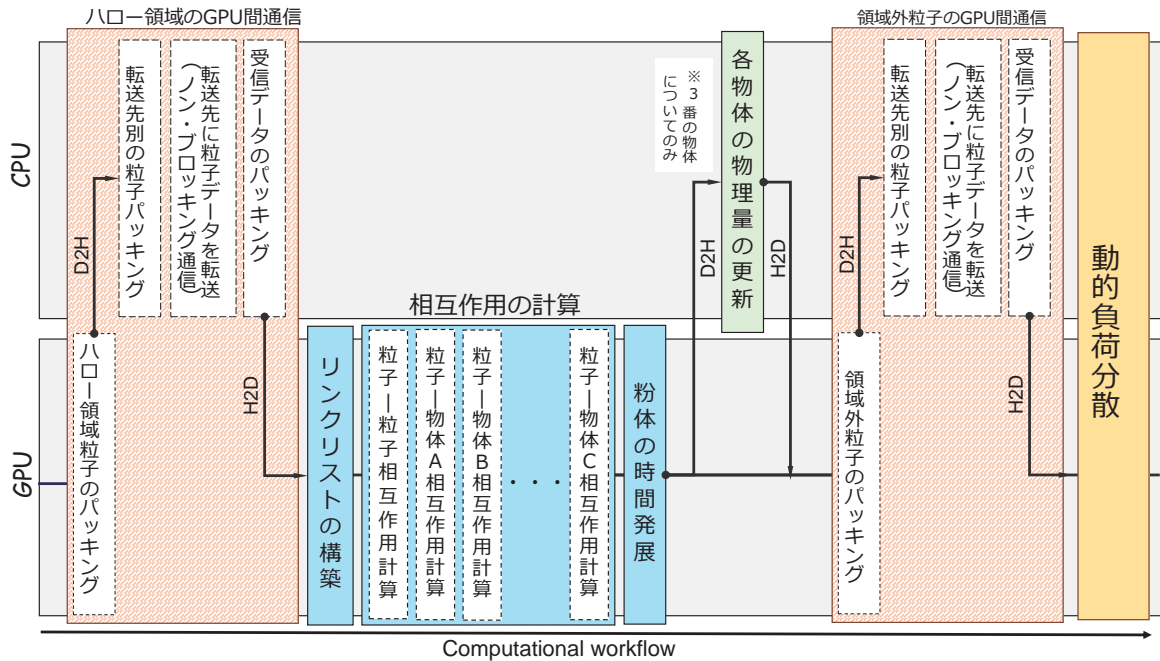


図 4.9 DEM における複数 GPU の計算手順

対応している。点線囲みの各計算項目はスレッド毎に独立に実行される CUDA のデバイス関数に相当する。まずリンクリストを辿って粒子間の相互作用計算を行い、粒子同士の接触力を求める。続いて物体 A~C との衝突計算を順次行っていく。物体の形状データは第 2.1.6 節で述べたように CAD データから生成された Level Set 値 (符号付距離関数場の値) を格納した 3 次元空間格子である (3 次元 CAD データから生成した、Level Set 関数を等値面表示した様子は図 4.11 参照)。計算開始前に予め物体 A~C のそれぞれの Level Set 関数のデータを GPU のデバイス側にコピーしておく。本研究で行う粉体計算では、Level Set 関数は簡単のためすべての GPU がそれぞれ同じデータを保持している。すなわち、図 4.9 の例で説明すると、どの GPU も A~C の Level Set 関数のデータを独立に持つようにしている。粉体との衝突判定では粒子の座標を Level Set 関数の空間格子が定義された領域の座標値に変換し、その粒子を含む格子の隣接点から線形補間して粒子座標における Level Set 値を求めることにより物体表面への距離がわかり、接触判定を行うことができる。このときの粒子と物体との接触計算の種類は 3 通りに分けられ、1) 物体が固定物体の場合、2) 物体が自動的に動く場合 (並進速度や回転速度が固定されている場合)、3) 粉体からの作用も考慮する場合のそれぞれについて処理が異なる。1) 固定物体の場合は簡単で、接触点における勾配から計算された法線ベクトルに垂直な面に対して通常の壁面に対する場合と同様に仮想粒子を配置すればよい。2) の場合は少し複雑になる。本研究で

は Level Set 関数は固定されていて、物体が移動する場合にはその並進と回転運動の操作を粒子に対して逆に操作することで物体の移動を表現している。したがって接触点の計算ではそのことを考慮する必要がある。3) の場合も移動物体であるので基本的に 2) と同じであるが、粉体からの作用を考慮するため、接触点において粒子が物体から受ける作用力を計算した際に、その反作用力を GPU のグローバルメモリ上に別途用意した配列に書き込んでおく。すべての相互作用計算が終わった後、それらを GPU 間でリダクションさせて力とトルクの総和計算を行い、物体の重心に対する並進運動と回転運動を計算する。

本研究では 3) に該当するのは第 5 章における 3 次元バンカーショット計算で使用するゴルフボールだけであり、総和計算におけるリダクションの通信コストについては流体計算の研究で詳しく議論するので、粉体計算ではこの部分のコストについては本論文では言及しない。第 5 章で示す 3 次元バンカーショット計算では、相互作用計算部分とほぼ同程度の実行時間 (力とトルクについて GPU 間でリダクションするのにかかる時間) であったことだけ述べておく。また、この後に述べる攪拌計算の強・弱スケージングでは攪拌ギヤが物体の計算方法の 2) に該当するが、本研究では Level Set 関数のグローバル情報を各 GPU で独立に保持していて、速度の更新や回転操作もそれぞれの GPU で (同じ処理を) 行っているため、GPU 間で通信が必要になることはないため、物体 (攪拌ギヤ) が自動的に回転することによる並列化効率への影響はない。3) において力とトルクの総和計算を行った際には、それらはマスターとなる領域に集められ、CPU 上でクォータニオンや角運動量などの物理量を更新する。更新された物体 (本研究の場合、ゴルフボール) の物理量は、その後 CPU 側に伝えられる。図 4.9 における緑色の部分で CPU-GPU 間通信が発生しているのは、粒子データではなく 1 個の物体の物理量についてであり、そのオーバーヘッドは無視できるほどに小さい。一方、粒子については、相互作用計算を完了した後、2 段 2 次精度ルンゲクッタ法により時間積分を行い、領域外粒子について GPU 間で通信する。動的負荷分散は 1 ステップの計算項目がすべて完了した後に定期的に行う (ここで「動的負荷分散」には第 4.1.1 節と ~ 第 4.1.2 節で説明した計算過程のすべてが含まれている。図 4.9 では負荷分散後に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変更されたことで行う GPU 間の通信や、粒子の再整列も含めて表示している)。

改良型 SPH 法の複数 GPU 計算の手順を図 4.10 に示す。ここで、図 4.10 では説明の都合上、第 3.2.6 節で示した改良型 SPH 法による流体計算の具体的な計算項目のうち、速度の修正子の計算と座標の修正子の計算をまとめて表示している。ハロー領域の GPU 間の通信は各計算ステップで 3 回実行される。図 4.10 では省略しているが、改良型 SPH 法計算でも図 4.10 中の予測子、粒子粘性、圧力補間、修正子の各計算の直前に Linked-list の構築を DEM 計算と同様に行っている。通常、ハロー領域の通信コストは DEM でも改良型 SPH 法でも粒子移動の GPU 間と比べて高く、特に流体計算では領域境界からカーネル半径の長さに含まれる粒子をすべて隣接小領域に送信するため、改良型 SPH 法による 3

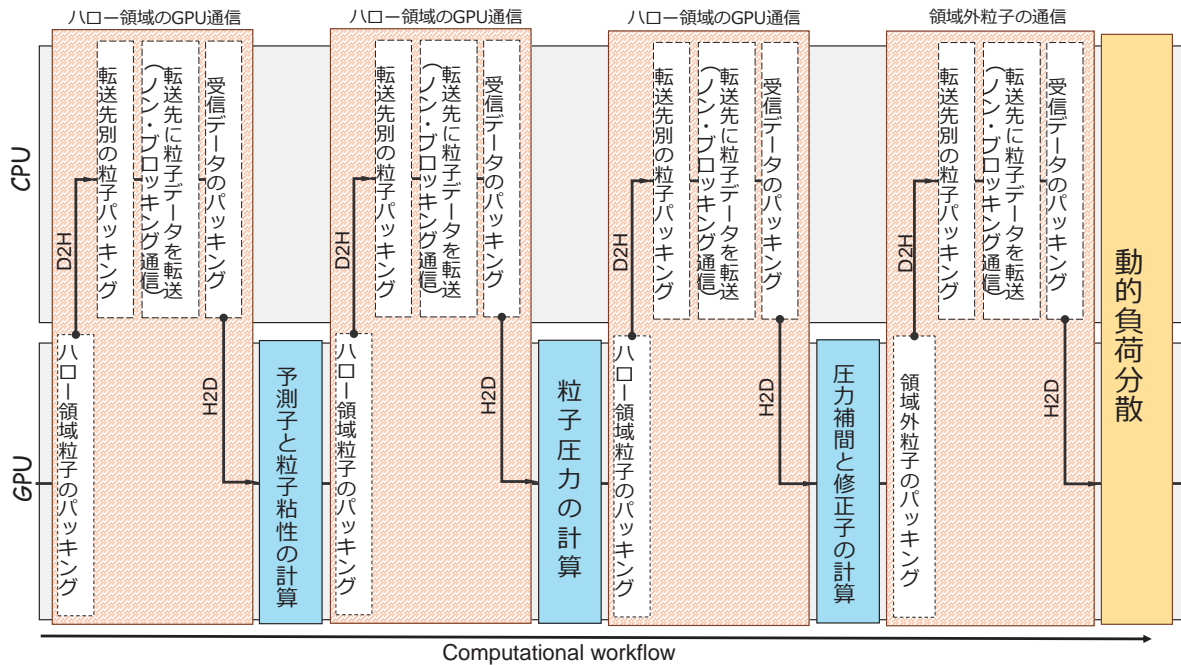


図 4.10 改良型 SPH 法における複数 GPU の計算手順

次元流体計算の場合では GPU 間の通信に要する全時間に対して「ハロー領域の GPU 間通信」が大半(多くの場合, 95% 以上)になる. 粒子移動による GPU 間通信のコストは相対的に無視でき, ハロー領域の通信コストが複数 GPU 計算の並列化効率を決定する大きな要因になる.

複数 GPU を用いた DEM 計算に対してスライスグリッド法を適用した場合の動的負荷分散の様子が分かる計算例として, 64 台の GPU を使い 100 万個の粒子を使った粉体のダム崩壊シミュレーションの計算結果を図 4.12 に示す(この計算の可視化には文献 [68] を用いている). 各小領域の領域番号を奇数番号と偶数番号に分けて赤色と緑色に塗り分けて表示している.

工業的, 産業的に重要ないくつかの典型的な粉体现象に対して, スライスグリッド法による動的負荷分散を用いた複数 GPU シミュレーションを行った. 任意形状の物体との相互作用を取り扱いは第 2 章の第 2.1.6 節で説明した符号付距離関数 (Level Set 関数) 法を用いている. また, シミュレーション結果の可視化には POV-Ray[69] を用いた光源追跡法 (レイ・トレーシング法) を用いている. 以下, それぞれのシミュレーションの詳細を述べる(なお, 改良型 SPH 法への適用例については第 5.3 節における「複雑形状を含むダム崩壊シミュレーション」で確認できるためここでは省略する).

螺旋すべり台シミュレーション

計算領域を $1.06 \text{ m} \times 0.61 \text{ m} \times 0.76 \text{ m}$ とし、そこに流路となる螺旋状のすべり台を配置する。全部で 416 万個の粒径が 1 mm の粒子をすべり台の上部に落下させて初期値を生成した後、時間刻み幅を 2.0×10^{-5} 秒として物理時間で 4.6 秒に相当する 230,000 ステップを 32 台の GPU を用いて計算した螺旋すべり台シミュレーションの結果を図 4.13 に示す。すべり台の流出口付近で粒子が堆積する様子が分かり、せん断方向の相互作用計算による摩擦の効果を確認することができる。

搬送計算

計算領域を $0.98 \text{ m} \times 0.28 \text{ m} \times 0.98 \text{ m}$ とし、そこに粉体を搬送するスクリューを設置する。スクリューは図 4.11 に示す様に CAD データから生成した Level Set 関数 (符号付距離関数) を用いて表現する。全部で 433 万個の粒径が 0.5 mm の粒子をスクリューの左端の部分に落下させて初期値を生成した後、時間刻み幅を 1.0×10^{-5} 秒として物理時間で 1.64 秒に相当する 164,000 ステップを 64 台の GPU を用いて計算した搬送シミュレーションの結果を図 4.14 に示す。回転するスクリューを含む場合にも、壁面との間に目詰まりを起こさずに搬送することができ、出口付近では粉体が堆積する様子を確認できる。

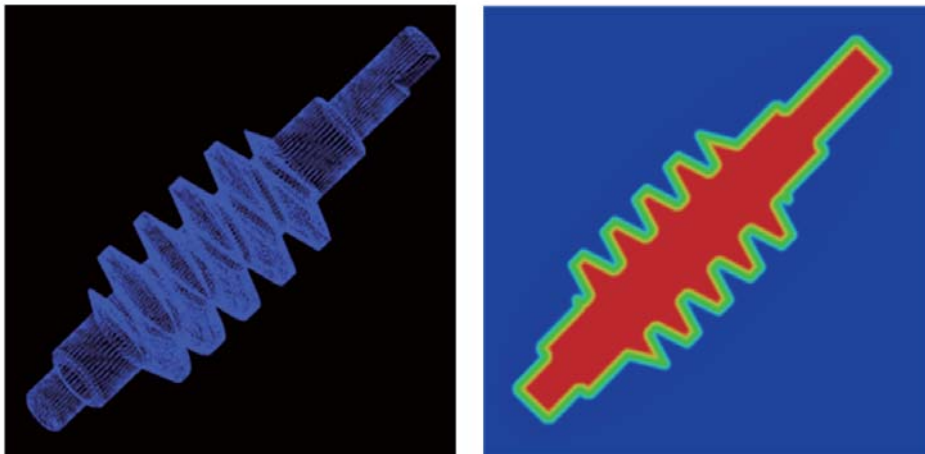


図 4.11 CAD データ (左) と CAD データから生成した符号付距離関数

攪拌計算

計算領域を $11.48 \text{ m} \times 11.48 \text{ m} \times 11.48 \text{ m}$ とし、そこに粉体を攪拌するギヤを設置する。全部で 412 万個の粒径が 70 mm の粒子をギヤの中央の上空から落下させて初期値を生成

した後、時間刻み幅を 4.0×10^{-6} 秒として物理時間で 5.2 秒に相当する 130,000 ステップを 64 台の GPU を用いて計算した攪拌シミュレーションの結果を図 4.15 に示す。搬送シミュレーションよりも鋭角な搬送ギヤを導入し、底面に対して垂直に回転させる実用的な粉体シミュレーションを行うことができ、図 4.15 からは 2 種類の粉体が混ざり合う様子が確認できる。

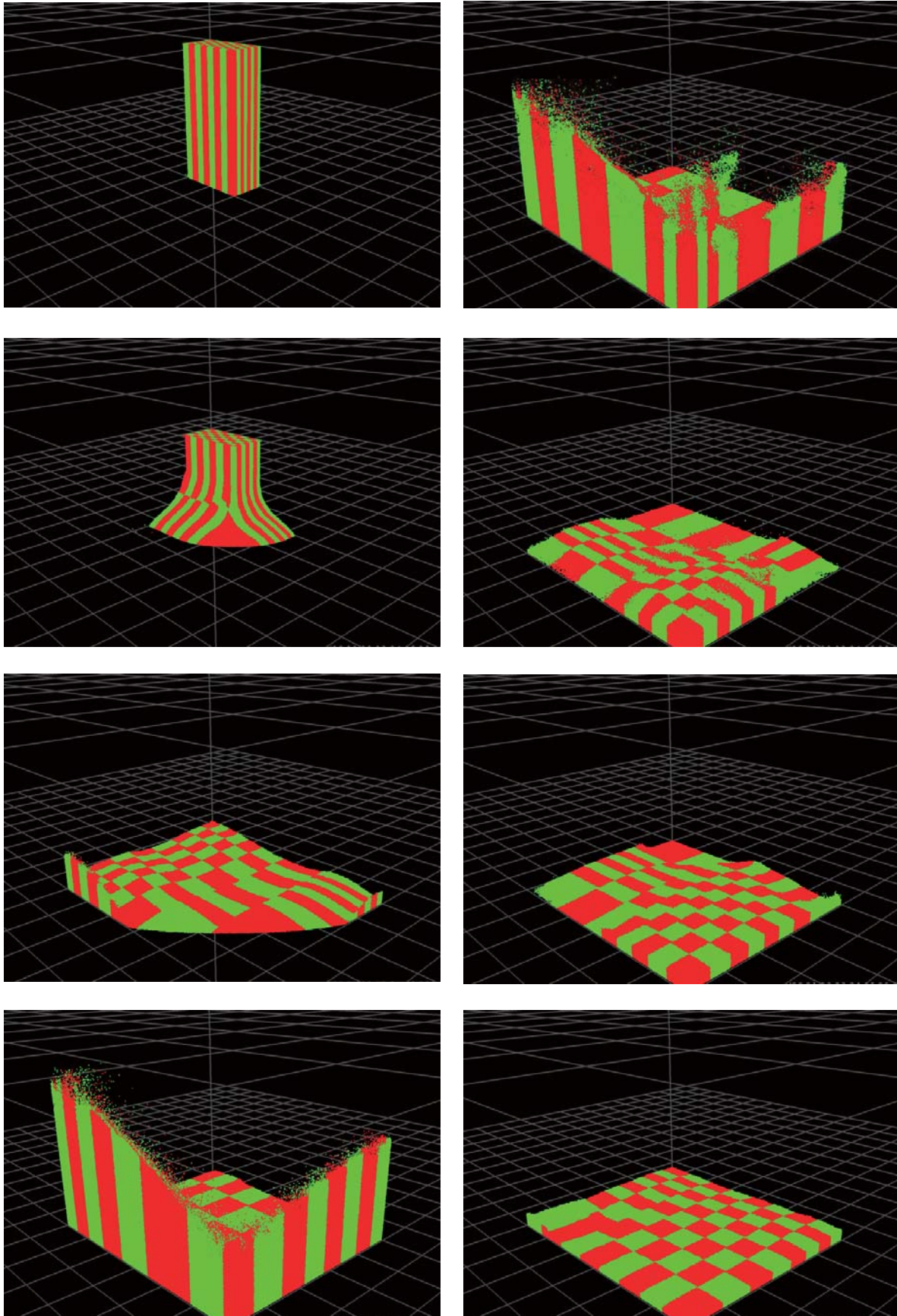


図 4.12 64 GPU を用いた粉体のダム崩壊シミュレーションにスライスグリッド法を適用した様子

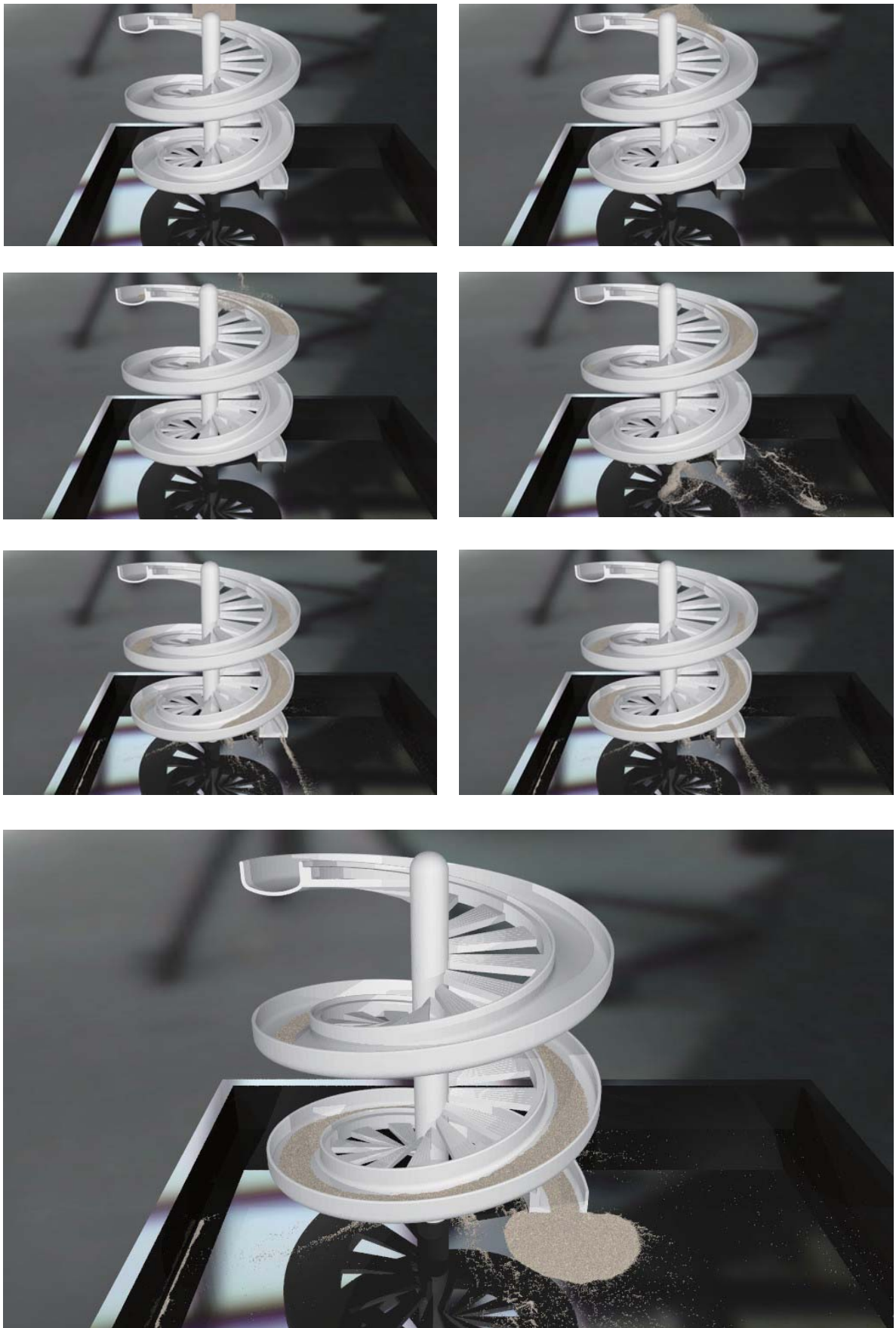


図 4.13 32 台の GPU を使い 416 万個の粒子を使った螺旋すべり台シミュレーション

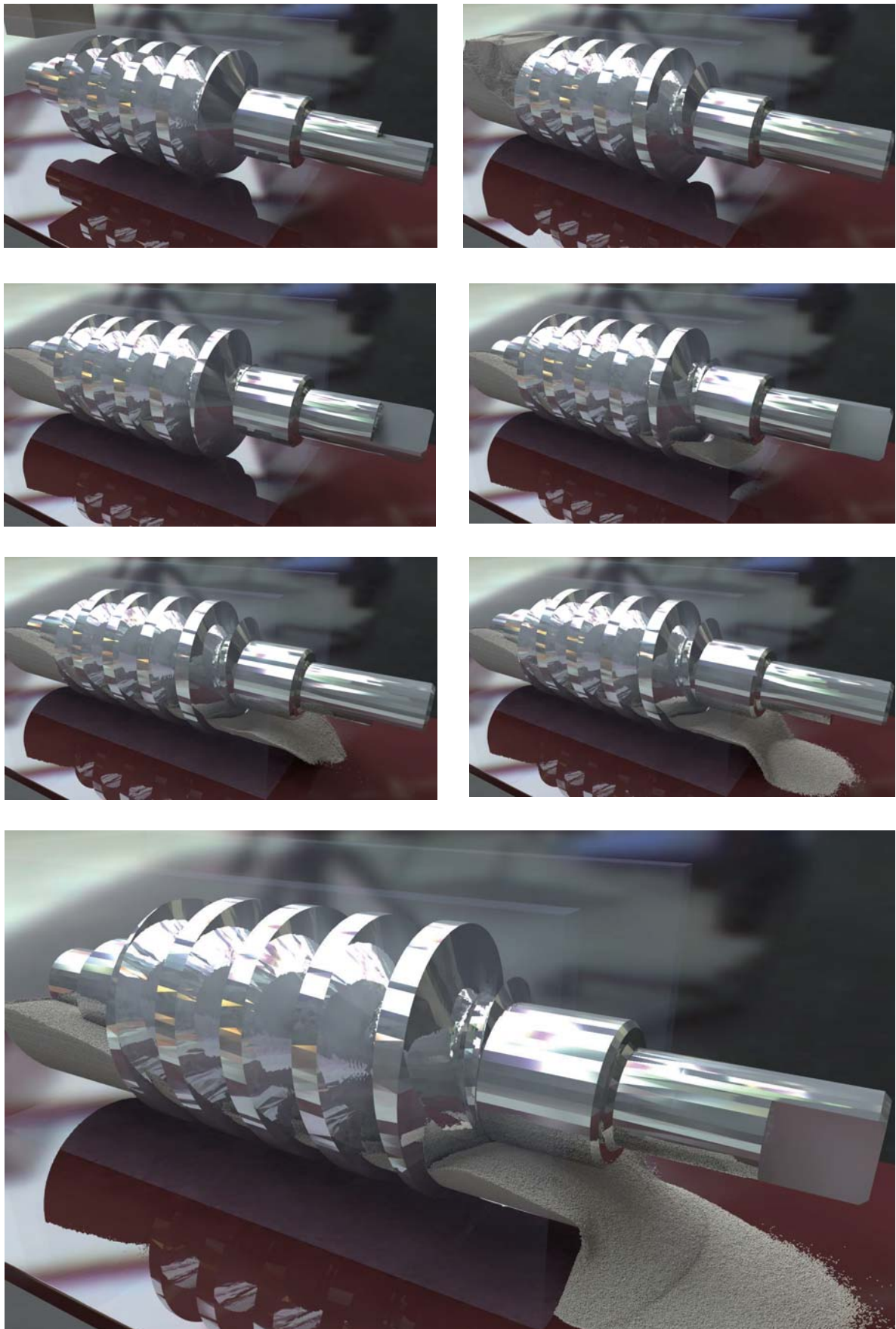


図 4.14 64 台の GPU を使い 433 万個の粒子を使った搬送計算

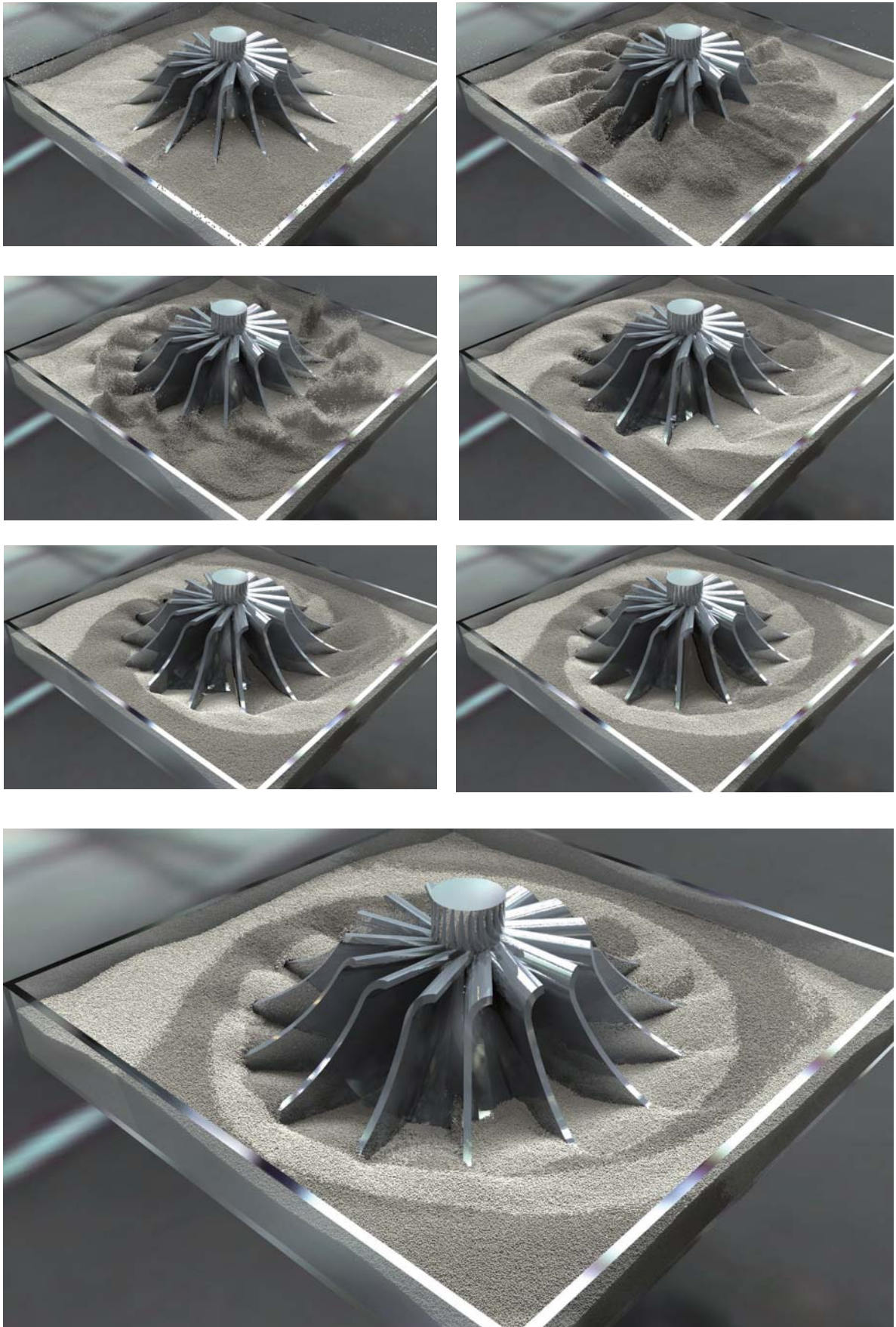


図 4.15 64 台の GPU を使い 412 万個の粒子を使った攪拌計算

4.1.5 DEM を用いた粉体計算の強・弱スケーリング

スライスグリッド法による動的負荷分散を導入した DEM の複数 GPU コードを用いて、TSUBAME 2.5 の GPU (K20X) で大規模粉体シミュレーションの実行性能を強・弱スケーリングで検証した。攪拌ギヤが粉体中で回転する図 4.15 の攪拌計算に対して、200 万個、1,600 万個、1 億 2,900 万個の粒子により計算する場合の実行性能を測定した。縦軸の実行性能は、1 ステップあたりの実行時間の逆数に粒子数を掛けて定義された性能の相対的な指標値である。測定結果を図 4.16 に示す。

まず、弱スケーリングについて説明する。弱スケーリングでは各プロセッサ(ここでは、GPU)に割り当てる問題サイズを固定したまま、台数に比例させて全体の問題サイズを大きくする。GPU の台数の増加に比例して粒子数も増加し、それに合わせて全体の計算領域も大きくなる。計算領域が均等に分割されていて、かつ粒子が各領域に均一に分布する理想的な場合には、問題サイズが大きくなっても隣接する GPU 間で行われる通信や計算内容が変わらないため 1 ステップあたりの計算時間は一定に保たれる。本問題では図 4.16 の四角印で表された測定値の始点(4 GPU, 200 万粒子)、三角印で表された測定値の

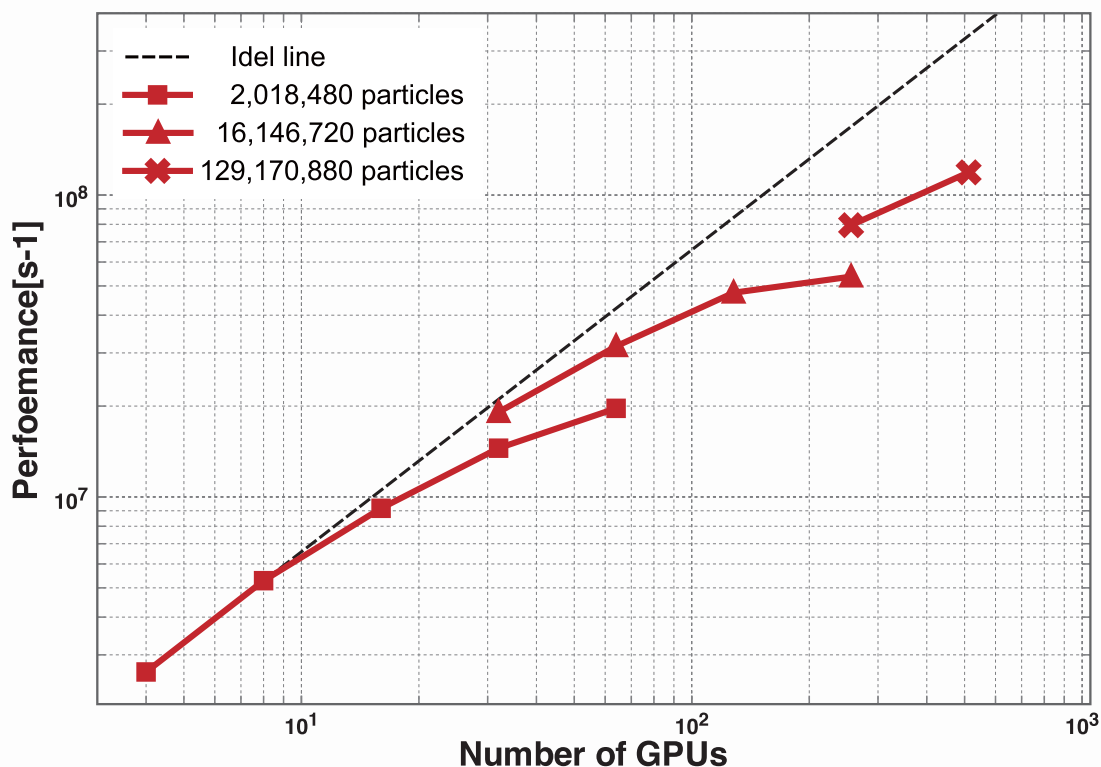


図 4.16 DEM による粉体の強・弱スケーリング

表 4.2 図 4.16 の 512 GPU 使用時の計算時間の内訳

	Linked-list の構築	相互作用計算	粒子移動の通信	ハロー領域の通信
計算時間 [msec]	17.6	182.6	26.3	694.6
割合 [%]	1.9	19.8	2.8	75.3

始点 (32 GPU, 1600 万粒子), X 印で表された測定値の始点 (256 GPU, 1 億 2,900 万粒子), の 3 点が弱スケーリングの測定結果である. 動的負荷分散により GPU の 1 台あたりの粒子数は 50 万粒子に保たれているため, 理想的にはこれらの 3 つの測定点の計算では実行時間は等しくなり, 図 4.16 においては 4 GPU の性能値を台数に比例させた黒色の点線に重なるはずである. しかし, 図 4.16 から分かるように理想的な直線 (点線) と比較して大きく性能低下していることが分かる.

次に強スケーリングについて説明する. 強スケーリングでは全体の問題サイズを固定したまま, 使用するプロセッサ (ここでは, GPU) の台数を増加させる. 本問題の場合, 弱スケーリングを実施した上述の 3 つの測定点に対し, 同一計算条件のもと GPU の台数だけを増加させてそれぞれ強スケーリングを実施し, 性能値の変化を調べた. 図 4.16 における四角印, 三角印, X 印の 3 つの折れ線がそれぞれの場合の強スケーリングの測定結果である. 強スケーリングにおいても性能の測定値が GPU の台数に比例すれば, 点線上に重なる直線を描く. しかし, 図 4.16 では四角印, 三角印, X 印のいずれも実行性能が低下して折れ線になっている. 四角印で表された 200 万粒子に対する測定結果から, 8~16 倍までは GPU 数の増加に応じて性能向上が期待できるが, それ以上は性能が飽和してることが確認できる.

図 4.16 の X 印で表された 2 つの測定点, それぞれ 1 億 2,900 万粒子を用いた場合の 256 GPU, 512 GPU に対する実行性能の測定結果である. 512 GPU を用いた場合の計算時間の内訳を表 4.2 に示す. 上段が 1 ステップあたりの計算時間を, 下段が 1 ステップあたりの全体の計算時間に占める各項目の割合をパーセント (%) で示している. ハロー領域の通信が全体の計算時間の 75% 以上となっており, これが原因となり性能が飽和していることが確認できる. また, 第 4.1.4 節で述べたように全体の通信時間 (粒子移動の通信と, ハロー領域の通信の和) のうち 96% をハロー領域の通信が占めていることが分かる. スライスグリッド法の欠点である分割された小領域の形状 (アスペクト比) の悪化による領域間通信量の増大が原因となり, DEM の強・弱スケーリングでは並列化効率が低下することが確認できる. また, これらの結果から DEM 計算へのスライスグリッド法の適用限界が 256 GPU~512 GPU 程度であると言える.

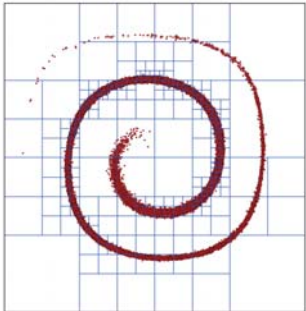
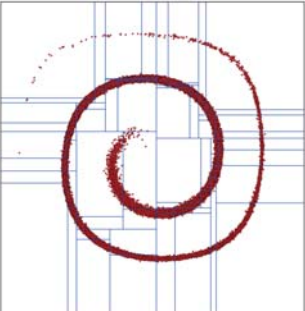
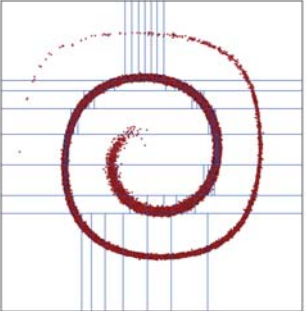
空間分割法	計算領域を再帰的に分割		領域境界線を移動
	四分木(八分木)	KD木	スライスグリッド法
粒子計算に適用した様子			
特徴	<p>特長と利点: 四分木や八分木を用いて計算領域を再帰的に分割する。通常は、モートン曲線などを使うことで領域分割に利用できる。任意の領域分割数が可能になる。3次元領域分割への拡張が容易である。再分割の局所性が高く分割の計算コストが低い。隣接ノードとの接続関係が明確で安定。</p> <p>問題点:</p> <ul style="list-style-type: none"> 複雑な領域形状になり、データ通信量の増加への影響が懸念 リフ・マイグレーションの負荷が不明 	<p>特長と利点: 水平方向に粒子数が半分になる位置で分割したあと、サブ領域で垂直方向に粒子数が半分になる位置で分割する。以降これを再帰的に行う。3次元分割が可能になる。</p> <p>問題点:</p> <ul style="list-style-type: none"> 領域分割数が2のべき乗に制限され、実用的な問題に適さない 領域同士の接続関係が複雑化し、実質的にランダム・データ通信になる 分割領域が高アスペクト比(ノード間データ通信量の増加) 	<p>特長と利点: 水平と垂直方向に順番に領域境界線を移動させる。アルゴリズムが簡単で理解しやすい。</p> <p>問題点:</p> <ul style="list-style-type: none"> 2次元領域分割までが限界(分割手続きが複雑) 粒子分布の偏りにより領域同士の接続関係の変動が激しく、著しく悪化する場合がある 分割領域が高アスペクト比(ノード間データ通信量の増加)

図 4.17 代表的な領域分割法の比較

4.2 空間充填曲線を用いた動的負荷分散

スライスグリッド法を用いた粉体の複数 GPU 計算では領域形状が高アスペクト比になり領域間通信量が増加して並列化効率が著しく低下することが第 4.1.5 節で議論され、その適用限界が 256 GPU~512 GPU 程度であることが分かった。本節では木(ツリー)を用いた再帰的な領域分割を粉体や流体の複数 GPU 計算に対して行う方法を提案する。

4.2.1 木(ツリー)を用いた格子細分化による領域分割法の利点

計算領域を再帰的に分割する方法は重力多体計算などで良く使われる KD 木 [41] や ORB(Orthogonal Recursive Bi-section)[42] などのように、計算領域の各方向に対して交互に二分法 (Bi-section 法) を適用して再帰的に分割する方法と、四分木 (三次元空間では八分木) などを用いて計算領域を等分空間 (四分空間や八分空間) に再帰的に分割する方法に分けられる。四分木と KD 木、及び比較のためスライスグリッド法による領域分割の特徴を図 4.17 にそれぞれ示す。

スライスグリッド法では上述のような領域の縦横が高アスペクト比になる問題のほか

に、分割手続きの複雑さから三次元領域分割の実現が困難であること、領域境界線を移動させるため粒子分布が偏ると領域接続数が著しく悪化すること、などの問題がある。一方、KD 木を用いる方法では水平方向と垂直方向(三次元空間では奥行き方向も加わる)に交互に二分割を繰り返しながら領域の再帰分割を行う。KD 木を用いると三次元領域分割が可能になるが、領域の分割数が2のべき乗に制限されるため実用性が低い。また、領域接続数が増加してデータ通信が複雑になる問題がある。さらに、図 4.17 からわかるように分割領域が高アスペクト比になる問題が解決されないため、粉体や流体の複数 GPU 計算においてスライスグリッド法と比較して大幅な並列化効率の向上は期待できない。

ORB を用いる場合も KD 木と同様の手順で再帰分割を行うが、ORB では水平方向に領域境界線の位置を揃えて分割するため領域間の接続関係は KD 木よりも改善される。しかし、ORB による分割は結局スライスグリッド法と同様であるためこの場合も高アスペクト比となることによる性能低下の問題の解決は期待できない。これらに対し、四分木(三次元空間では八分木)などを用いて計算領域を等分空間に再帰的に分割する方法では、各方向に均等な格子(二次元領域分割では正方形、三次元領域分割では立方体状の格子)で細分化するため、KD 木やスライスグリッド法に見られるような縦横の高アスペクト比の領域は生じにくくリーフ同士の接続も良い。これにより粉体や粒子法の流体計算のように粒子が頻繁に領域境界線を横切るような計算に対しても KD 木やスライスグリッド法を用いた場合に問題となる領域形状が高アスペクト比となる問題を回避して複数 GPU 計算における並列化効率の大幅な向上が期待できる。ただし、四分木や八分木の構築だけでは粒子分布に対して格子細分化を行っただけであり、そのままでは領域分割に利用できない。通常、空間充填曲線(多くはモートン(Z-order 曲線))によってツリーを辿り、曲線の通過したリーフ内の粒子数の和が均等になるようにリーフを連結させて領域分割を完了する。そのため領域形状はスライスグリッド法や KD 木のように矩形にはならず複雑形状になる。分割された個々の領域の表面積の悪化は懸念されるものの、領域形状を矩形とした場合に発生する縦横比が極端に高アスペクト比となる領域が生じる問題を回避できる利点があるため、後者による性能低下が著しい粉体や流体の複数 GPU 計算では実行性能の向上が期待される。本研究では、空間充填曲線を用いた動的領域分割を流体の複数 GPU 計算に適用した場合の並列化効率についてスライスグリッド法と比較する。

粉体の複数 GPU 計算にスライスグリッド法を用いる場合に領域の縦横比が高アスペクト比となることで並列化効率が悪化することは本研究で得られた新しい知見のひとつであり、この問題は改良型 SPH 法による流体計算ではさらに深刻になると予想される。第 4.1.5 節の「DEM を用いた粉体計算の強・弱スケーリング」では領域境界線を横切る粒子についての GPU 間通信のみならずハロー領域の通信のコストも増加し、スライスグリッド法の適用限界となる 256 GPU~512 GPU で通信時間の大部分をハロー領域の GPU 間通信が占めることが示された。DEM による粉体計算では領域境界線から粒子直径だけ内側

の領域に含まれる粒子を隣接する小領域にコピーするだけでよいが、改良型 SPH 法による流体計算ではカーネル半径 (=粒子直径の 3~4 倍の長さ) 分だけ内側の領域に含まれる粒子を隣接領域にコピーするためハロー領域のデータ通信量が大幅に増加する。ハロー領域は各小領域の表層領域であり領域形状がその通信性能に大きく影響するため、高アスペクト比な小領域を発生させないことが並列化効率を維持する上で粉体計算の場合よりも一層重要になる。

改良型 SPH 法の複数 GPU 計算ではこのように空間充填曲線を用いた格子細分化に基づく再帰的な領域分割法を用いることに分割領域が高アスペクト比の問題を回避できる点で大きな利点がある。このほか、四分木から八分木などの三次元空間に対応した分割木に変更し、空間充填曲線を 3 次元曲線に切り替えるだけで 3 次元領域分割へと容易に拡張できること、粒子分布の変更のある箇所だけその部分を担当するリーフ単位で負荷分散を行うことができるため動的負荷分散の局所性に優れていること、それにより領域再分割処理のコストが低いこと、リーフ同士の接続関係が明確であり、通信と計算のオーバーラップ手法の導入など高速化手法への対応が簡単であることなどの利点がある。以上、改良型 SPH 法による流体計算に格子細分化に基づく再帰的な領域分割法を用いる利点を述べたが、勿論これは通常の SPH 法に対しても同様である。一方、この手法のデメリットとしては領域再分割によって生じる大量の粒子の移動 (=以降、リーフ・マイグレーション) などのコストが不明であることなどが挙げられ、本論文ではこれらについても検討する。

4.2.2 空間充填曲線を用いた動的負荷分散法の概要

空間充填曲線を用いた格子細分化に基づく再帰的な動的負荷分散法の概要を図 4.18 を用いて説明する。図 4.18 では簡単のため 2 次元正方領域の計算領域に粒子番号が $a \sim r$ の 18 個の粒子が分布している場合を想定している。これらを一つの領域が 3 個の粒子を含むよう全部で 6 個の小領域に分ける場合を考える。はじめにツリーを用いてリーフ内の粒子数が所定の粒子数以下になるまで再帰的に分割する。粒子計算に対して格子細分化を行う場合、一枚のリーフに含まれる粒子数が 1 個か 0 個になるまで細分化を続けるのが理想的である。しかし、本研究のように 1 億個に及ぶような粒子を扱う場合、一つ一つの粒子の全てにセルを対応させて生成すると容易にメモリが枯渇してしまう。そこで、10 個~50 個などある程度の粒子数を「所定の粒子数」に設定して、その閾(しきい)値に達するまで格子細分化を行う方法を用いる。「所定の粒子数」と全体の負荷分散の精度の関係や、所定の粒子数のメモリ使用量の違いについては、第 4.3.2 節で検討する。図 4.18 では例として所定の粒子数を 2 個とした場合の分割の様子を示しており、各リーフには 0~2 個の粒子が含まれるよう分割されていることが分かる。

図 4.18 右側には生成された四分木の構造図を示している。四分木の末端にリーフ同士

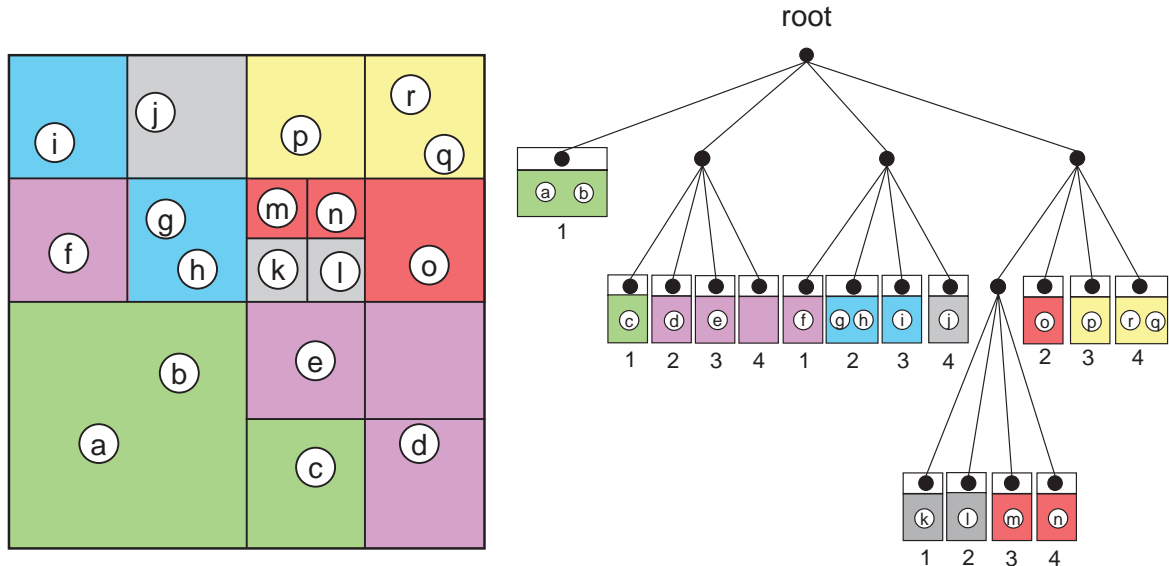


図 4.18 木 (ツリー) を用いた格子細分化による動的負荷分散法の概要

を粒子数が等しくなるように束ねて連結していき、リーフ群の占める領域と小領域を対応させることで領域分割が実現できる。図 4.18 では右側の四分木と同一の色の部分が左側の各小領域に対応している。リーフをどのように連結するかによって領域形状が変わるため、再帰分割の際にどのようにリーフに順番付けを行うかは非常に重要である。図 4.18 は簡単な例としてモートン曲線 (Z-order) 曲線の例を示しており、木構造の各階層においてリーフの下部に示す順序で端から番号を振ることにより 1 筆書きで末端のリーフをたどることができる。モートン曲線は理解しやすいが、図 4.18 の左側からわかるように同一の色 (同一の領域) が二つに分化してしまうことがしばしば生じるため、粒子計算ではそれぞれの領域片がそれぞれ隣接領域と通信を行うことで通信コストの増加が懸念される。そのため、いくつかの異なる種類の空間充填曲線の利用を検討する必要がある。

4.2.3 空間充填曲線の数学的背景

空間充填曲線の発見は区間 $[0, 1]^2$ から正方形 $[0, 1]^2$ への写像についての議論に端を発している [70][71]。すなわち、「区間と平面の間の写像において全単射写像を考えたとき、このような写像は連続となり得るか」という議論が 1870 年代の後半から始まり、これが空間充填曲線の発見につながった。全単射写像とは写像において「像」と「元」の間に 1

対1対応の関係 (one-to-one correspondence) が成立することである。元はここでは区間であり、像はここでは平面である。数学者 E. ネット (1879) によってそのような写像は必ず不連続となることが証明された。すなわち、「区間と平面の間の写像において全単射写像となる曲線は存在しない」ことが証明された。これに続く命題として、全単射性の条件がない場合、全射性だけを要求するのではあればどうかということが議論になった。つまり、「区間と平面の間の写像を考えたとき、全射性だけを要求したときにはそのような写像は連続となり得るか」という問いについての議論が始まった。全射性だけを要求するとは像における重複を許すことであるから、この問題は言い換えれば、2次元平面の正方形内に曲線を描くことを考えるような場合に、「始点と終点を任意に固定し、一筆書きで領域内を埋め尽くすことができる曲線は、領域内での重複点を許した場合には存在するか」という問題と同じである。このような曲線は G. ペアノ (1890) により初めて発見されその存在が確認された。その後、同様の性質をもつ曲線が D. ヒルベルト (1892) をはじめ他の数学者達によっても発見され、現在ではこれらの曲線は「空間充填曲線」と呼ばれている。当初は区間 $[0, 1]^2$ から正方形 $[0, 1]^2$ への写像を考えたが、その後の集合論や位相幾何学分野における研究により、 n 次元ユークリッド空間に適用できることが示された。すなわち、我々のいる3次元物理空間でも、幾何曲線として空間充填曲線が存在することが確認された。

空間充填曲線は「始点と終点を任意に固定し、一筆書きで重複点を許した場合に領域内を埋め尽くすことができる曲線」であるから、交差や重複する場合もあり種類により異なる。このような性質を持つ空間充填曲線を始めて発見したのは上述のように G. ペアノだが、それらの幾何的生成を初めて行ったのは D. ヒルベルトである。D. ヒルベルトは、元である区間を「部分区間」に分割し、像である正方形を「部分正方形」に分割したときの元と像における対応を考え、「一つの正方形が一つの区間に対応するとき、その部分正方形がその区間の部分区間に対応する。また、隣接する部分区間は共通辺を持った隣接する部分正方形に対応する」ことを示した。ヒルベルトによる厳密な定理は参考文献 [70][71] で詳しく示されているが、そこから示される本研究における重要なことは、

- N 回の繰り返しで区間を長さ $1/2^{2n}$ の 2^{2n} 個の部分区間に分割したとき、もとの区間において $|t_1 - t_2| < 1/2^{2n}$ を満たすような t_1 と t_2 による区間 $[t_1, t_2]$ は、せいぜい連続する部分区間で重なり合い、その像は辺の長さが $1/2^{2n}$ の連続する2つの正方形になる

である。つまり、正方形をどこまで部分正方形に分割していても、その部分空間内を充填できる曲線があり、また曲線が複数の部分区間を跨がる場合は対応する部分正方形同士は隣接する(連続になる)ことを意味している。木(ツリー)を用いた格子細分化による領域分割法では、ツリーのリーフをヒルベルト曲線の定義における部分平面に対応させるこ

とで一筆書きですべてのリーフを辿ることができるようになる。

正方形を仮定し四分木を例に説明してきたが、D. ヒルベルトの定義は任意平面 (空間) に対して定義されており、九分木を用いるペアノ曲線の場合も同様である。また、3次元空間に対しても正方形の代わりに立方体を考えるだけで同じである。ヒルベルト曲線やペアノ曲線では部分正方形 (立方体) において始点と終点を設け、連続する正方形 (立方体) 同士でそれらが接続されるようになっている [70][71]。そのため、空間分割において同じリーフを曲線が2回通過するようなことは起こらず、また曲線同士も交わらない。なお、モートン曲線では図 4.19 などの作図の上では同一リーフを2回を通過するが、2回リーフを辿っているわけではなく、また計算の上では部分正方形 (部分立方体) は番号で管理しているので作図の上で2回目に通過するときセル内の粒子数を数える訳ではない。モートン曲線同士が交わることは自体はあまり問題にならないが、その際に曲線がジャンプすることは領域分割においては問題になる。

以上、空間充填曲線の数学的な背景について述べた。本節の記述はすでに引用しているように参考文献 [70] 及び和訳である [71] を参考に作成した。また数学的な厳密性よりも内容を伝えることに主眼をおいて記述している。

4.2.4 空間充填曲線の特徴と辿り (たどり) 順序

本研究では木 (ツリー) 構造を用いた格子細分化による領域分割においてヒルベルト曲線、モートン曲線、ペアノ曲線の3種類の空間充填曲線を用いる。各曲線を用いる場合の空間における各方向への分割方式、使用する木 (ツリー) の種類、それらの子リーフの辿り方の種類を表 4.3 にまとめた。表 4.3 から分かるように、ヒルベルト曲線とモートン曲線では各方向に 1:2 分割を行い、ツリーは 4 分木 (3次元領域分割では 8 分木) を利用する。一方、ペアノ曲線では各方向に 1:3 分割を行い、8 分木 (3次元領域分割では 27 分木) を利用する。木 (ツリー) におけるそれぞれのリーフの辿り方についても表 4.3 の示す通りになっている (子リーフへの辿り方については後に詳述する)。

表 4.3 それぞれの空間充填曲線の分割方式と使用する木 (ツリー)、子リーフの辿り方の種類

	ヒルベルト曲線	モートン曲線	ペアノ曲線
分割方式	1:2 分割	1:2 分割	1:3 分割
木 (ツリー) の種類	4 分木 (8 分木)	4 分木 (8 分木)	9 分木 (27 分木)
リーフのたどり方	4 通り (12 通り)	1 通り (1 通り)	4 通り (4 通り)

(括弧内は 3 次元曲線の場合)

それぞれの空間充填曲線の辿り方の特徴について図 4.19 を用いて説明する。モートン

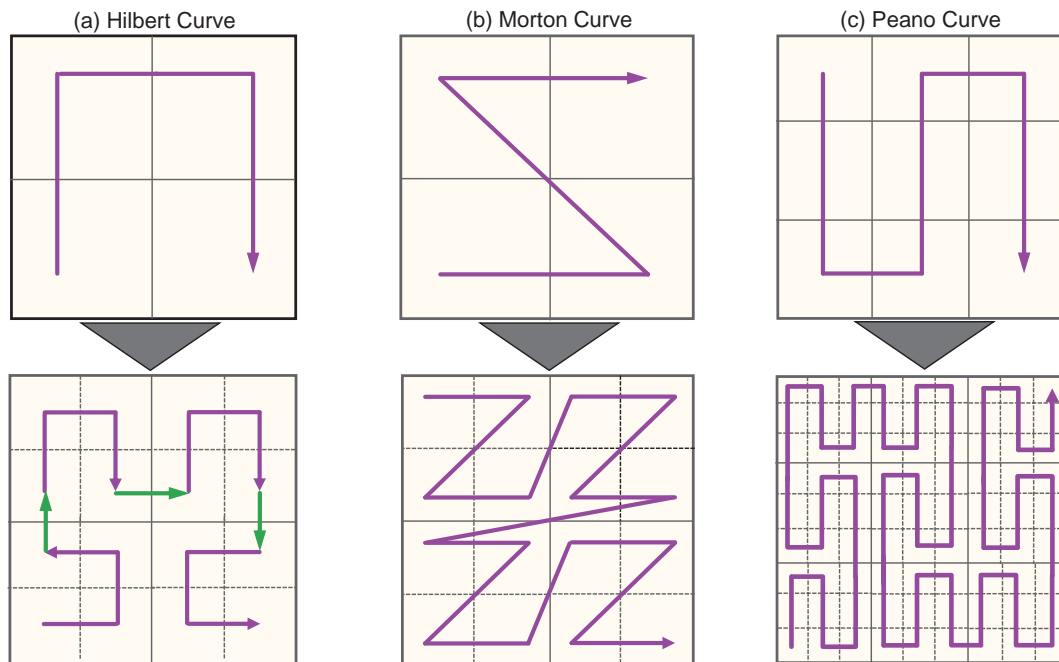


図 4.19 ヒルベルト曲線，モートン曲線，ペアノ曲線の 3 種類の空間充填曲線における特徴的な通り順序

曲線は図 4.19 で示すように Z 字を描くような通り方が特徴である。モートン曲線を領域分割に用いる場合，曲線が空間的にジャンプすることでひとつの領域が計算機のメモリ上でも空間的にもしばしば離れた場所に分かれるため，隣接との接続数が増えることによる通信性能の低下が心配される。これに対して，ヒルベルト曲線では曲線は必ず隣接するリーフを辿るのでともとリーフ同士の連結がよい。領域分割に用いた場合も，それらリーフを束ねた小領域同士の隣接関係も良くなることが期待できる。一方，ペアノ曲線はヒルベルト曲線やモートン曲線とは異なり，1:3 分割を行う。ペアノ曲線もヒルベルト曲線と同様，隣接するリーフを必ず辿るためヒルベルト曲線と同じく小領域同士の良好な接続関係が期待でき，さらにペアノ曲線では 1:3 分割を行い 9 分木 (27 分木) を用いるため，格子細分化の効率が良く粒子分布の変化の激しい粉体や流体計算では，ヒルベルト曲線やモートン曲線と比べて局所性の高い領域分割が期待できる。

木 (ツリー) の子リーフの通り方について 2 次元ペアノ曲線を例により詳しく説明する。表 4.3 で示したように，2 次元ペアノ曲線では 9 分木を用い，それらの子リーフの通り方は 4 通りである。文献 [43] に倣いここではこれら 4 通りの通り方を P, Q, R, S とする。その具体的な順序を図 4.20 に示す。再帰的な格子細分化においては図 4.20 に示す曲線を辿るように分割された子空間 (子リーフ) に対して番号付けを行う (後からその番号順に子リーフを辿ることで空間充填曲線を再現できるからである)。個々の子空間に対して指定

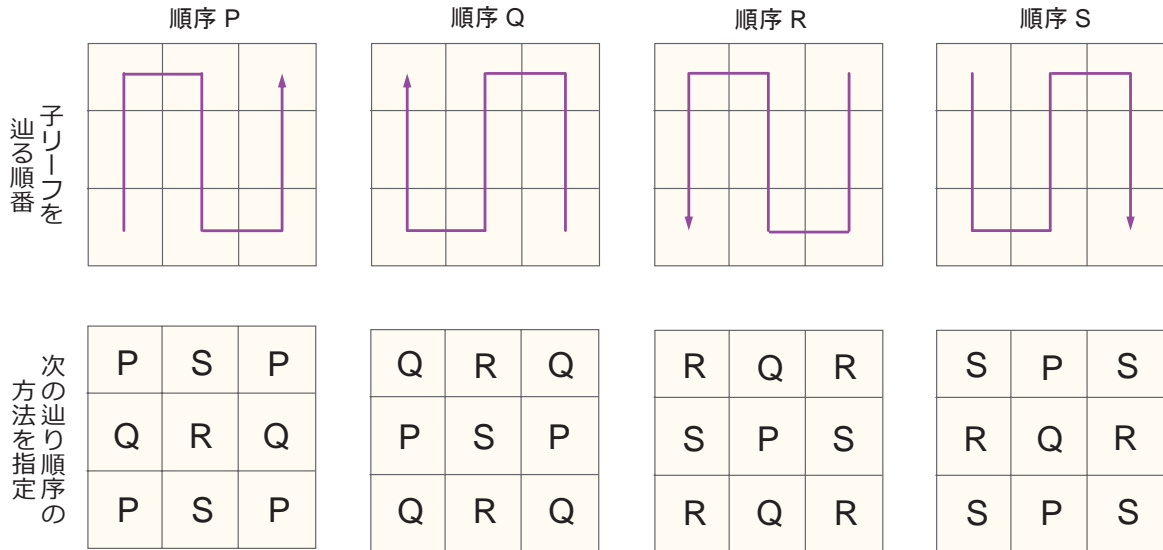


図 4.20 2次元ペーノ曲線を用いた場合の9分木に対する4通りの辿り方と、子リーフに対する次の順序の指定方法

する次の辿り方は文献 [43] にあるように図 4.20 の下段のようになっている。このような対応が4種類の辿り方のそれぞれに対して決まっている(後に示す様に、第 4.2.5 節「領域再分割を用いた動的負荷分散アルゴリズム」で示す Algorithm. 2 中の *pattern* がここでの P, Q, R, S のことであり、子リーフに割り振られる *s_pattern* は、図 4.20 中の下段の対応表により与えられる)。

次の第 4.2.5 節「領域再分割を用いた動的負荷分散アルゴリズム」での理解が進むように、この部分の実装における要所についても述べておく。上述のような空間充填曲線の辿り方を指定するためには、具体的には配列の番号(インデックス)を空間インデックスに対応させる。例えば図 4.20 における辿り順序の P の場合には、9分木における9個の子空間を指す番号:

$$\text{配列番号の列} : \{0, 1, 2, 3, 4, 5, 6, 7, 8\} \quad (4.9)$$

に対して、

$$\text{空間番号の列} : \{0, 3, 6, 7, 4, 1, 2, 5, 8\} \quad (4.10)$$

を対応させ、子空間に対する次の辿り順序の指定についても、

$$\text{辿り順序の列} : \{P, Q, P, S, R, S, P, Q, P\} \quad (4.11)$$

を対応させるようにする。実装では予めこれらの変換表を作っておき、格子細分化における再帰処理ではこの変換表をもとに子リーフへの順番付けと次の辿り方の指定を行う。以上、2次元ペアノ曲線を例にして格子細分化における子空間の順序を指定する方法について述べたが、2次元ヒルベルト曲線、2次元モートン曲線についても同様である。2次元空間充填曲線を用いた場合のそれぞれの変換表について C/C++ によるコード (定数の一覧表) を次に示しておく (2次元モートン曲線の場合は辿り順序は1通りだが同様に示す)。

3次元空間曲線の場合や図 4.20 の下段に示した次の順序を指定する規則に対する数学的な裏付けについては文献 [43] を参照されたい。第 4.2.2 節と本節により空間充填曲線を利用した木 (ツリー) による格子細分化を用いた動的負荷分散法の概要を説明した。続く第 4.2.5 節では具体的なパラメータを示しながら、領域再分割により粒子数を均一化して動的負荷分散を行う方法についてより詳しく説明する。

プログラム 4.1 2次元ペアノ曲線で使用する変換表の C/C++ によるコード (定数の一覧表)

```

1 enum PEANO_RULES {P, Q, R, S}; // 列挙型で指定
2 const int PEANO_P[9] = {0, 3, 6, 7, 4, 1, 2, 5, 8}; // 順序 P
3 const int PEANO_Q[9] = {2, 5, 8, 7, 4, 1, 0, 3, 6}; // 順序 Q
4 const int PEANO_R[9] = {8, 5, 2, 1, 4, 7, 6, 3, 0}; // 順序 R
5 const int PEANO_S[9] = {6, 3, 0, 1, 4, 7, 8, 5, 2}; // 順序 S
6 const int PEANO_PATTERN_P[9] = {P, Q, P, S, R, S, P, Q, P}; // P に対応する次の順序の指定
7 const int PEANO_PATTERN_Q[9] = {Q, P, Q, R, S, R, Q, P, Q}; // Q に対応する次の順序の指定
8 const int PEANO_PATTERN_R[9] = {R, S, R, Q, P, Q, R, S, R}; // R に対応する次の順序の指定
9 const int PEANO_PATTERN_S[9] = {S, R, S, P, Q, P, S, R, S}; // S に対応する次の順序の指定

```

プログラム 4.2 2次元モートン曲線で使用する変換表の C/C++ によるコード (定数の一覧表)

```

1 const int MORTON_ORDER [4] = {0, 1, 2, 3};

```

プログラム 4.3 2次元ヒルベルト曲線で使用する変換表の C/C++ によるコード (定数の一覧表)

```

1 enum HILBERT_RULES {H, A, B, C}; // 列挙型で指定
2 const int HILBERT_H[4] = {0, 2, 3, 1}; // 順序 H
3 const int HILBERT_A[4] = {0, 1, 3, 2}; // 順序 A
4 const int HILBERT_B[4] = {3, 2, 0, 1}; // 順序 B
5 const int HILBERT_C[4] = {3, 1, 0, 2}; // 順序 C
6 const int HILBERT_PATTERN_H[4] = {A, H, H, B}; // H に対応する次の順序の指定
7 const int HILBERT_PATTERN_A[4] = {H, A, A, C}; // A に対応する次の順序の指定
8 const int HILBERT_PATTERN_B[4] = {C, B, B, H}; // B に対応する次の順序の指定
9 const int HILBERT_PATTERN_C[4] = {B, C, C, A}; // C に対応する次の順序の指定

```

4.2.5 領域再分割を用いた動的負荷分散アルゴリズム

計算領域を木 (ツリー) (2次元空間充填曲線なら4分木または9分木、3次元空間充填曲線なら8分木または27分木) を用いて再帰的に分割し、全てのリーフ領域内の粒子数が所定の粒子数になるまで細分化を続ける。計算領域内のリーフを空間充填曲線でたどり、通過したリーフ内の粒子数の積算により領域を分割することができる。木 (ツリー) を用いて再帰的分割を行う関数の擬似コードを Algorithm 2 に示す。

Algorithm 2 Recursive Tree Construction

```

1: function GENTREE(min, max, dep, pattern)
2:   Count  $N_p \in [min, max]$ 
3:   if  $N_p < N_{min}$  or  $dep > dep_{max}$  then
4:     Leaf.end=true
5:     for  $k=0,1\dots N_{child}$  do
6:       Leaf.child[ $k$ ]=NULL
7:     end for
8:   else
9:     for  $k=0,1\dots N_{child}$  do
10:      Set [s_min, s_max], s_pattern by pattern
11:      Leaf.child[ $k$ ]=GENTREE(s_min, s_max, dep+1, s_pattern)
12:    end for
13:  end if
14:  return Leaf
15: end function

```

Algorithm 2 中の `genTree` において第一と第二引数の min と max はリーフ領域の最小と最大の座標を示すベクトル型の変数であり、第三引数の $depth$ が 0 の時に計算領域になる。第四引数の $pattern$ はツリーにおける子空間 (8 分木なら 8, 27 分木なら 27 の子空間) をたどる順序を指定する番号である。genTree でははじめにリーフ領域内に存在する粒子数 N_p を数え、これが所定の粒子数 N_{min} よりも少ない場合は再帰分割を終了する。リーフ領域内の粒子数 N_p が N_{min} よりも多い場合には新たにリーフを生成して分割を続ける。11 行目で親から指定された $pattern$ に対応する空間充填曲線の順序通りに N_{child} 個のそれぞれのリーフに子空間を割り当てることで、最終的に構築されたツリーを空間充填曲線によって辿ることができる。生成されたリーフの連結により領域分割を行う擬似コードを Algorithm 3 に示す。空間充填曲線を辿りながら各リーフ領域内の粒子数 N_p を全体の粒子数を計測する N_{sum} と各小領域内の粒子数を計測する N_{sub} に積算していき、均等粒子数である N_{std} に達するまでリーフを連結する。連結されたことを表すために領域番号を表す $index$ を各リーフに割り振る。 N_{std} を N_{sum} と N_{sub} が超えると連結をやめ、 $index$ をひとつ繰り上げ、 $N_{sub} = 0$ として再びリーフの連結を開始する。連結されたリーフ群をそれぞれ新たな小領域として領域分割が完了する。

2 次元の粒子分布に対し、Algorithm 2 と Algorithm 3 に従い 64 の領域に分割し、12 種類の色で塗り分けた結果を図 4.21~ 図 4.23 に例として示す。それぞれの曲線を用いてリーフを辿る様子を上段に、リーフを連結した際に得られた領域分割の様子を下段に示し

Algorithm 3 Connection of Leafs and Decomposition

```

1: function CONNECTLEAFS(Leaf,  $N_{sum}$ ,  $N_{sub}$ , domID)
2:   if Leaf.end=false then
3:     for  $k=0,1\dots N_{child}$  do
4:       CONNECTLEAFS(Leaf,  $N_{sum}$ ,  $N_{sub}$ , domID)
5:     end for
6:   else
7:     if  $N_{sub} \geq N_{std}$  then
8:        $N_{sub} = 0$ 
9:        $domID += 1$ 
10:    end if
11:    Count  $N_p \in Leaf$ 
12:     $N_{sum} += N_p$ 
13:     $N_{sub} += N_p$ 
14:    Leaf.domain_id = domID
15:  end if
16: end function

```

ている。ペアノ曲線では 1:3 分割であるためヒルベルト曲線やモートン曲線と比較して領域の局所性が高い。モートン曲線も局所性は良いが、中心付近でひとつの小領域が離れた位置に 2 つに分かるなど、各曲線の特徴が確認できる。空間充填曲線を用いた領域分割では、リーフ内の上限の粒子数 N_{min} をいくつに設定するかにより全体の負荷分散の精度 (各小領域における均等粒子数からのずれ幅) R_{load} を制御することができる (第 4.3.2 節参照)。

$$R_{load} = \frac{N_{sub} - N_{std}}{N_{std}} \times 100 \quad (4.12)$$

なお、本論文では木 (ツリー) 領域再分割の際には Algorithm 2 と Algorithm 3 に従い毎回ツリーを再構築し、空間充填曲線を用いて全てのリーフを一筆書きで辿り直している。本来、粒子数が増加し高負荷になった部分だけ曲線を一度切断しその部分だけツリーのリーフを細分化して正常な部分とつなぎ直すことや、粒子数が少なくなった部分のリーフを解放してつなぎ直す「リーフの解放と生成」を行う方が、ツリーの再構築にかかるコストを回避することができるため望ましいといえる。しかし、その場合は均等粒子数に対してどの程度の粒子数の増減があった場合にリーフの解放や生成を行うかというリーフの解放と生成の条件を設定する必要がある、負荷分散の精度を左右する変数が多くなり複雑になる。本論文ではツリーの再帰分割において所定の粒子数の N_{min} を導入しており、まずは

これと全体の負荷分散の関係を調べる必要がある (第 4.3.2 節参照). 加えて, 粒子法計算では近傍粒子探索をはじめ全体の計算コストが高いため, 動的負荷分散においてツリーを毎回再構築しても計算コストは実用上は問題になりにくいことなどから, 本論文では毎回ツリーを再構築し, その都度全てのリーフを空間充填曲線によって辿るようにしている.

4.2.6 複数 GPU 実装

Algorithm 2 のツリーの構築では, 再帰処理の中で各リーフ領域内の粒子数を計測する. 粒子配列に対してリーフ内の粒子を探索すると探索時間が $\mathcal{O}(N)$ (N :全粒子数) に比例するためコストが高い. そこで計算領域内に空間格子を生成し, 各格子に格子内の粒子数を登録した粒子数テーブルを用いることで探索時間を削減する.

複数 GPU を用いた粒子計算では, 粒子データは図 4.24 の 1 番のように各小領域の GPU のデバイス・メモリ上に分散している. そのため, 図 4.24 の 2 番のようにそれぞれの小領域で粒子数テーブル断片を作成し, ホスト (CPU) を介してマスターを担当する小領域に集める. この際, 通信データ量を抑えるために図 4.24 において粒子が存在するセル (黄色の部分) のみを GPU 上でパッキングして通信する. マスターを担当する小領域では, 集めた断片から図 4.24 のように計算領域全体の粒子数テーブルを作成し, Algorithm 2 に従い領域分割を行う. 十分な解像度で領域分割が行われるように空間格子のセルの大きさをあらかじめ設定するツリーの最大深さで決まる最小リーフ単位と等しい大きさにする. さらに, 粒子数リストと同じ大きさの領域番号を格納するための空間格子を用意し, Algorithm 3 により各リーフに対して小領域の番号が割り振りながら, 同時に空間格子において各リーフがしめる部分の格子に領域番号を格納して図 4.24 の 4 番のような領域番号テーブルを作成する.

複数 GPU を用いた粒子計算では, 時間発展により粒子が領域外に移動する. GPU のデバイス・メモリに置かれた領域分割テーブルを参照して移動先の判定を行う. 本論文ではツリーの構築はホスト (CPU) 側で行う方法をとっている. そのため領域番号テーブルもホスト側に確保されている. 領域分割完了後, あらかじめ領域番号テーブルをデバイス側にコピーして用意する. この際, CPU-GPU 間の通信量を抑えるため, 領域分割前のテーブルと更新後の領域分割テーブルと比較して更新されたリーフ領域部分に該当するセルのみを GPU 側に転送する.

DEM や SPH などの近接相互作用に基づく粒子法の複数 GPU 計算では, 小領域間の接続情報として領域境界の裾 (ハロー) 領域を隣接小領域にコピーするために, 隣接小領域の接続関係を調べる必要がある. 各小領域において自身の領域番号を領域番号テーブル上で探索し, 図 4.25 のように該当する格子点の隣接 8 点 (3 次元では 26 点) のセルに格納された番号と比較する. 自身の領域番号が一致しなかった場合, 当該セルは異なる小領域と隣

接していると判定できる.

粒子数と領域番号のテーブルはそれぞれツリーの最大深さで決まる大きさのセルを持っている. ツリーの深さを d , 計算領域の 1 方向の長さを L としたとき, 1 セルの 1 辺の長さを L_{cell} とすると以下の関係にある.

$$L_{cell} = \frac{L}{S^d} \begin{cases} S = 3 & (\text{Peano}) \\ S = 2 & (\text{Hilbert, Morton}) \end{cases} \quad (4.13)$$

本研究ではハロー領域として領域境界から 1 セル分の L_{cell} 領域に含まれる粒子を GPU 上で集めてホスト側に転送し, 取得した接続関係に従って隣接小領域に MPI 通信によりコピーするようになっている. そのため, 領域が正しく接続されるようにハロー領域の 1 方向の長さを R_{halo} とした場合に $R_{halo} \leq L_{cell}$ の条件が課される. すなわち,

$$C_{halo} := \frac{L_{cell}}{R_{halo}} = \frac{L}{S^d \times R_{halo}} \geq 1 \quad (4.14)$$

である. R_{halo} は各小領域内で近傍粒子探索を行う際の空間格子のセル幅に等しく, SPH 法では通常, カーネル半径であり, DEM では粒子直径が用いられる.

各曲線を用いた場合の空間格子のセル数は式 (4.13) の分母に等しい. 本領域分割の複数 GPU 実装では, 粒子数, 領域番号, 再分割前の領域番号テーブルという 3 つの整数型の空間格子を利用するため, 設定可能なツリーの最大の深さがアーキテクチャのメモリ容量の制約を受ける. 図 4.27 は各 3 次元空間充填曲線において, ツリーの最大深さとテーブル生成に必要なメモリ容量の総量の関係を示しており, 粒子配列なども考慮すると, 本研究で使用するメモリ容量が 6 GB の GPU (NVIDIA Tesla K20X) 上では, 3 次元ヒルベルト曲線と 3 次元モートン曲線の場合に 9 階層, 3 次元ペアノ曲線の場合に 6 階層程度が上限だと確認できる.

本研究では木 (ツリー) により生成されるリーフは, 2 次元空間充填曲線による 2 次元領域分割では正方形になり, 3 次元空間充填曲線を用いた 3 次元領域分割では立方体になっている. これに対して粒子数テーブルや領域番号テーブルは 2 次元領域分割でも 3 次元領域分割でも最小リーフと同じ大きさのセル幅である等間隔の 3 次元空間格子を用いるようにしている. 3 次元の計算領域に対して 3 次元空間充填曲線による 3 次元領域分割を行う場合は, 図 4.26 の (a) のように計算領域の直方体を抱合できる最小の立方体の領域を考え, その領域に対して等間隔の 3 次元空間格子を生成するようにしている. 一方, 3 次元の計算領域に対して 2 次元空間充填曲線による 2 次元領域分割を適用する場合は少し複雑になる. 直方体の計算領域に対して領域分割を行う方向から見た計算領域を抱合できる最小の正方形の領域に対して最小リーフと等しいセル幅で格子を生成する (図 4.26 の (b) における水平と奥行き方向). 領域分割を行わない方向については最小リーフと同じセル幅の格子をその方向に敷き詰めて覆うだけの格子数を用意する (図 4.26 の (b) における鉛

直方向). 簡単のためこれまでは L を計算領域の一片の長さとして説明したが, 実際の計算領域 (l_x, l_y, l_z) とは, 図 4.26 の対応関係にある. 従って, 図 4.26 において計算領域 (l_x, l_y, l_z) は任意に設定でき, L の長さはセル幅 $\times 2$ のべき乗 (又は 3 のべき乗) となる.

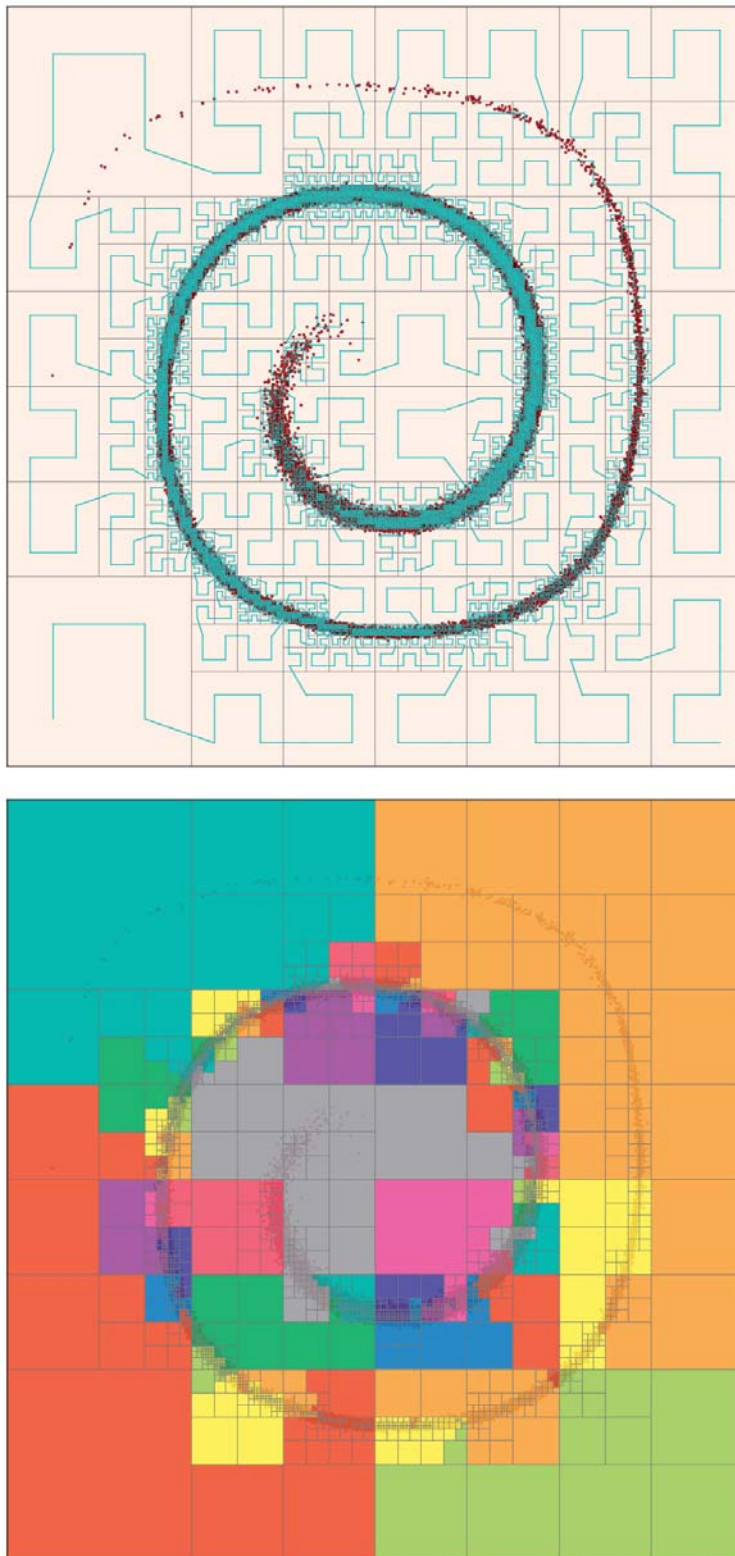


図 4.21 ヒルベルト曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割された小領域を 12 種類の色で塗り分けた様子 (下段)

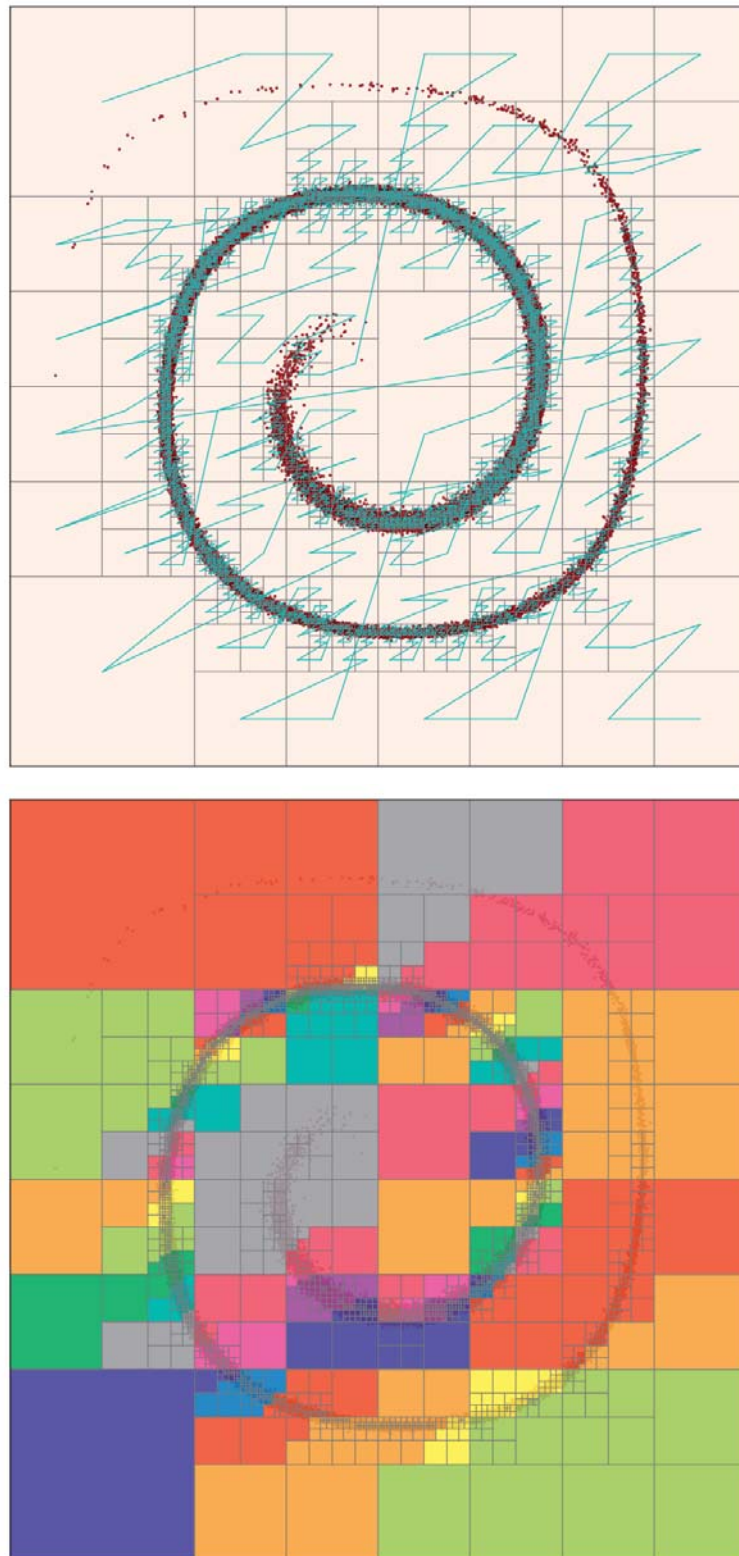


図 4.22 モートン曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割された小領域を 12 種類の色で塗り分けた様子 (下段)

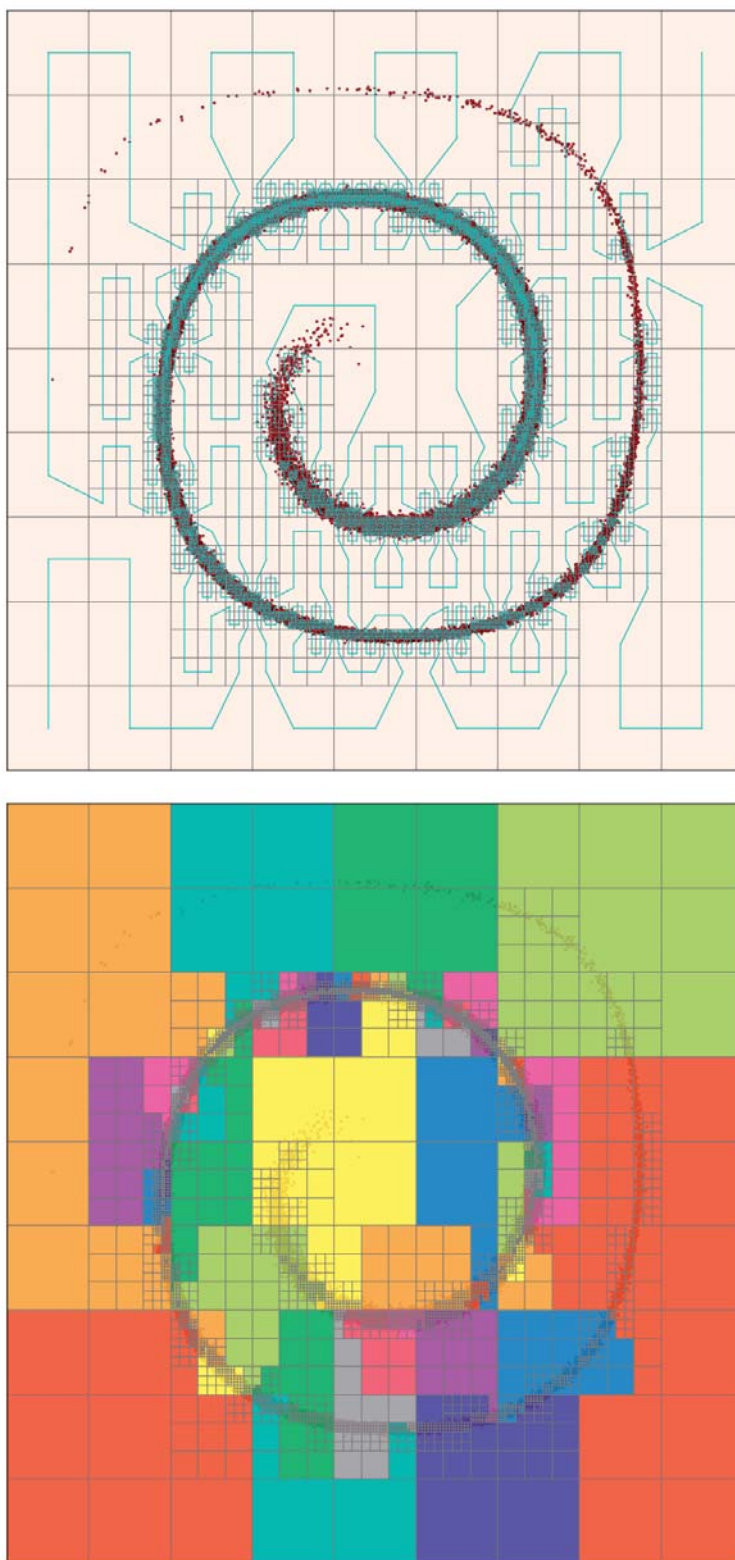


図 4.23 ペアノ曲線におけるリーフ領域の連結の様子 (上段) 及び 64 分割された小領域を 12 種類の色で塗り分けた様子 (下段)

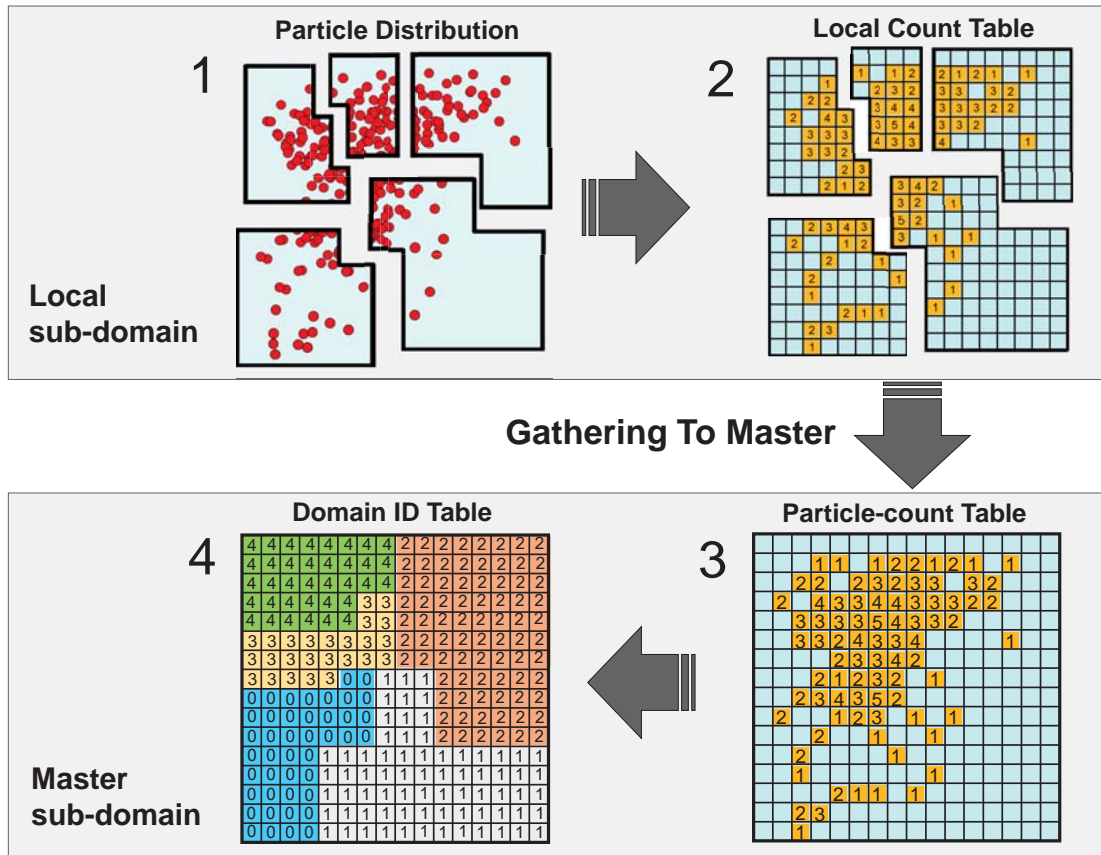


図 4.24 複数 GPU 計算における粒子数・領域番号テーブルを用いた領域分割の手順

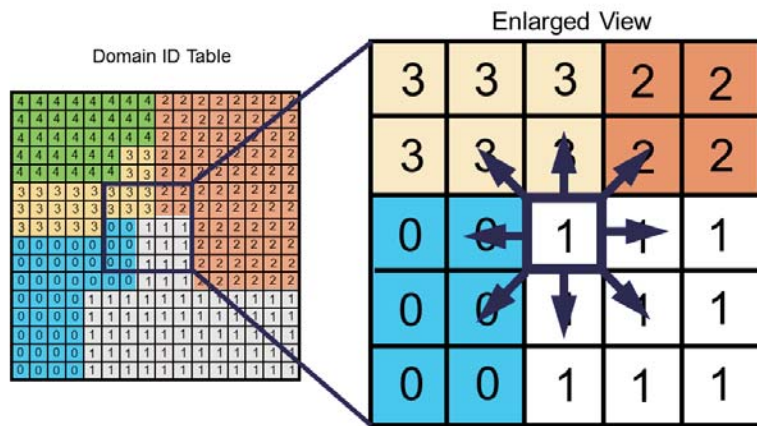


図 4.25 隣接小領域の探索の様子

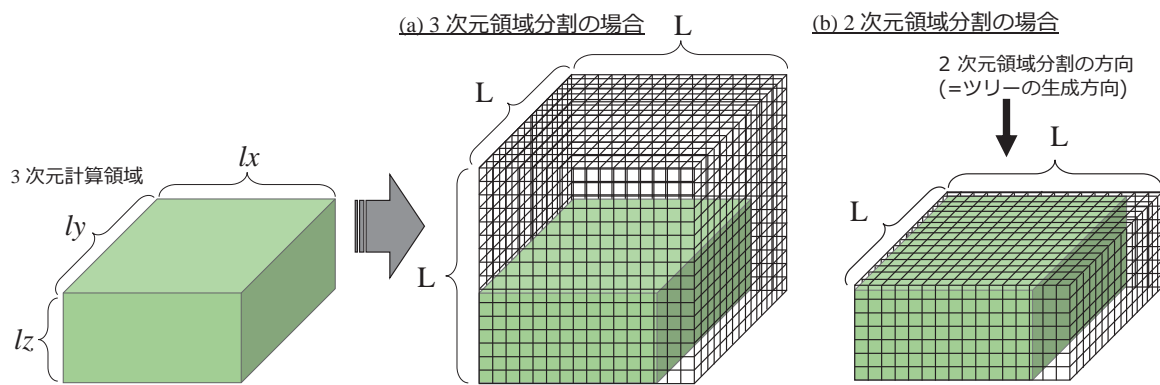


図 4.26 直方体の計算領域に対して、実際に木 (ツリー) が生成される領域 (粒子数・領域番号テーブルの空間格子を生成する領域)

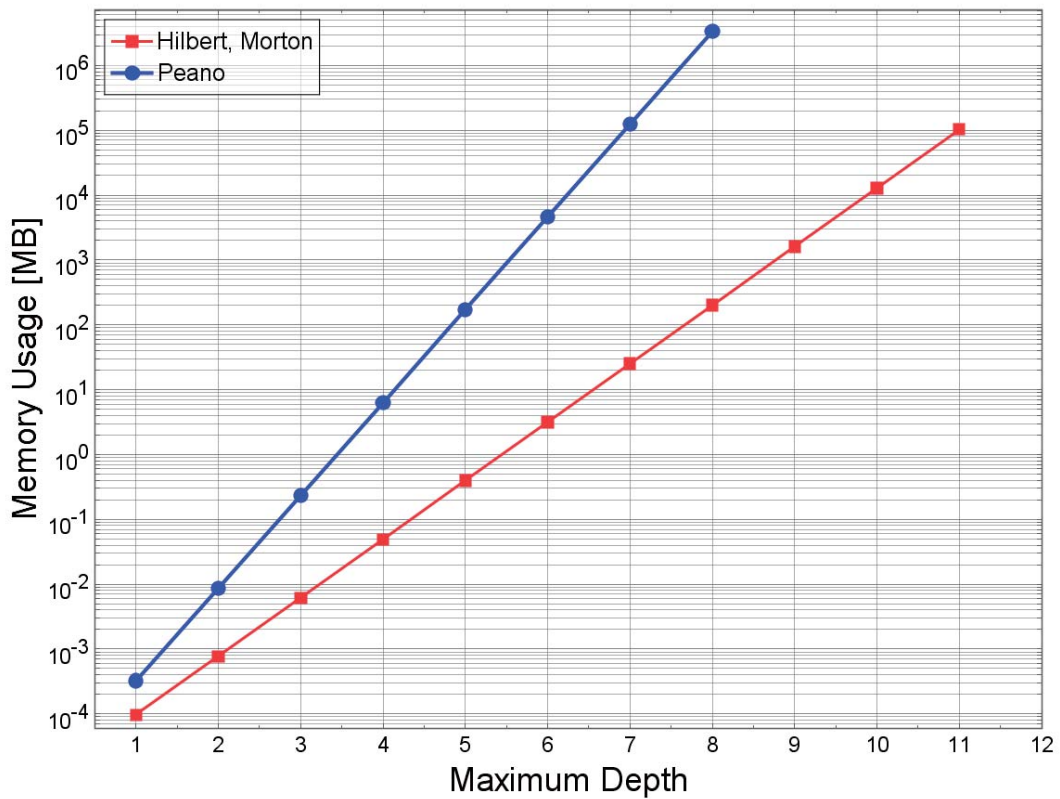


図 4.27 3次元の各空間充填曲線における領域番号・粒子数テーブルに消費されるメモリ使用量の比較

4.3 GPU スパコンにおける各動的負荷分散法の性能評価

4.3.1 スライスグリッド法と空間充填曲線の比較

東京工業大学学術国際情報センターの GPU スパコン「TSUBAME 2.5」に搭載された複数台の GPU を用いて、スライスグリッド法、又は各 2 次元空間充填曲線に基づく 2 次元動的領域分割を導入した改良型 SPH 法による 3 次元流体計算に対して強・弱スケールリングを実施し並列化効率を比較する。

GPU 一台あたりの計算領域を $9.0 \text{ m} \times 9.5 \text{ m} \times 10 \text{ m}$ とし、そこに $9.0 \text{ m} \times 8.0 \text{ m} \times 10 \text{ m}$ の領域に 0.1 m 間隔で粒子を置き水柱を設置する。鉛直方向と水平な方向に対してスライスグリッド法または 2 次元空間充填曲線を用いた 2 次元動的領域分割を行い、1 台の GPU あたりに 441,535 個の粒子を割り当てる。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して弱スケールリングを実行する。性能評価の指標として N_{total} を全

粒子数, T を計算時間として, 以下に定義される P_{weak} を導入する.

$$P_{weak} := N_{total}/T \quad (4.15)$$

時間刻み幅を 5.0×10^{-4} とし初期の 2,000 ステップに対して計算時間の測定を行い, そこから平均の 1 ステップあたりの計算時間を求め, 式 (4.15) に従って性能値を算出した (データのばらつきによる, いわゆる統計誤差については各測定点で 1% 未満であることを確認している, また, 負荷分散のずれ幅は $R_{load} = 1.0\%$ になるよう N_{min} を設定している).

弱スケーリングの結果を図 4.28 に示す. 三角印付きの赤線が 2 次元ペアノ曲線を, 四角印付きの青線が 2 次元モートン曲線を, 丸印付きの緑線が 2 次元ヒルベルト曲線を用いた場合の実行性能の測定値を表し, 菱形印付きの灰色の点線がスライスグリッド法による実行性能の測定値を表している. 図 4.28 では並列化効率を比較するために始点である 4 GPU の性能値を 2 次元ペアノ曲線の場合に規格化して示している. 256 GPU を用いた

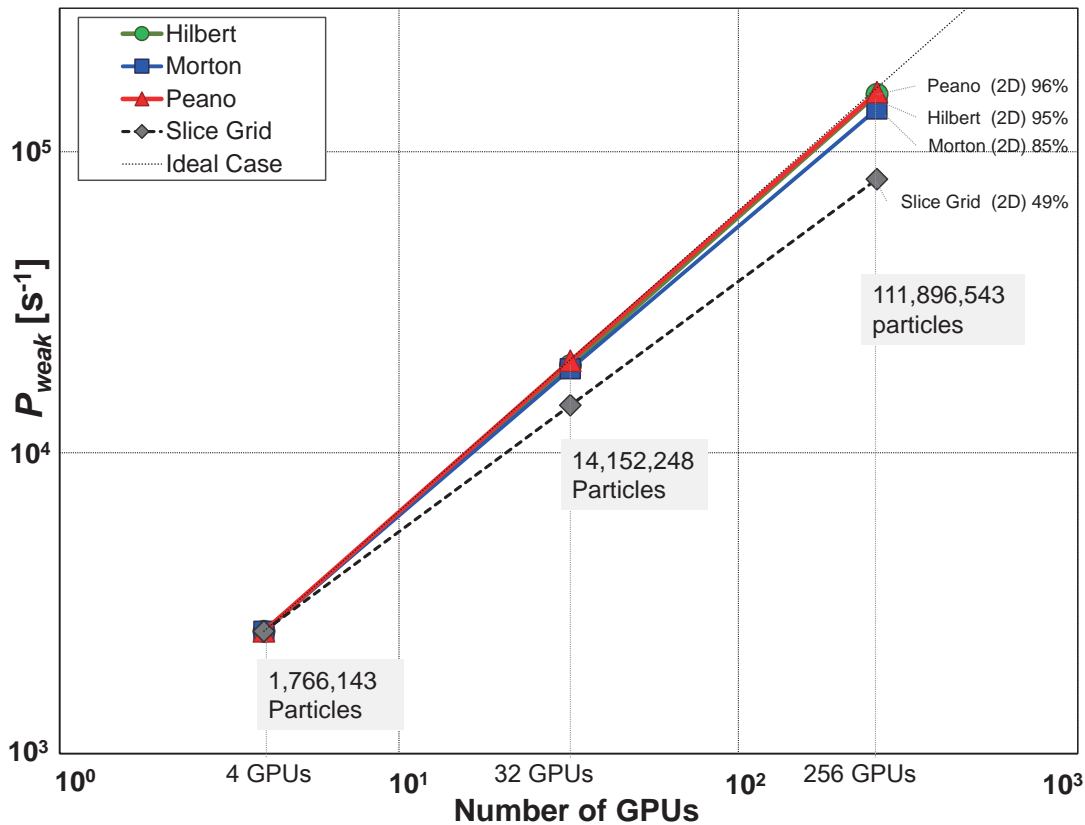


図 4.28 弱スケーリングによるスライスグリッド法と 2 次元空間充填曲線の実行性能の比較

時点で、4 GPU の測定値 (2 次元ペアノ曲線) から外装した理想性能と比べて、2 次元ペアノ曲線を用いた場合に 96%、2 次元ヒルベルト曲線を用いた場合に 95%、2 次元モートン曲線の場合に 85% の並列化効率を達成した。一方、スライスグリッド法を用いた場合は 49% であり、空間充填曲線を用いた場合に大幅に性能向上できることがわかった。

改良型 SPH 法では 1 回の計算ステップの間に 3 回のハロー領域の GPU 間通信と、1 回の粒子移動の GPU 間通信を行い、全通信時間の多くはハロー領域の通信が占めることを第 4.1.4 節で述べたが、今回の測定でも通信時間全体の 97% 以上がハロー領域の通信時間であった。

表 4.4 弱スケーリングにおける小領域間の最大接続数の変化

	2次元ヒルベルト曲線	2次元モートン曲線	2次元ペアノ曲線	スライスグリッド法
4 GPU	3	3	3	4
32 GPU	9	11	8	10
256 GPU	9	14	9	34

各測定点における小領域間の最大接続数を調べると表 4.4 の様になり、空間充填曲線を用いた場合、GPU 数が増えても 32 GPU と 256 GPU とで最大接続数が変わらないことが分かり、空間充填曲線のもつ高い局所性が確認できる。ヒルベルト曲線とペアノ曲線では曲線が交差しないために隣接接続性に優れており、モートン曲線では曲線が交差しジャンプして一つの領域が分化するためヒルベルト曲線やペアノ曲線に比べるとやや接続数が増える傾向にあることが確認できる。一方、スライスグリッド法の場合、小領域の接続数は水平方向と垂直方向の分割数によって異なるため、ここでは分割数を変えて複数種類について測定し、最大性能を示した場合 (=図 4.28 の結果が得られた場合) の最大接続数を示している。2 次元ヒルベルト曲線や 2 次元ペアノ曲線の場合と比べると最大接続数が 4 倍近く、2 次元モートン曲線と比較しても 2.4 倍と接続数が多い。小領域間の接続数が増加すると粒子データの転送先が増えるため、MPI_Isend などの非同期通信を用いた場合にも for 文 (繰り返し文) で順次転送処理を行うため通信処理にかかるコストが増加する。表 4.4 で示したスライスグリッド法や 2 次元モートン曲線における最大接続数の増加は、実行性能の低下の大きな要因のひとつであると言える。

図 4.28 における 3 つの弱スケーリングの測定 (4 GPU, 1,766,143 粒子による測定点, 32 GPU, 14,152,248 粒子の測定点, 及び 256 GPU, 111,896,543 粒子の測定点) から、同一計算条件のもと GPU の台数を増加させて強スケーリングを測定した結果を図 4.29 に示した。ここで、図 4.29 の縦軸の P_{strong} は式 (4.15) と同義である。強スケーリングにおいても、空間充填曲線を用いた場合に並列化効率が大幅に向上することが確認できる。ス

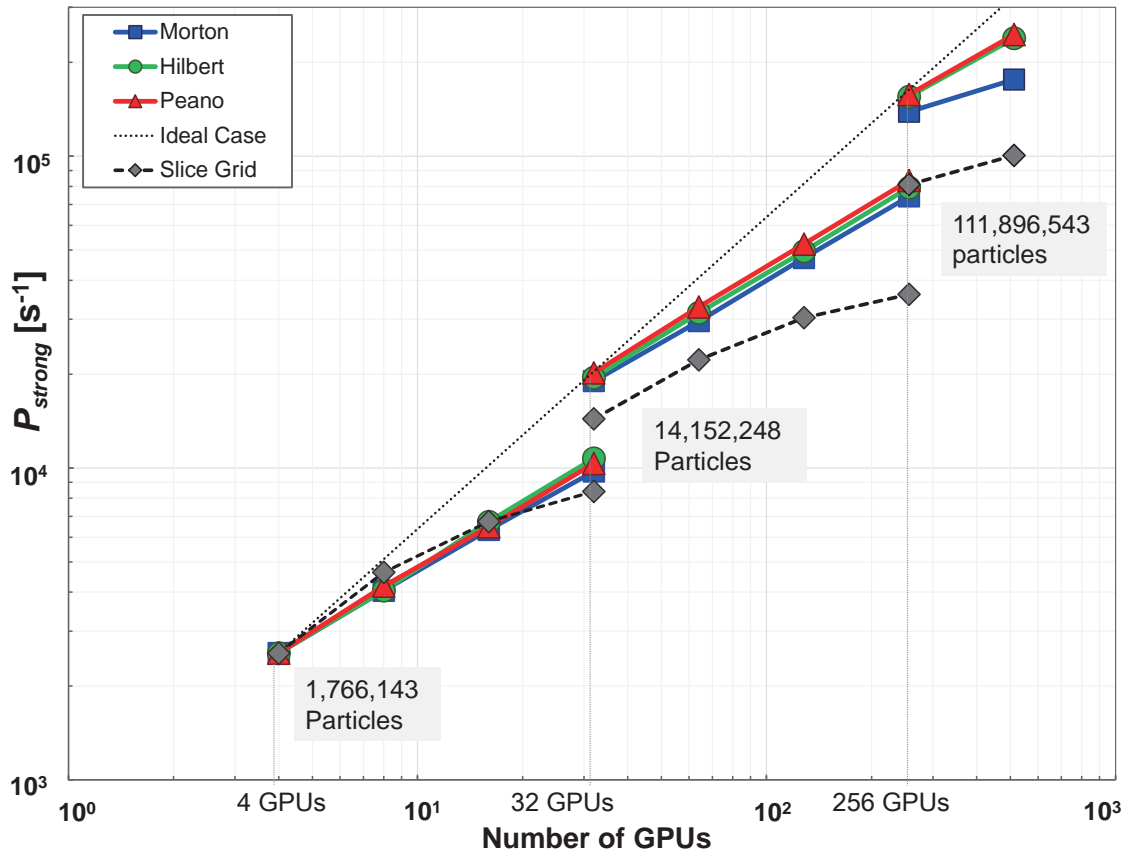


図 4.29 図 4.28 の各測定点において GPU の台数を増加させて測定した強スケーリングの結果

ライスグリッド法の場合，第 4.1.5 節で示した DEM による粉体計算の強スケーリングと同様，GPU 数が増加するにつれてスライスグリッド法に特徴的な弧を描くような実行性能の低下が確認できる．空間充填曲線を用いた場合，4 GPU，32 GPU，256 GPU のそれぞれを始点とするいずれの測定でも直線的な性能の増加となっている．弱スケーリングにおける 3 つの始点の段階で十分に通信コストが抑制されている必要があり，今後，計算に対する演算の隠蔽 (オーバーラップ) 手法の導入による並列化効率の向上が課題である．

図 4.29 の強スケーリングにおける最後の一山の始点である 111,896,543 粒子，256 GPU による流体計算を実行した際の，動的負荷分散 1 回あたりの計算時間の内訳を図 4.30 に示す．2,000 ステップの計算において 10 回に 1 回の頻度で動的負荷分散を実行し，全部で 200 回行われたうちの平均値を示している．流体計算 1 ステップあたりの合計の実行時間が約 800 msec 前後であるのに対し，2 次元ヒルベルト曲線，2 次元モートン曲線，2 次元ペアノ曲線のいずれの場合においても動的負荷分散を 1 回実行するのに要する時間は，リーフ・マイグレーションを含めても最大で 275 msec となり，動的負荷分散 1 回あたりのオーバーヘッドは，1 ステップ当たりの全計算時間の 1/3 程度であった．

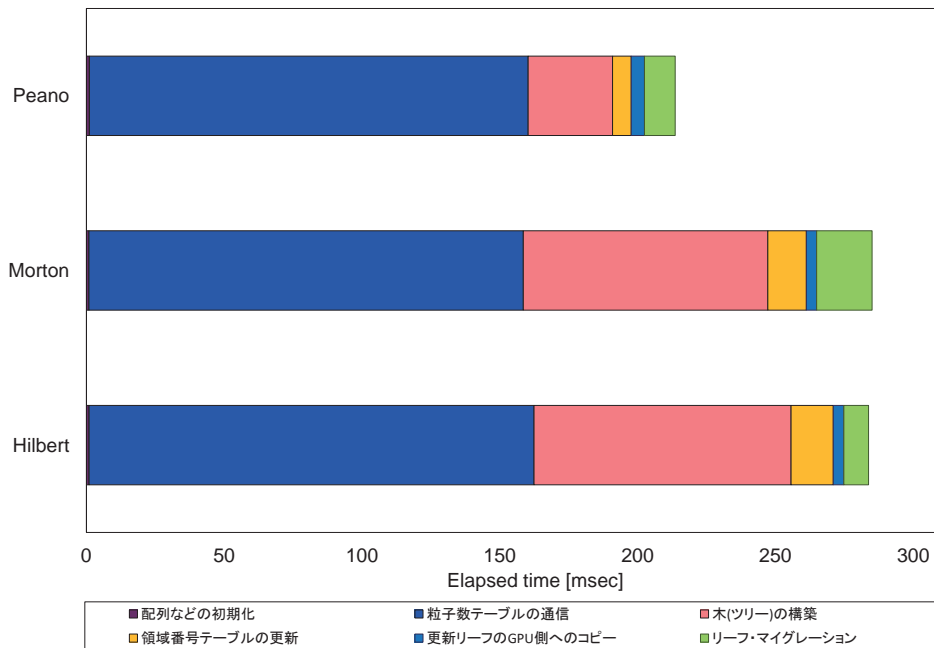


図 4.30 動的負荷分散を 1 回実行するのに要する計算時間の内訳

4.3.2 パッシブ・スカラー粒子計算による 3 次元領域分割法の検討

65,536 個の粒子を 3 次元空間に配置し、計算領域を 32 個の小領域に分割する。周期 T が 8 秒となる渦度速度場：

$$u(r, t) = -2 \cos(\pi t/T) \sin(\pi x) \sin(\pi y) \cos(\pi z) \sin(\pi t) \quad (4.16)$$

$$v(r, t) = 2 \cos(\pi t/T) \cos(\pi x) \sin(\pi y) \sin(\pi z) \sin(\pi t) \quad (4.17)$$

のもと時間刻み幅を $dt = 0.002$ として 1 周期分に相当する 4,000 ステップ分を計算する。それぞれの 3 次元空間充填曲線における粒子分布の変化と 3 次元領域分割を行う様子を図 4.36 に示す。また、このときの 3 次元ペアノ曲線で生成された 27 分木のリーフを例として半透明のグリッドで図 4.37 の左側に、それらのリーフの接続の様子を中央に、速度場の目安として水平方向の速度分布の変化を右側に示す。

同一計算条件のもと粒子数を 1 億 5,300 万個に増やし、256 台の GPU を用いて計算した際のメモリ使用量、負荷分散の精度、小領域間の接続関係を比較した。

負荷分散の精度

第 4.2.2 節で述べたように、もし仮に木 (ツリー) の再帰分割において各リーフに含まれる粒子数が 1 個か 0 個になるまで細分化を行い、すべてのリーフを空間充填曲線で辿りながら領域分割を実行したとすると、全粒子数 N_{total} が領域数 N_{sub} で余りなく割り切れる場合には負荷分散は完全に達成することができる。しかし、現状ではリーフ内部の粒子数が 1 個以下になるまで再帰分割を行うことはツリー構築の時間、探索に要する時間、メモリ容量のいずれも高コストで現実的でないため、本研究では「所定の粒子数」 N_{min} を設け、リーフ内部の粒子数が N_{min} 以下になるまで再帰分割を行うようにしている。

N_{min} が全体の負荷分散の精度に与える影響を調べるため、A) N_{min} を均等粒子数 N_{std} の 1.5 % とした場合、B) N_{min} を均等粒子数 N_{std} の 1.0 % とした場合、C) N_{min} を均等粒子数 N_{std} の 0.5 % とした場合、の全部で 3 通りについて 3 次元パッシブスカラー粒子計算を 3 次元空間充填曲線による 3 次元領域分割のもと 256 台の GPU を用いて行い、時間方向と空間方向 (GPU 間のばらつき) の負荷分散の精度を比較した。まず、時間方向の負荷分散のずれ幅 R_{load} の変化を図 4.31 に示す。一方、全部で 4,000 ステップのうち、500 ステップ目に各 GPU で測定された負荷分散のずれ幅 R_{load} の GPU 毎の値を図 4.32 に示す。これらの結果から分かることとして、時間方向の変化について、ペアノ曲線だけが単独で安定しているのは A) でヒルベルト曲線とモートン曲線は C) で安定するなど、 N_{min}/N_{std} の値によってそれぞれ異なることが確認できる。また、時間方向と空間方向のいずれに対しても、 N_{min}/N_{std} の割合が、ほぼそのまま全体の負荷分散の精度を決定していることが分かる。すなわち、セル内の所定の粒子数をいくつに設定するかによって GPU 間の動的負荷分散の精度を制御できることが分かる。

小領域間の接続関係

N_{min}/N_{std} の異なる A)~C) のそれぞれの場合において、隣接する小領域間での最大接続数の時間変化を測定した結果を図 4.33 に示す。A)~C) で描く曲線の変化は同様になり違いは見られなかった。C) の場合を例に詳しく述べると、モートン曲線を用いた場合、4,000 ステップの間に最大接続数は 55 から 85 の間を変動しており、3 種類の曲線の中で一番接続数が多い結果となった。ペアノ曲線では 39 から 79 の間を変動しており、各時刻における最大接続数の幅が広い。ヒルベルト曲線では 4,000 ステップの間に最大接続数は 40 から 69 の間を変動しており、最大接続数の変動は 3 種類の曲線の中では最も抑えられていた。C) において各空間充填曲線を用いた場合の 500 ステップ目における接続数のヒストグラムを図 4.34 に示す。このヒストグラムから、3 種類のいずれの空間充填曲線を用いた場合も、領域間の接続を悪化させているのは高々数個の少数の小領域が原因であり、多くの小領域は 20 前後の領域と接続されていることが確認できる。接続数の悪い特

定の小領域について領域形状を改善するなどにより、3次元空間充填曲線を用いた3次元領域分割における接続数の悪化を改良できる可能性があり今後の課題である。

メモリ使用量

本研究では第4.2.5節で述べた様に木(ツリー)がある深さより深くならないよう最大深さを設定している。3次元空間充填曲線による3次元領域分割を用いた本問題においては、予め設定したツリーの最大深さはヒルベルト曲線とモートン曲線の場合に8階層、ペアノ曲線では5階層である。一方、各空間充填曲線で実際に構築されたツリーの深さは計算開始から終始、ヒルベルト曲線とモートン曲線では7階層、ペアノ曲線では5階層となった。

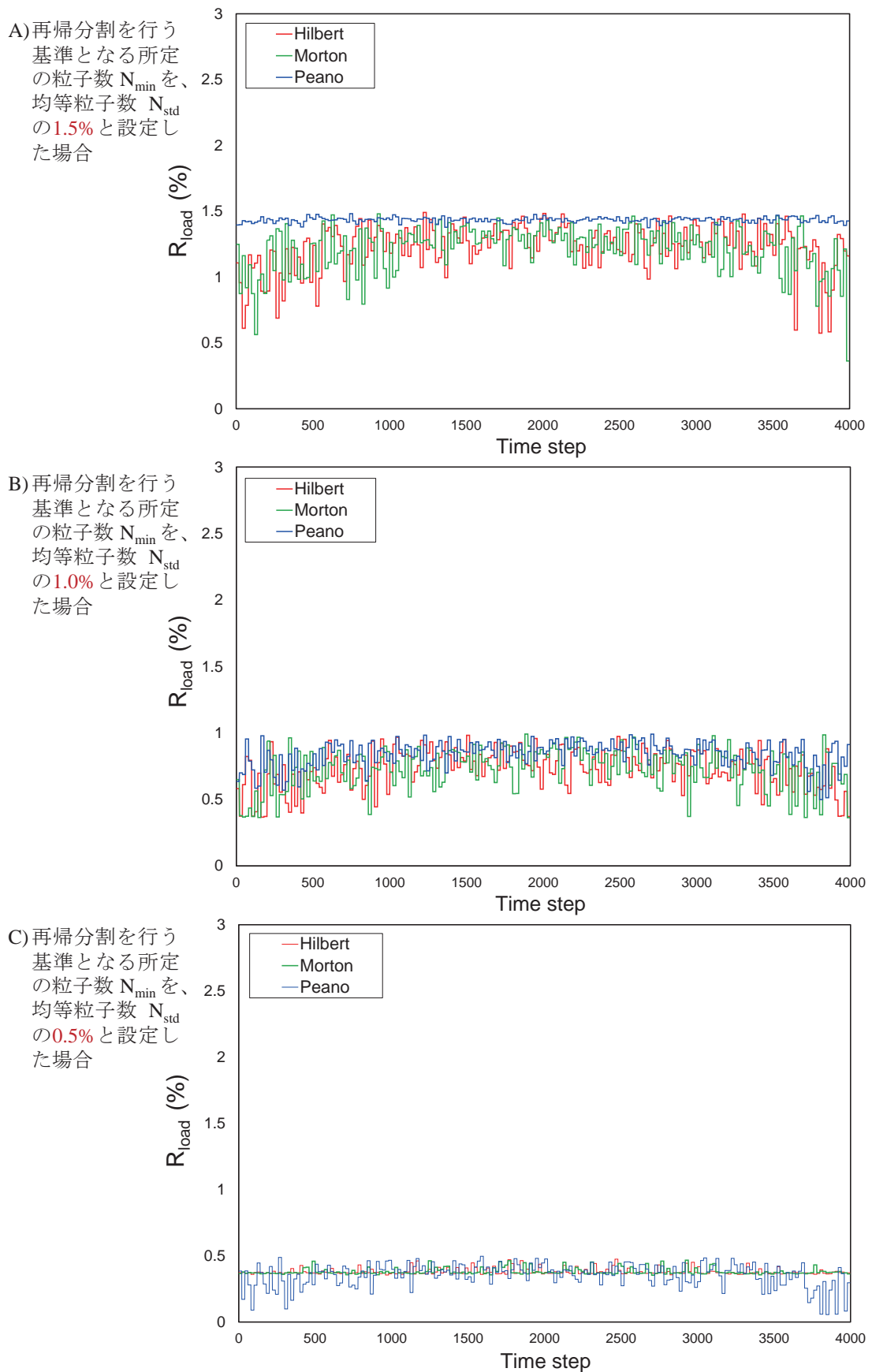
最大深さはこのように上限を設けているため変わらないが、リーフの枚数は粒子分布により最大深さの範囲内で変化し、それによりツリーによるメモリ消費量は異なる。ヒルベルト曲線、モートン曲線、ペアノ曲線の各曲線におけるツリーの時間方向のメモリ使用量の変化を、 N_{min}/N_{std} の異なるA)~C)のそれぞれの場合で比較した結果を図4.35に示す。横軸は時間ステップを、縦軸がメモリ使用量を表している。ここでペアノ曲線の場合だけ縦軸が0から70の範囲で表示している。

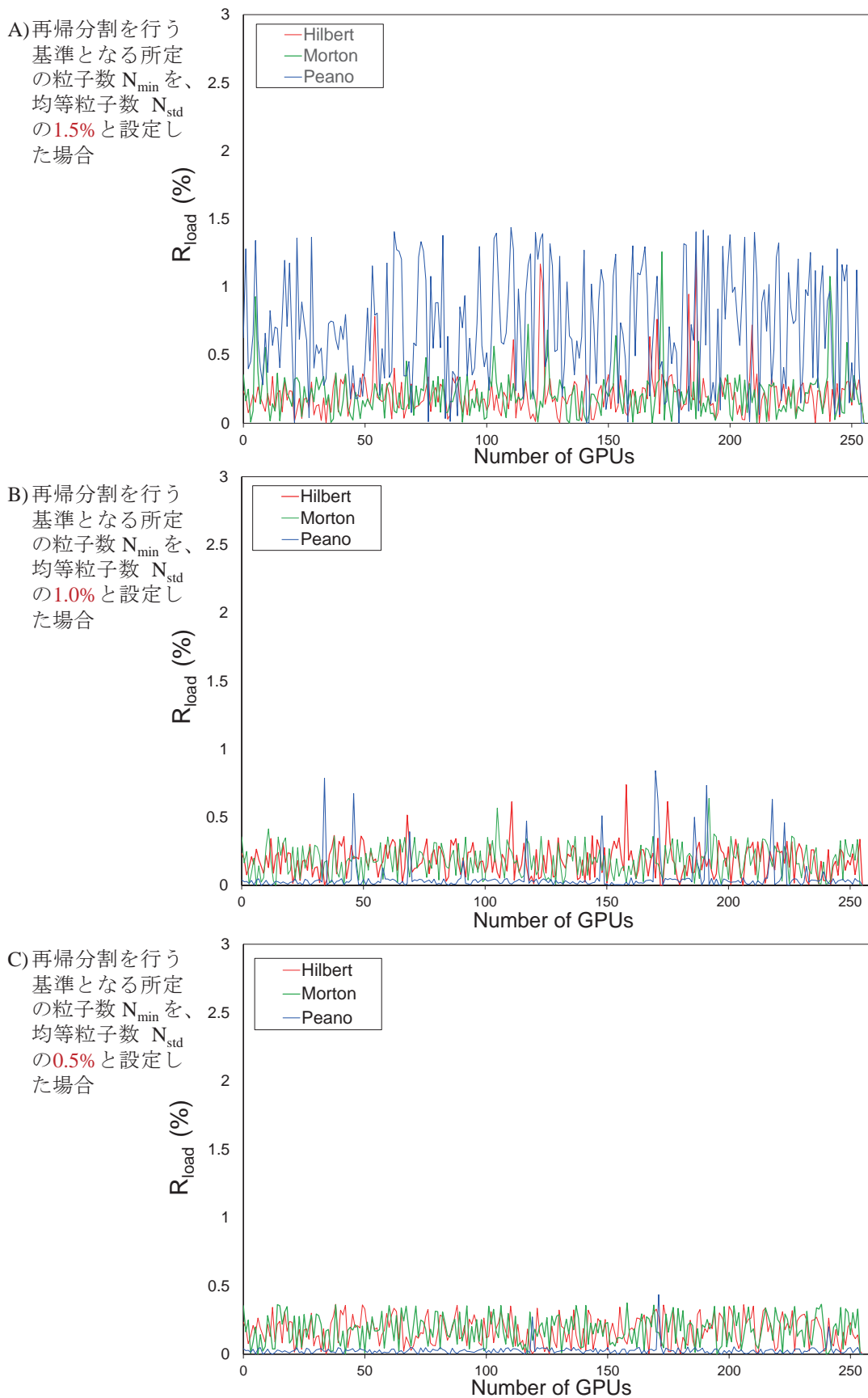
図4.35について、ヒルベルト曲線及びモートン曲線と、ペアノ曲線とで結果が大きく異なることが確認できる。ヒルベルト曲線とモートン曲線ではメモリ消費量は9MB~13MBの間で変化し、A)からC)へと負荷分散の精度を上げるにつれて次第に増加する傾向にある。一方、ペアノ曲線の場合、A)からB)に負荷分散の精度を上げた段階でメモリ使用量が20倍近く増加している。粒子分布に対して生成されるリーフの数が増えたためであり、1:3分割により一回の細分化で27リーフが生成されるペアノ曲線に特徴的である。さらにB)ではメモリの増減の変化の傾向がA)やC)と反転している。格子細分化の基準値である所定の粒子数 N_{min} をいくつにするかにより、各時刻の粒子分布に対してどこまで細分化を行うかどうかの判定が変わるためであり、これも1回あたりの細分化率が高いペアノ曲線の場合に特有である。

粒子数テーブルと領域番号テーブルの空間格子に要するメモリ容量はヒルベルトとモートン曲線の場合に約201MB、ペアノ曲線では約172MBで同程度であった。なお、ツリーの最大深さがペアノ曲線の場合だけ3階層低く設定されている理由は図4.27に示した空間格子の示すメモリ使用量が原因である。ペアノ曲線のツリーの最大深さをひとつ深くして6階層とすると、空間格子に消費されるメモリ使用量が4.6GBまで急増する。TSUBAME2.5に搭載されたNVIDIA Tesla K20Xのデバイス・メモリの上限は6GBであるため、粒子データを確保するメモリ領域が不足してしまい計算を行うことができなかった。空間格子を利用する現在の実装では、メモリ容量の制限されたGPUスパコン環境においては、メモリが豊富にある通常のCPUの環境とは異なり最大ツリーの設定の自由度

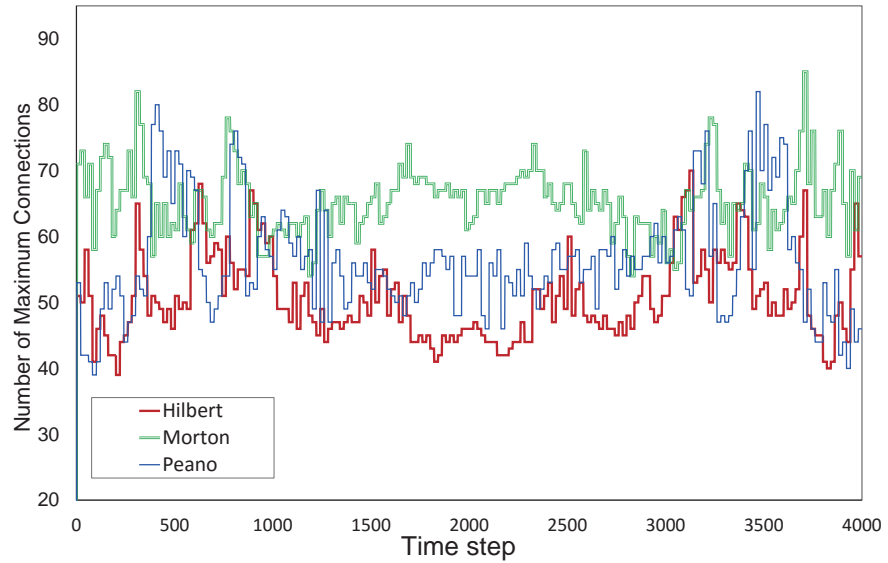
の点で 3 次元ペアノ曲線は厳しい制約を受けるといえる。

本節ではパッシブ・スカラー粒子計算を用いて 3 次元空間充填曲線による動的領域分割の性能評価を行ったが、空間格子を用いる本手法を GPU スパコンで用いる場合、3 次元ペアノ曲線による動的領域分割は、メモリ使用量の観点から厳しい制約を受けることが確認された。また、隣接する小領域との最大接続数はいずれの空間充填曲線を用いた場合も悪く、中でもモートン曲線を用いた場合に 256 個に分割された小領域のうち、最大で 79 領域が接続される結果となった。空間充填曲線による動的領域分割の利点として 3 次元領域分割の実現が容易であることが挙げられ、実際、2 次元領域分割から 3 次元領域分割への拡張自体はリーフの辿り順序を 3 次元領域分割のパターンに切り替えるだけで可能であるが、現状では、領域間の接続性が悪く局所性が悪いことが明らかになった。

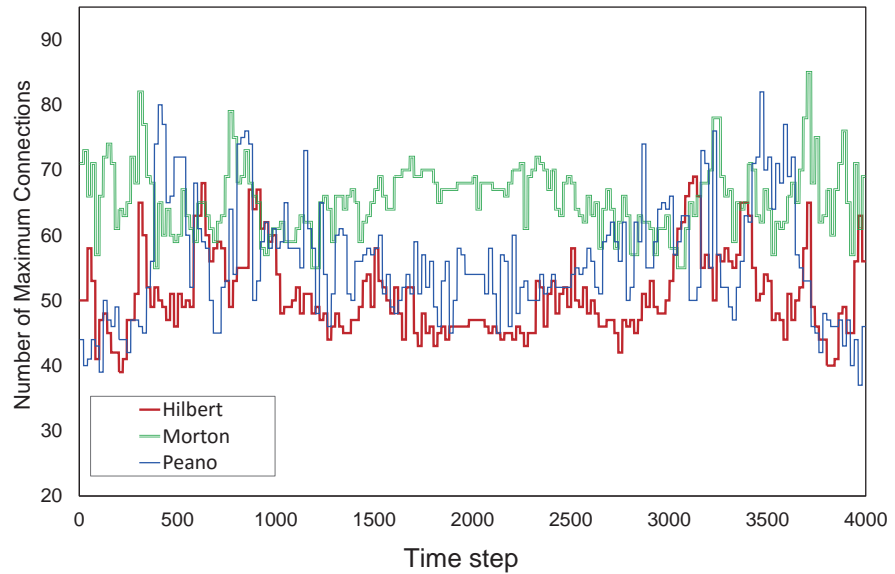
図 4.31 時間方向における動的負荷分散の精度の N_{min} への依存性

図 4.32 空間方向における動的負荷分散の精度の N_{\min} への依存性

A) 再帰分割を行う
基準となる所定
の粒子数 N_{\min} を、
均等粒子数 N_{std}
の1.5%と設定し
た場合



B) 再帰分割を行う
基準となる所定
の粒子数 N_{\min} を、
均等粒子数 N_{std}
の1.0%と設定し
た場合



C) 再帰分割を行う
基準となる所定
の粒子数 N_{\min} を、
均等粒子数 N_{std}
の0.5%と設定し
た場合

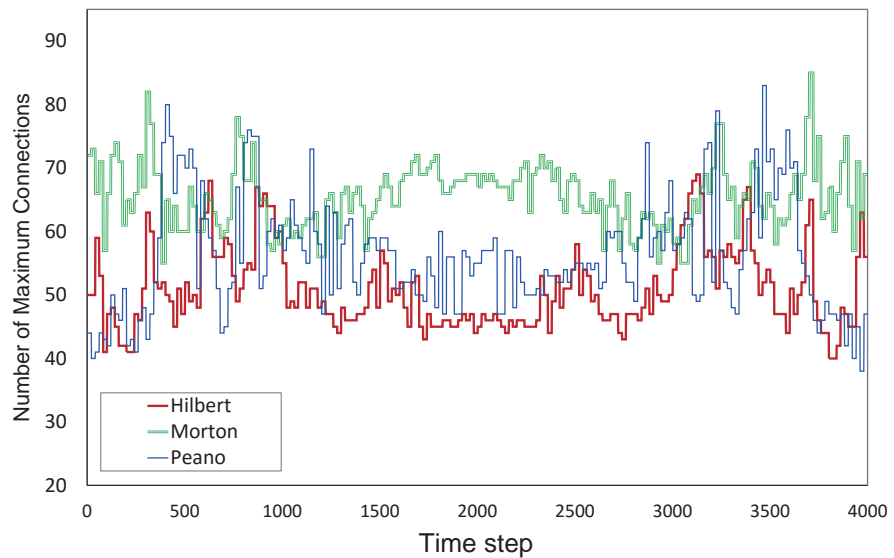


図 4.33 A)~C) のそれぞれの条件で計算した最大接続数の変化

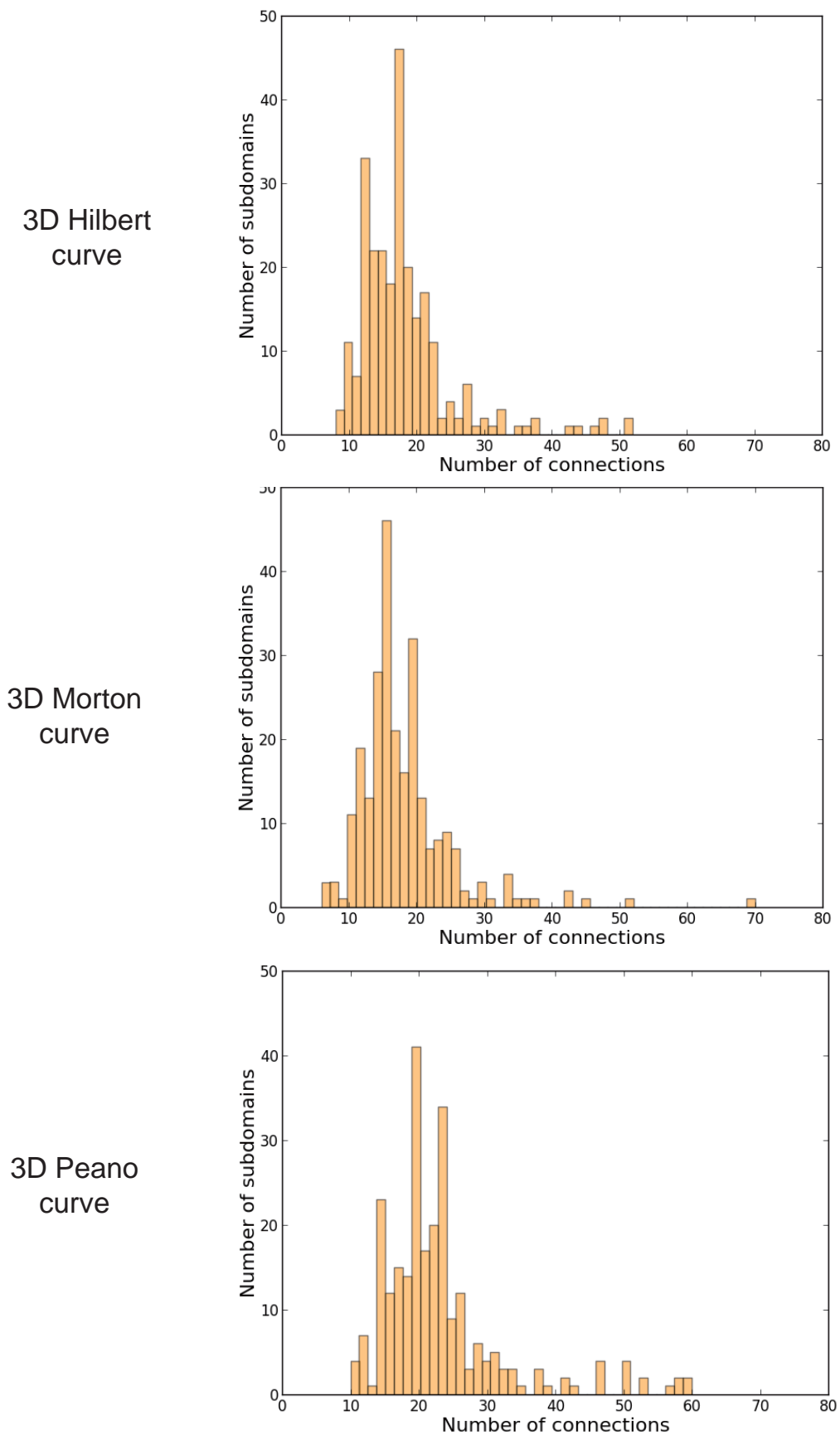


図 4.34 C) による計算時の 500 ステップ目における接続数のヒストグラム

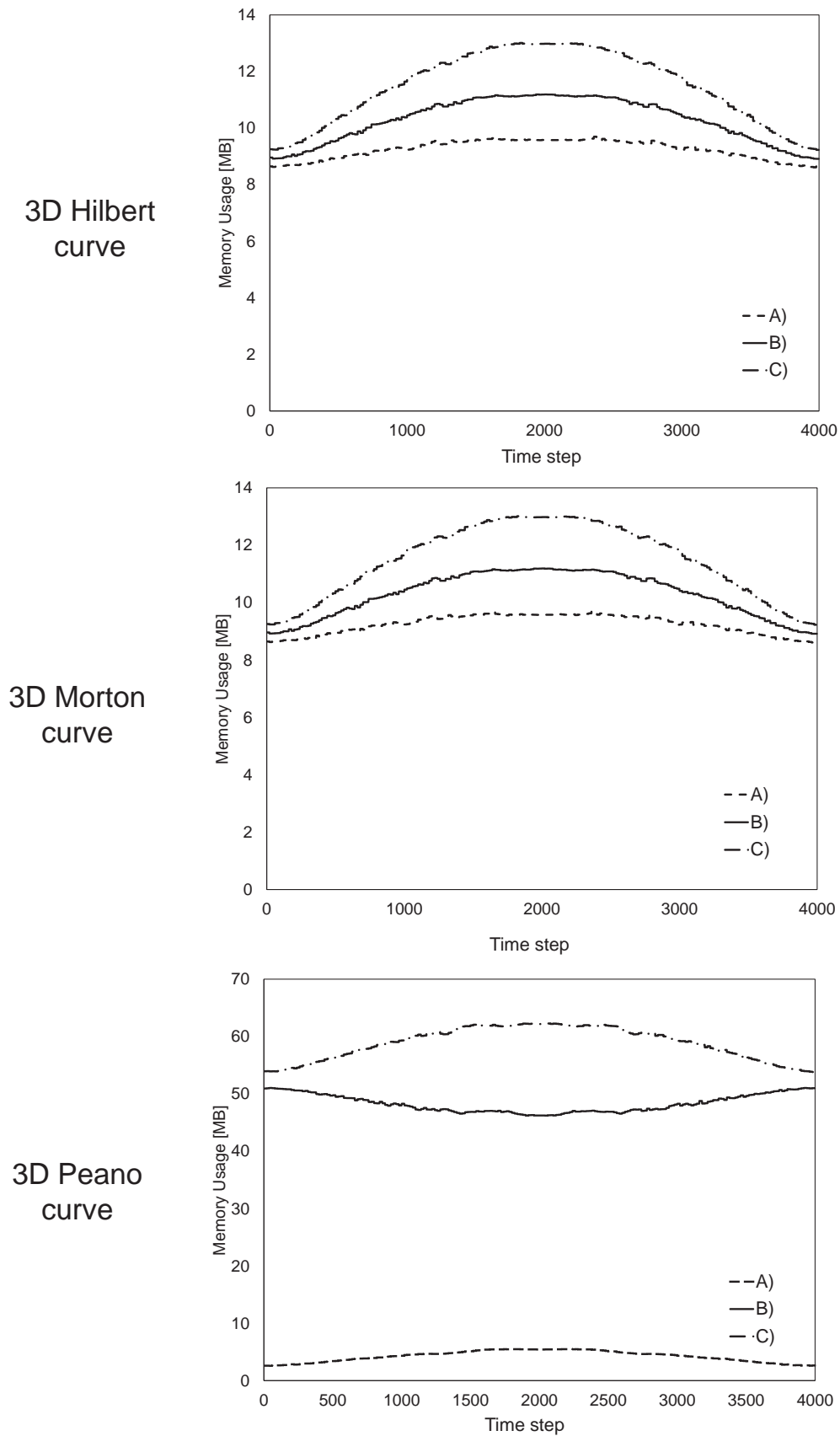


図 4.35 各空間充填曲線におけるツリーの時間方向のメモリ使用量の変化について、計算条件 A)~C) で比較した結果

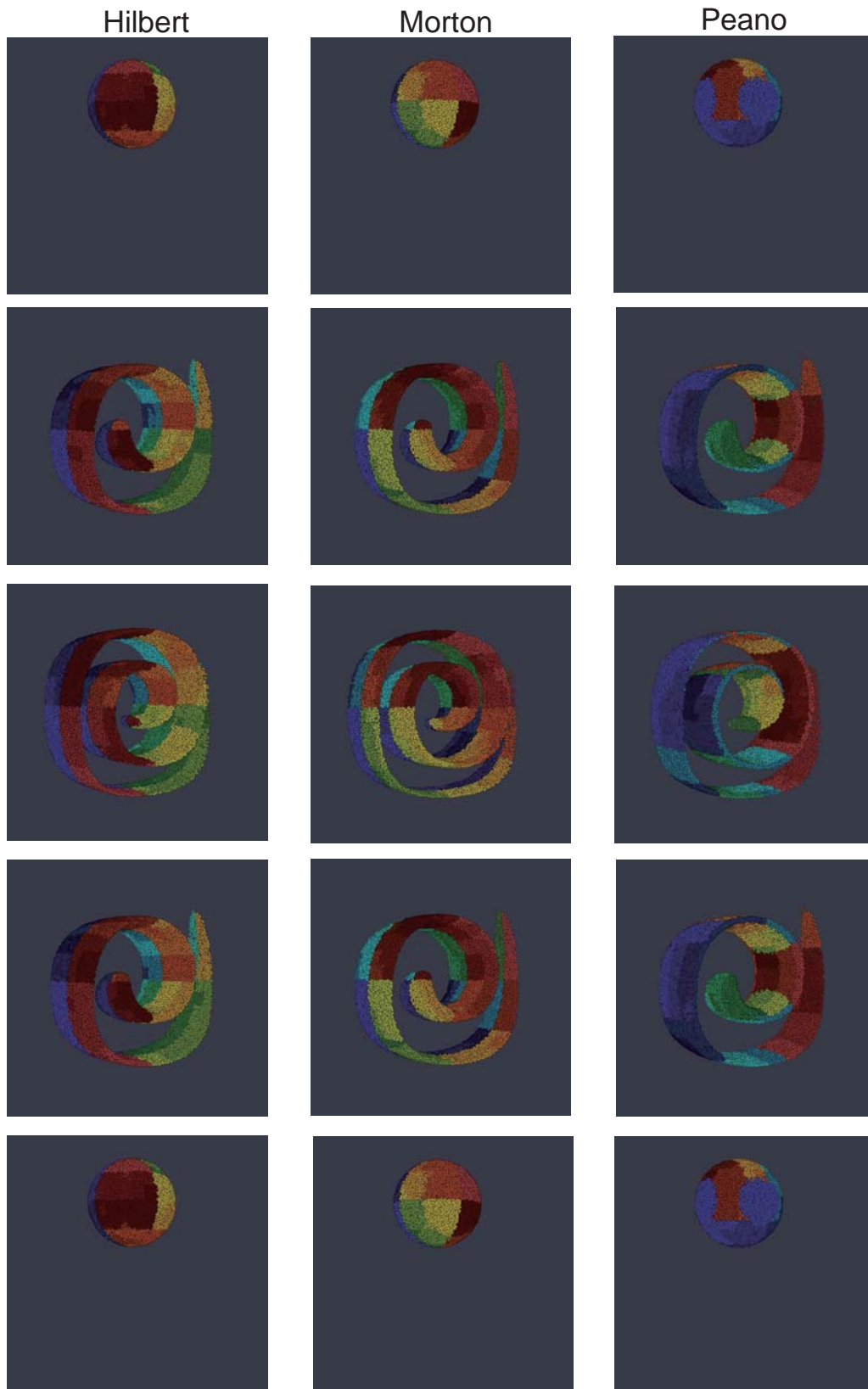


図 4.36 各空間充填曲線を用いた場合の 3 次元パッシブ・スカラー粒子計算における動的負荷分散の様子

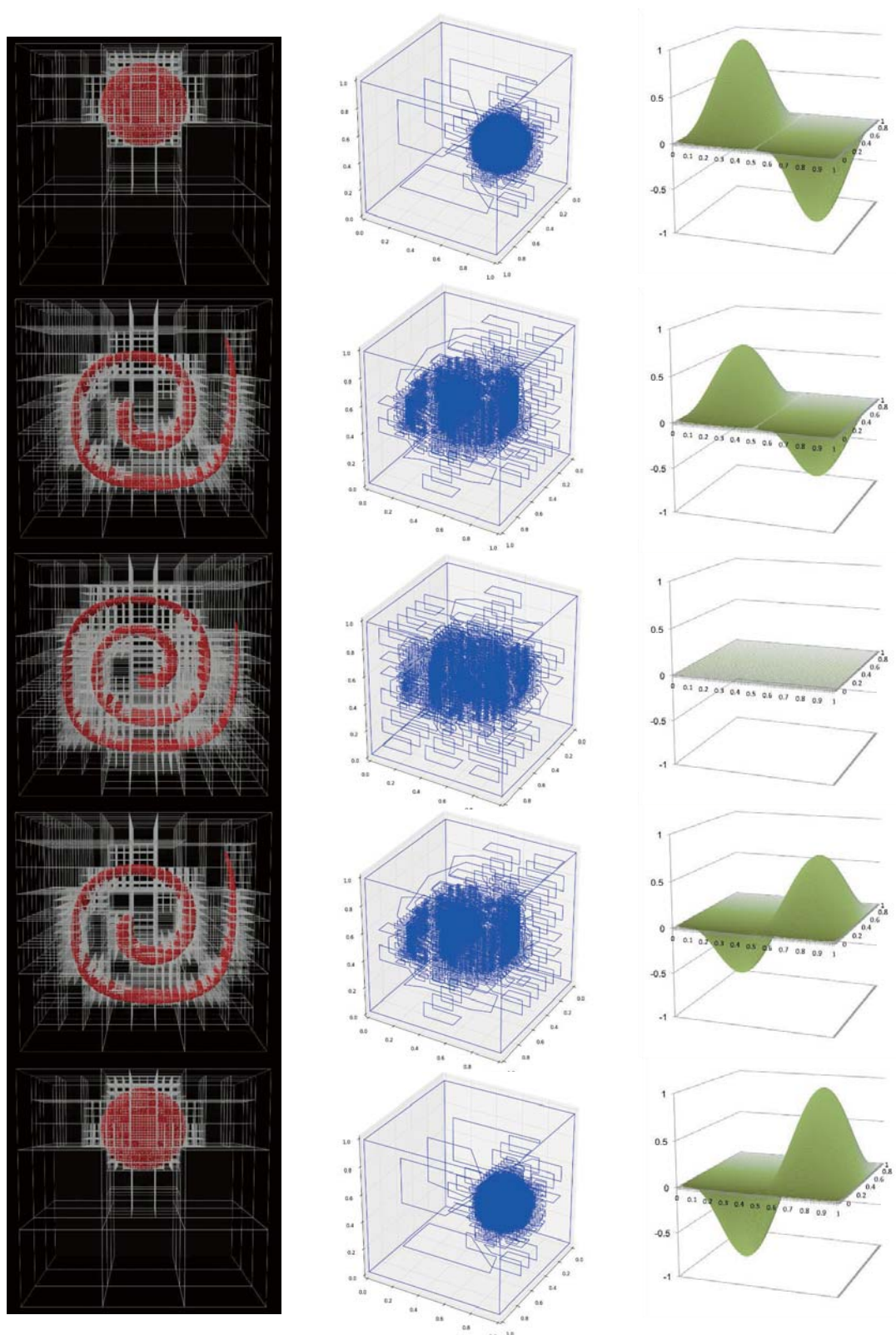


図 4.37 ペアノ曲線を用いた場合のリーフの変化 (左), リーフの接続の様子 (中央), 及び水平方向の速度分布の変化 (右)

4.3.3 空間充填曲線を用いた 2 次元領域分割と 3 次元領域分割の比較

改良型 SPH 法による流体の 3 次元ダムブレイク問題に対して、2 次元空間充填曲線に基づく 2 次元領域分割、及び 3 次元空間充填曲線に基づく 3 次元領域分割を用いた場合で計算時間を比較する。計算領域を $10\text{ m} \times 20\text{ m} \times 15\text{ m}$ とし、そこに $10\text{ m} \times 5\text{ m} \times 10\text{ m}$ の大きさの水柱を設置する。図 4.39 は 10 万粒子を使用した場合の各空間充填曲線の計算例であり、上段は初期状態を表し、下段はダム崩壊後のスナップショットを示している。(背後と手前の壁面は見やすさのため除外して表示している)。水柱及び壁面を構成するため、 0.0125 m 間隔に配置し、全部で 2 億 1,100 万個の粒子を配置する。GPU 一台あたりに 412,110 個の粒子を割り当て、全部で 512 台の GPU を使用する。ダム崩壊開始から一定時間経過後は図 4.39 における下段のようにハロー領域の通信量も変動するため、図 4.39 における上段のように粒子が等間隔に固定されて配置されハロー領域の通信量が一定に保たれた状態で計算時間を測定する。物理時間で 0.125 秒に相当する 2,000 ステップ分を計算した場合の 1 ステップ当たりの計算時間の比較を図 4.38 に示す。

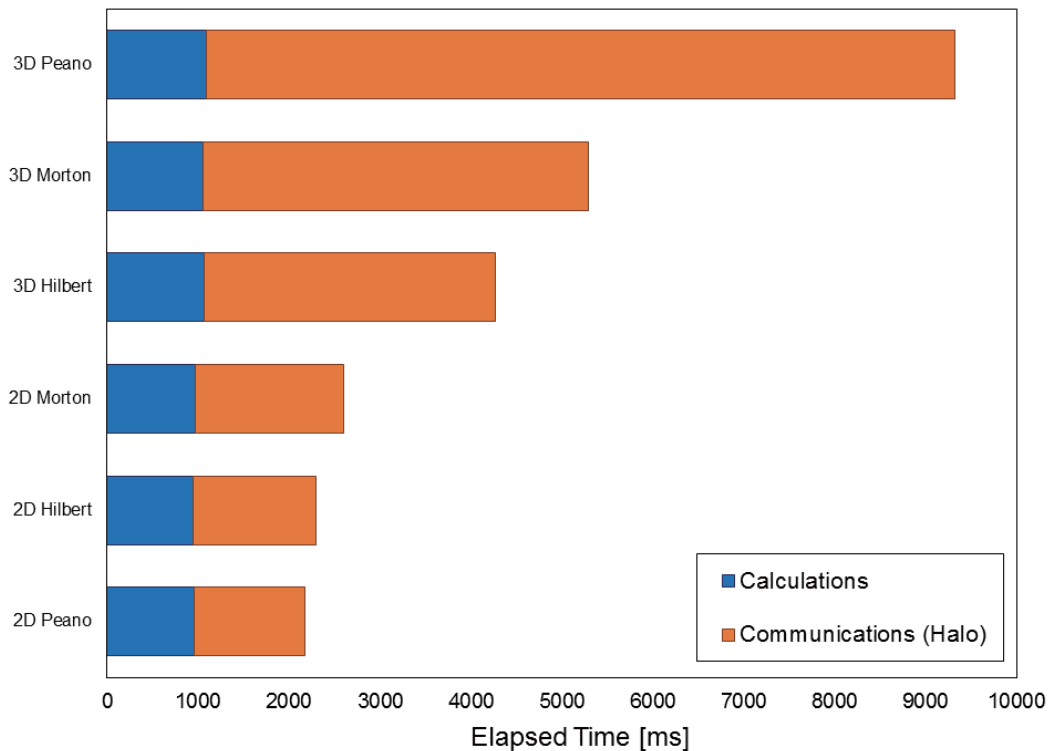


図 4.38 2 億 1,100 万個の粒子と 512 GPU を用いたダムブレイク計算における、空間充填曲線の違いによる計算時間の比較

各空間充填曲線における通信量及び通信性能の評価基準として、 i 番目の小領域における粒子データの通信量 DS_i 及びバンド幅 BW_i を以下のように定義する。

$$DS_i := \sum_{k=0}^{N_{nbr}^{(i)}} \{NP_k \times PSIZE\} \quad (4.18)$$

$$BW_i := DS_i / T \quad (4.19)$$

ここで、 $N_{nbr}^{(i)}$ は i 番目の小領域に隣接する小領域の数であり、 NP_k は隣接する小領域のうちの k 番目の小領域に転送される粒子数である。 $PSIZE$ は粒子ひとつあたりの大きさであり、本研究では倍精度計算の場合に 80 Byte である。 T は隣接するすべての小領域に粒子データを転送し終わるまでに要する時間である。データ通信量 DS_i 及びバンド幅 BW_i の最大値をそれぞれ次のように定義する。ここで、 N_{sub} はすべての小領域の数 (= 領域分割数) である。

$$DS_{max} := \max_{i \in N_{sub}} \{DS_i\} \quad (4.20)$$

$$BW_{max} := \max_{i \in N_{sub}} \{BW_i\} \quad (4.21)$$

さらに、各計算ステップにおける最大接続数 N_{nbr}^{max} を以下のように定義する。

$$N_{nbr}^{max} := \max_{i \in N_{sub}} \{N_{nbr}^{(i)}\} \quad (4.22)$$

ハロー領域の最大通信量 DS_{max} 、最大通信バンド幅 BW_{max} 、最大接続数 N_{nbr}^{max} を測定した結果を表 4.5 と表 4.6 にまとめた。ここで、表 4.5 が 3 次元空間充填曲線を用いて 3 次元動的領域分割を行った場合を、表 4.6 は 2 次元空間充填曲線を用いて 2 次元領域分割を行った場合を表している。また、それぞれの場合に設定したツリーの最大深さ dep_{max} と袖 (ハロー) 領域の大きさを表す指標である C_{halo} を示している。

図 4.38 の計算時間の測定結果と関連してまずはじめに述べておくべきこととして、3 次元ペアノ曲線を用いた 3 次元領域分割では空間格子によるメモリ容量の制限からツリーの最大深さ dep_{max} を 5 階層までしか設定できず、袖 (ハロー) 領域の 1 方向の長さがそれ以外と比較して約 2.2 倍になってしまった。実際、測定された最大データ通信量 DS_{max} も極端に多くなっている。このように条件が異なり、他の曲線による結果との比較ができないため 3 次元ペアノ曲線を用いた 3 次元領域分割の測定結果はあくまで参考値として示している。

3 次元ヒルベルト曲線や 3 次元モートン曲線を用いた 3 次元領域分割の方が、2 次元空間充填曲線を用いた 2 次元領域分割のいずれの場合よりも袖 (ハロー) 領域の最大通信量 DS_{max} が半分以下になっている。3 次元領域分割の場合の方が領域の表面積が小さくなるため妥当な結果である。一方、3 次元ヒルベルト曲線や 3 次元モートン曲線による 3 次

表 4.5 512 GPU, 2 億 1,100 万粒子の改良型 SPH 法のダムブレイク問題に対して, 3 次元空間充填曲線による 3 次元領域分割を適用した場合の最大データ通信量, 通信バンド幅, 最大接続数

	3D Hilbert	3D Morton	3D Peano (参考値)
dep_{max}	9	9	5
C_{halo}	1.04	1.04	2.19
DS_{max} [MB]	24.6	26.0	133.1
BW_{max} [GB/s]	1.7	1.6	1.6
N_{nbr}^{max}	71	103	54

表 4.6 512 GPU, 2 億 1,100 万粒子の改良型 SPH 法のダムブレイク問題に対して, 2 次元空間充填曲線による 2 次元領域分割を適用した場合の最大データ通信量, 通信バンド幅, 最大接続数

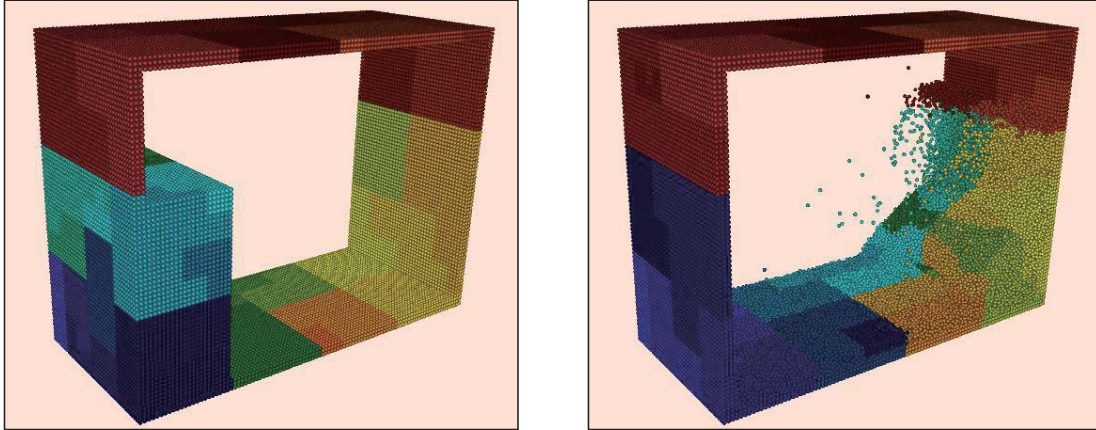
	2D Hilbert	2D Morton	2D Peano
dep_{max}	9	9	6
C_{halo}	1.04	1.04	1.04
DS_{max} [MB]	64.3	69.3	69.3
BW_{max} [GB/s]	2.11	2.13	2.76
N_{nbr}^{max}	14	18	24

元領域分割の場合に, 2 次元領域分割の場合と比較して袖領域の最大通信量 DS_{max} が半分以下になっているのに, 最大通信バンド幅 BW_{max} は 2 次元曲線による 2 次元領域分割の方がいずれも良い結果になっている. この理由は次のように考えられる. BW_{max} は転送されたデータ量をすべての通信を完了するまでに要した時間で割った値であるから, 例えば隣接する接続数が極端に多くなった場合には, 転送先についての繰り返し処理のコストが増加することで BW_{max} は低下してしまう. 実際, 3 次元空間充填曲線を用いた 3 次元領域分割における最大接続数 N_{nbr}^{max} を見てみると, いずれも 2 次元空間充填曲線による 2 次元領域分割を行った場合よりも 6 倍近くに及んでいるため, 3 次元曲線による 3 次元領域分割を用いた場合の通信性能が悪い原因は隣接接続数が多すぎるのが原因である可能性が高いといえる.

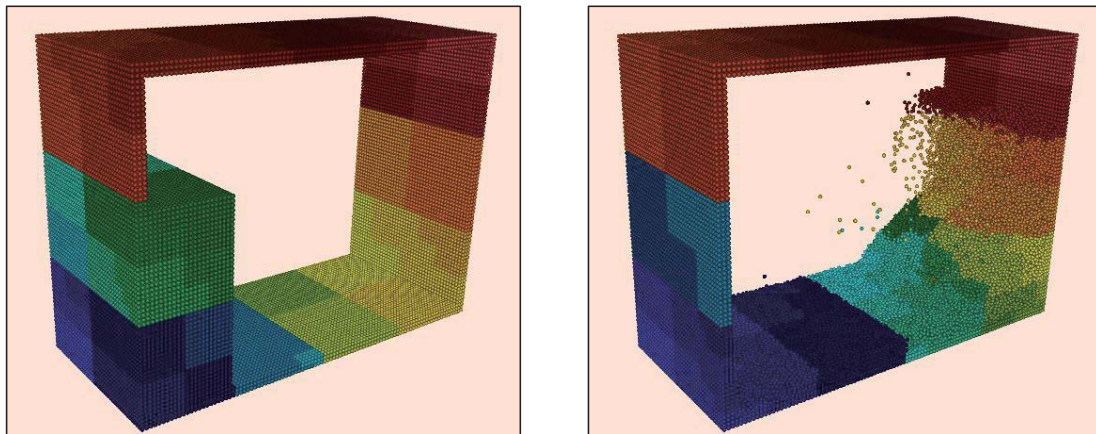
図 4.38 における流体計算時間 (青色) はそれぞれの曲線により若干のばらつきがあるが、図 4.38 はひとつの小領域についての内訳であり、この問題では負荷分散のずれ幅 R_{load} を 5% と粗く設定したために、その分のばらつきが原因だと考えられる。

本節で得られた結果はあくまで特定の計算条件に対する一例にすぎない。例えば全体の領域形状が高いアスペクト比になる問題に対しては、最大データ通信量が今回の測定以上に 2 次元空間充填曲線による 2 次元領域分割の場合に増加すると考えられ、そのような場合には隣接との接続関係が今と変わらなくても (特に改善されなくても) 3 次元空間充填曲線による 3 次元領域分割の方が良くなるといえる。つまりは隣接領域との接続数に依存する転送処理のコストとデータ通信量のどちらが勝るかという問題である。本研究では粒子法の複数 GPU 計算に対して 3 次元空間充填曲線を用いることによりスライスグリッド法では難しかった 3 次元領域分割を適用することに成功したが、これらにおける並列化効率や実行性能の議論については今後の課題である。

(a) Hilbert Curve



(b) Morton Curve



(b) Peano Curve

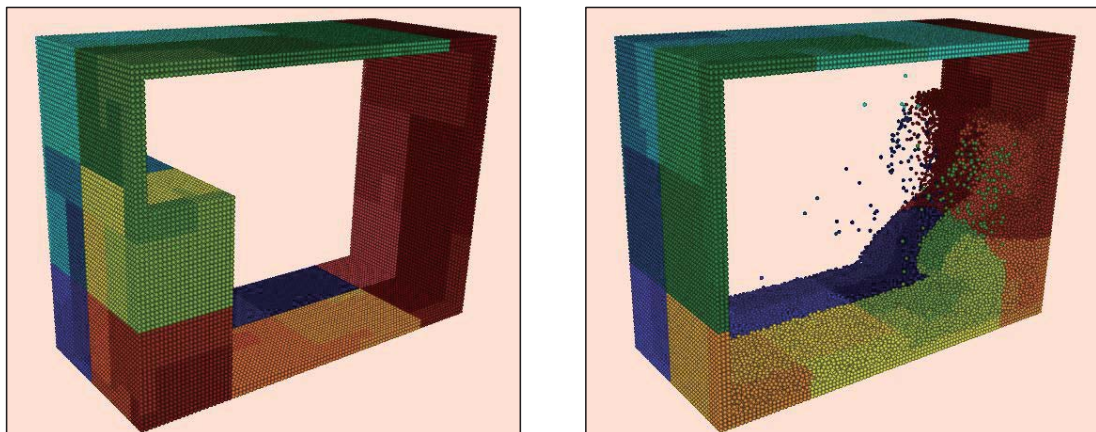


図 4.39 3次元ヒルベルト曲線, モートン曲線, ペアノ曲線による3次元領域分割を改良型SPH法の3次元ダムブレイク計算に適用した結果

4.4 流体構造連成問題への適用

動的な領域分割により各小領域内の粒子数を一定に保つ負荷分散を複数 GPU 計算で導入する方法を説明してきた。これにより大規模な流体シミュレーションを GPU スパコンを用いて効率的に実行できることも第 4.3.3 節までに示された。しかし、現実において我々が直面する流体现象の多くは構造物(浮遊物や建物など)との連成を含んでいる。第 2.2.3 節で述べたように粒子法の流体構造連成計算では流体、壁、物体のすべてを粒子により表現するため解像度(=粒子数)の不足が深刻な問題になる。例えば、一つあたり 1,000 個の粒子から構成される物体を計算領域に 1,000 個配置するとした場合、それだけで 100 万個の粒子が必要になる。物体を構成する粒子と同サイズの粒子で計算領域を構成してその中に流体を満たすことを考えるとすぐに粒子数の問題に突き当たる。物体構成粒子の大きさと流体粒子の大きさの異なる粒子を用いた高精度な計算スキームは確立されているとは言えず、現状では本研究のように均等なサイズの粒子を用いた大規模粒子法シミュレーションが唯一の手段である。

本研究では、さらなる発展段階の取り組みとして粒子法の流体構造連成計算にも提案する GPU 間の動的負荷分散手法を適用できるようにする。このような取り組みは先攻研究の例が希であるため、まずは第一段階として現在の流体シミュレーションに用いた動的負荷分散手法を、流体粒子と多数の物体が存在する流体構造連成計算にもそのまま適用することにする。すなわち、流体粒子、物体構成粒子、壁粒子の区別をせず粒子数のみで均等化して領域分割を行う。一つあたり 100 個の粒子から構成される直方体の浮遊物体を 100 個含んだ全部で 36 万個粒子を用いた流体構造連成の検証計算の様子を図 4.40 に例として示した。このように流体粒子も壁粒子も、物体構成粒子も区別なく動的負荷分散する(見やすさのために物体は色分けして表示しているが負荷分散の対象である)。計算手順の詳細を以下に説明する。

流体構造連成の複数 GPU 計算の手順を図 4.41 に示す。ここで、D2H は GPU から CPU にデータをコピーすることを表す。H2D はその逆であり CPU から GPU にデータをコピーすることを表している。

図 4.41 において「物体同士の衝突計算」より手前については、これまでの改良型 SPH 法による流体計算の手順と同じである。「物体同士の衝突計算」では異なる物体の物体構成粒子間の衝突について DEM のモデルに従い GPU 上で計算する。その後、各物体の運動方程式の時間発展(緑色)を行う。物体の運動方程式の時間発展について、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ため物体の時間積分で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(なお、物体の変形計算の導入を予定

した経緯から重心についての総和計算も力とトルクの総和計算より前に行っているが、変形計算は扱っておらず、本研究で扱う物体はすべて剛体であることを述べておく)。これらの総和計算の手順は次の通りである。まず、自身の小領域内の GPU 上で総和計算を行う。これは第 3.2.5 節の実装 (A) により行う。すなわち、すべての物体について GPU 上で並列にスレッドを走らせ、各スレッドが担当する物体についての総和計算を行う。GPU 上での総和計算が完了したあと、各物体の計算結果を CPU 側に物体番号順にコピーする。CPU 側へのコピーが完了した段階では小領域内 (ノード内) の総和計算 (=ノード内リダクション) が完了しただけであるから、各物体について `MPI_Gather` により各小領域の総和計算の結果をルートとなる小領域に送信し、ルートの CPU 上で受信した値を足し合わせて総和計算を完了させ (ノード間リダクション)、それをもとに CPU 上で角運動量や並進運動量、クォータニオンなどの物体の物理量を更新する。その結果を `MPI_Bcast` を用いて各小領域にブロードキャストする。ルートへの転送とルートからのブロードキャストは各物体毎に順次行う。

物体の運動方程式の時間発展のうち、物体同士の衝突計算や、クォータニオン等の物体の物理量の計算コストは流体計算のコストに比べて十分に小さく、動的負荷分散は前述のように流体と剛体の合計の計算コストとはせずに粒子数を均一化する方法を用いている。一方、物体の運動方程式の時間発展のためにルートにデータをリダクションさせる 1 対多型の MPI による集団通信を行うようになっており、これが並列化効率を低下させる。物

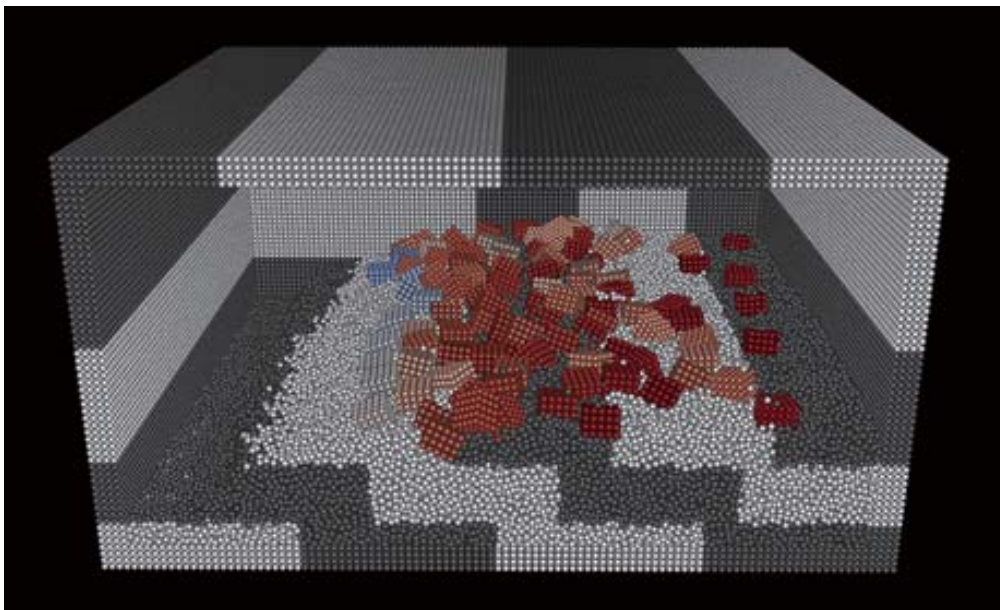


図 4.40 サスペンション・フローの流体構造連成問題の複数 GPU 計算に対する 2 次元のスライスグリッド法を用いた動的負荷分散

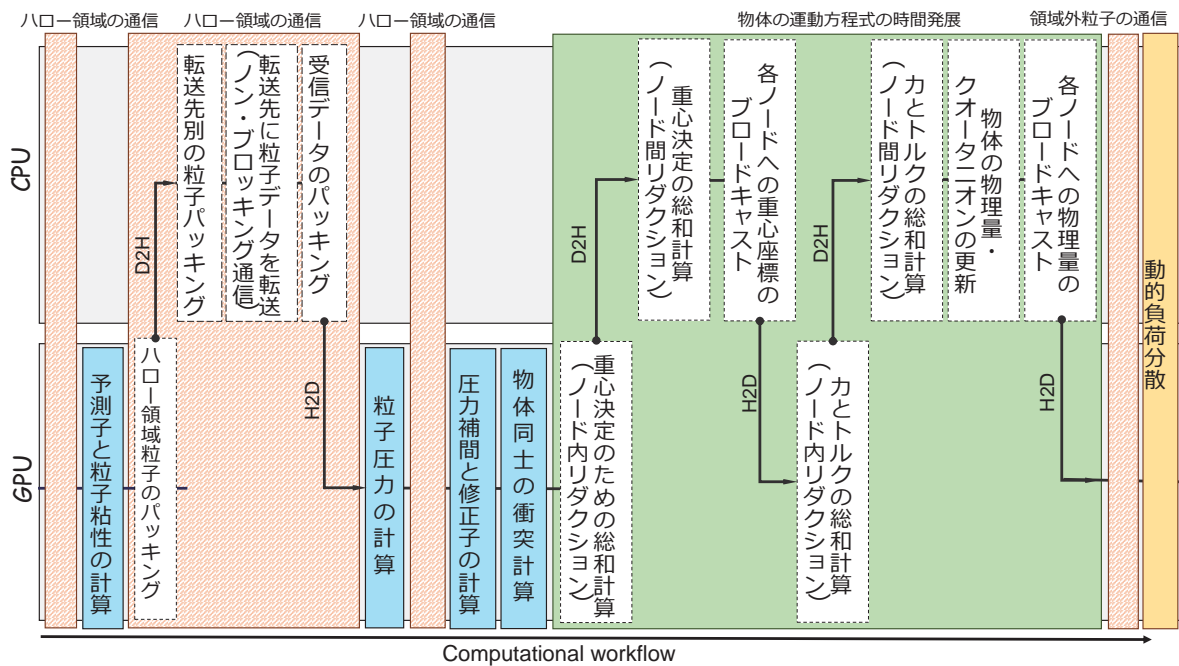


図 4.41 流体構造連成の複数 GPU 計算の手順

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り容易ではないため、本研究では行わない。まずは以上のような流体構造連成計算の実装を GPU スパコン上で行った場合に、どの程度の計算コストになるのかについて調べる。次の第 4.4.1 節では、これまで達成されていないサスペンション・フローの大規模な流体構造連成計算を GPU スパコンを用いて実現し、その並列化効率について弱スケーリングを実施して議論する。

4.4.1 大規模サスペンション・フロー計算の実現と弱スケーリング

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 $144\text{ m} \times 160\text{ m} \times 60\text{ m}$ 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。図 4.42 に初期配置の様子を示す。流体の挙動を計算するために合計で $8,743$ 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 $9\text{ m} \times 10\text{ m} \times 10\text{ m}$ 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 4.45 は時刻 9.75 sec までのスナップショットを示している。また、図 4.46 は時刻 6.8 sec のときのスナップショット (図 4.45 に示した 8 枚のうち、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 13,600 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。

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 4.43 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 4 GPU 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 4 GPU の流体の時間発展

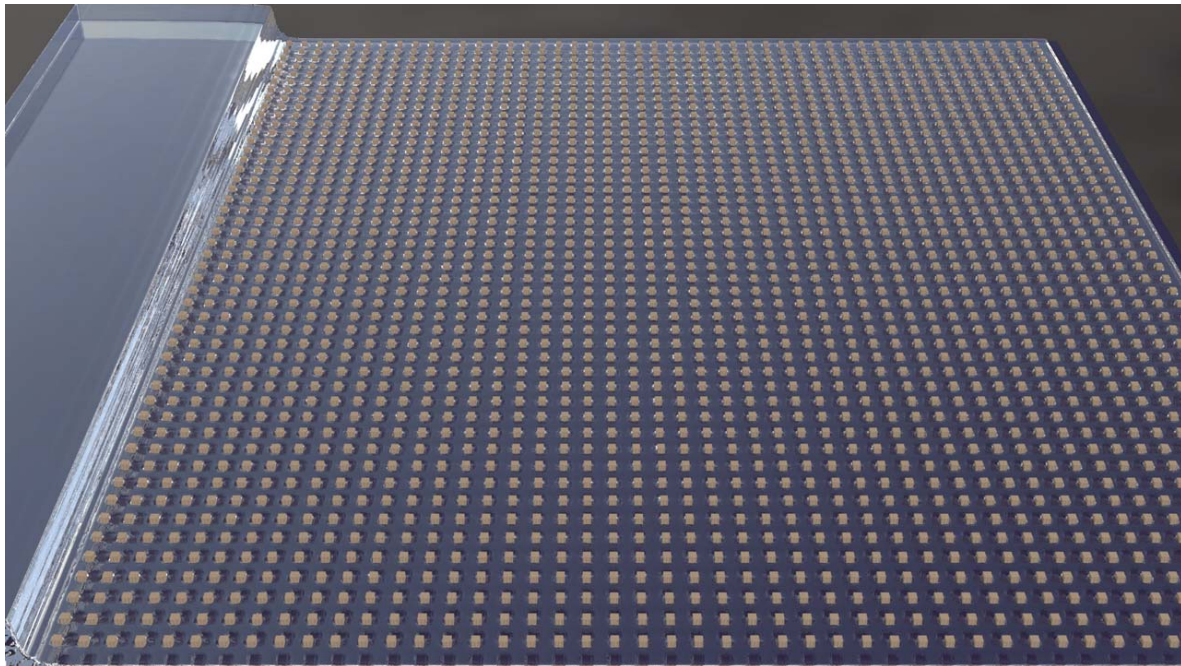


図 4.42 大規模サスペンションフロー計算の初期配置

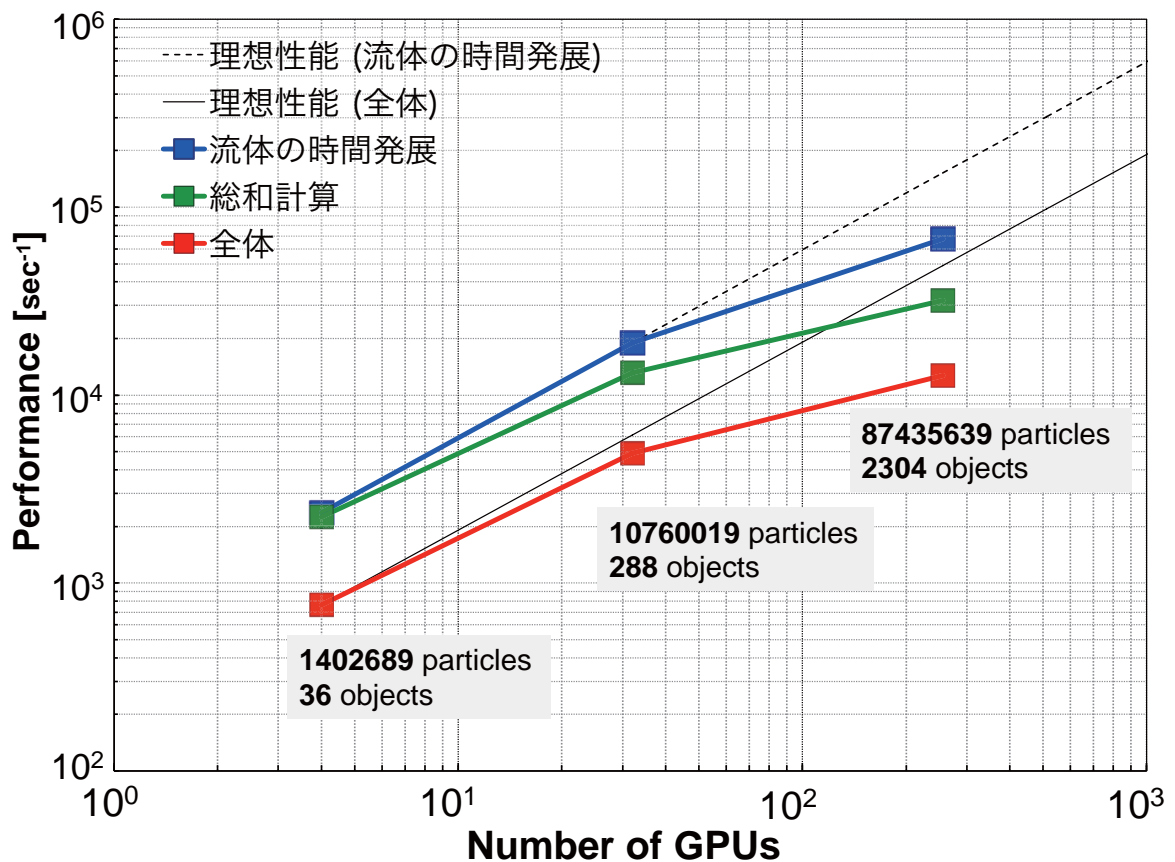


図 4.43 TSUBAME 2.5 における弱スケーリングによる性能評価

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、32 GPU と 256 GPU 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、32 GPU と 256 GPU 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 4.43 における 1 回の Reduction 操作に要する計算時間の内訳を図 4.44 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。

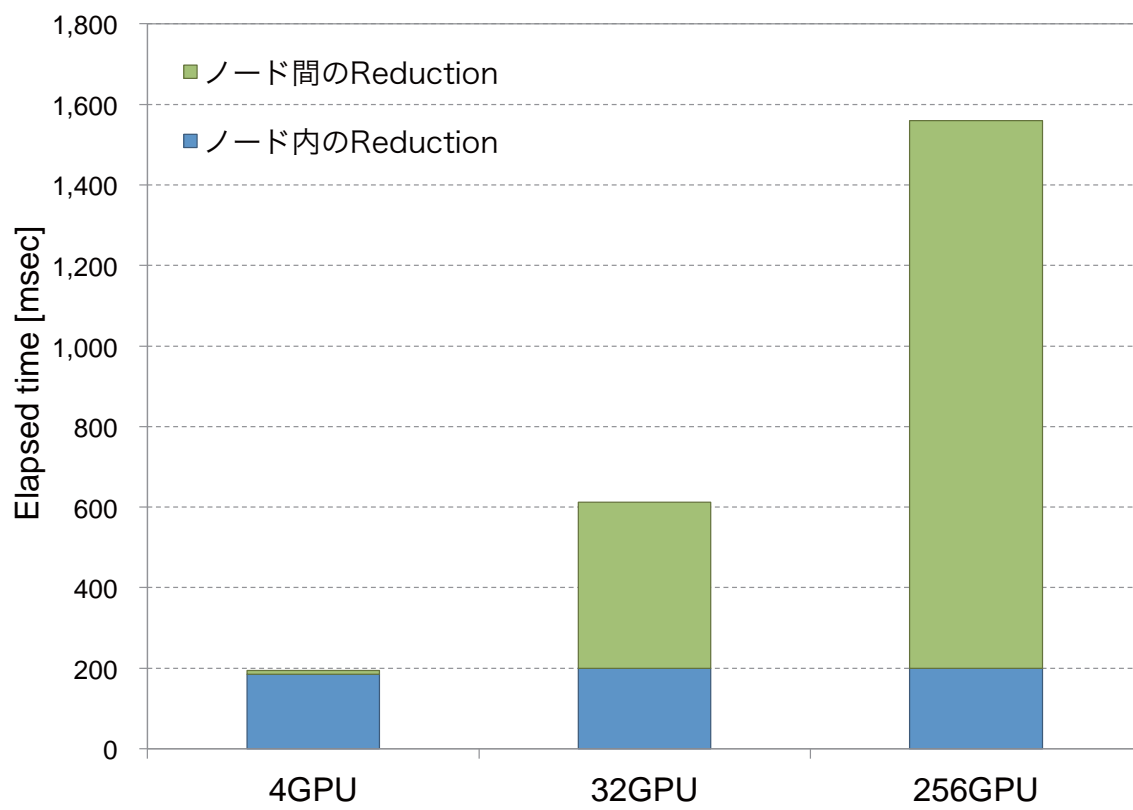


図 4.44 図 4.43 における Reduction の内訳

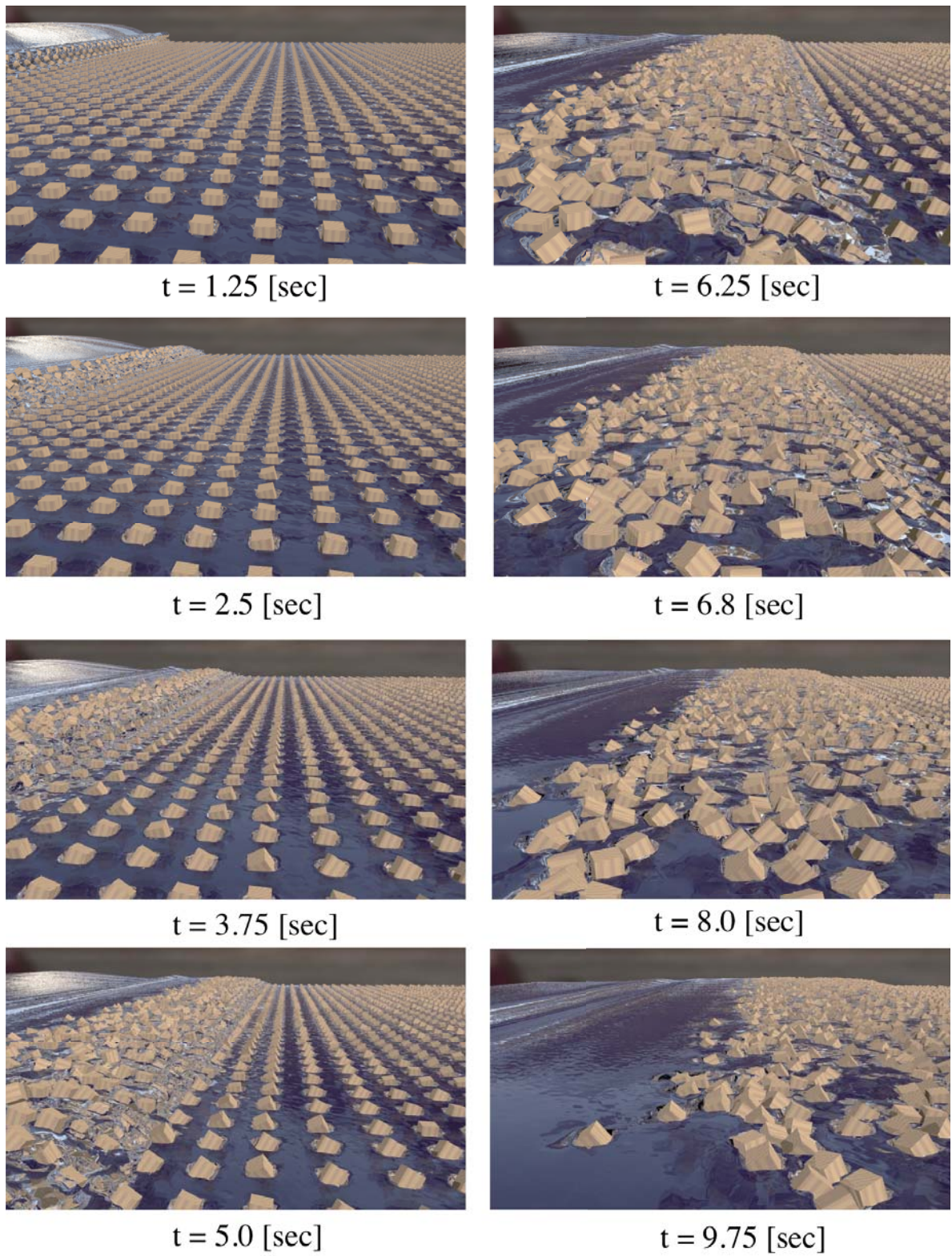


図 4.45 2,304 個の物体を含む 8,743 万個の粒子を用いた大規模サスペンションフロー計算を 256 GPU を用いて計算した結果

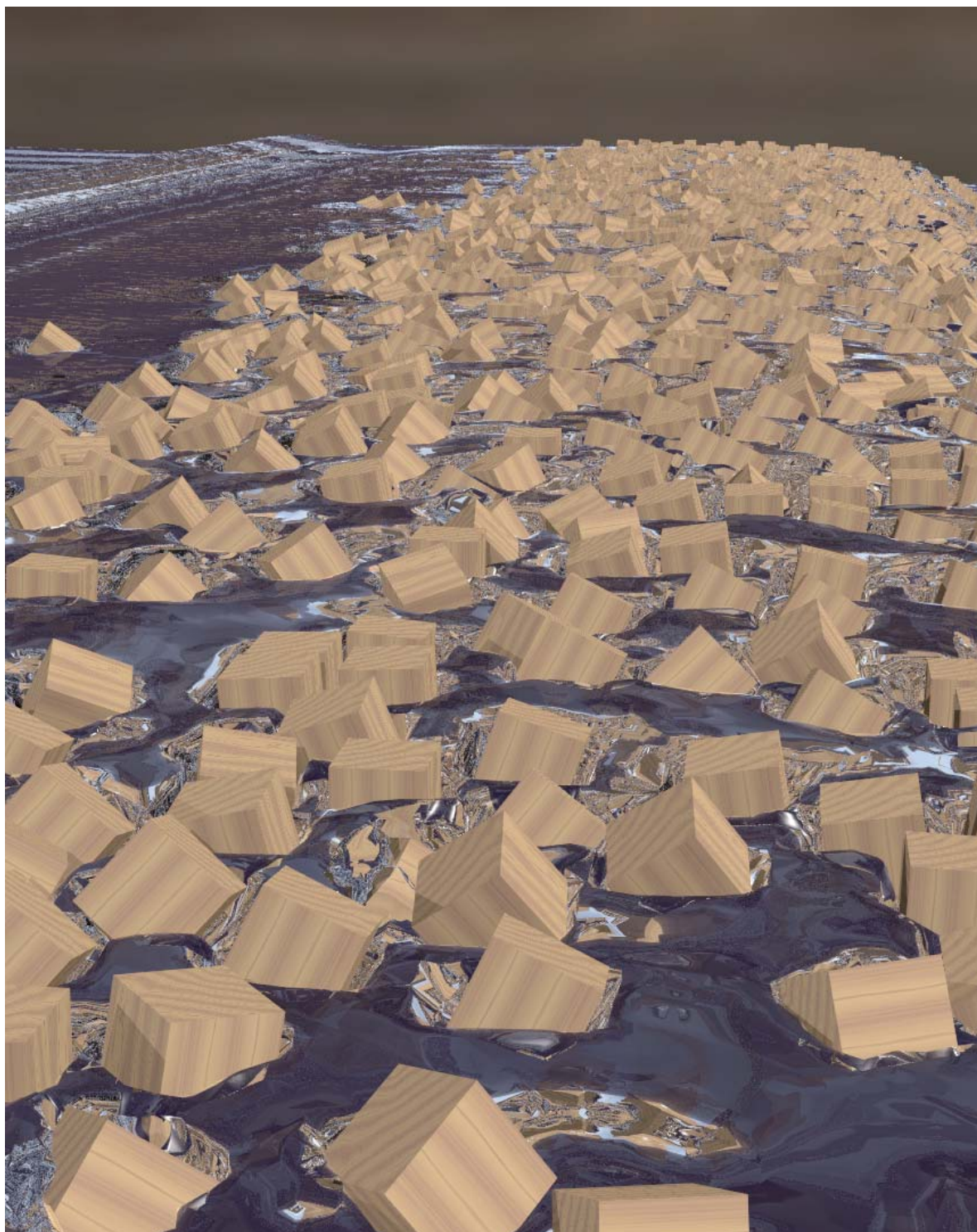


図 4.46 2,304 個の物体を含む 8,743 万個の粒子を用いた大規模サスペンションフロー計算を 256 GPU を用いて計算した結果 (時刻 6.8 sec における拡大図)

第 5 章

複数 GPU による大規模粒子法シミュレーションの実問題への適用

5.1 実行環境

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 Intel Xeon X5670 が 2 基、GPU (NVIDIA Kepler K20X) が 3 枚装着され、全部で 4,224 枚の GPU が搭載されている。単精度計算で 17 ペタフロップス、倍精度で 5.7 ペタフロップスの理論演算性能値となっている。全体で 1,408 ノードから構成されており、ノード間は QDR InfiniBand により Fat Tree 型で接続され、合計で最大 80 Gbps の通信性能を有している。TSUBAME 2.5 に搭載された複数 GPU を用いて、粒子法による実問題の大規模シミュレーションを実現する。

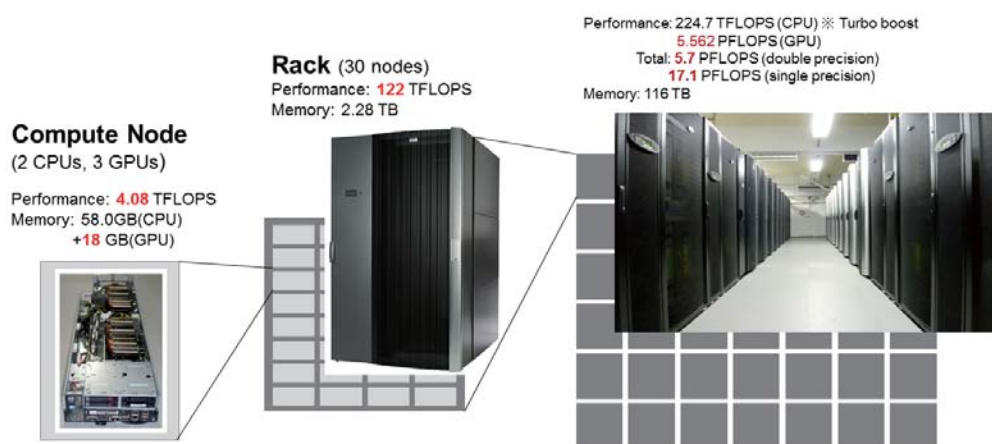


図 5.1 TSUBAME2.5 のシステム構成

5.2 ゴルフ・バンカーショットの解析

ゴルフのバンカーショットはサンドウェッジのスイングによる砂のかき上げと、かき上げられた砂によるゴルフボールへの運動伝達を含む複雑な問題である。解析手法の DEM (Discrete Element Method) は計算コストが高いため、これまでは 10 万個程度の粒子による 2 次元計算にとどまっている [72]。実際の砂と同程度のサイズの粒子を 1,670 万個使用した 3 次元の大規模バンカーショット・シミュレーションを実現する。

バンカー砂に含まれる粗砂 (粒径 0.5 mm~1.0 mm) を想定して粒子半径を 0.4 mm とし、計算条件を表 5.1 のように設定する。ヤング率とポアソン比を用いて Hertz の弾性接

表 5.1 バンカーショットの計算条件

時間刻み幅	5.0×10^{-6} [s]
粒子半径	0.4 [mm]
粒子質量	5.09×10^{-7} [kg]
ヤング率	2.8 [GPa]
ポアソン比	0.17
摩擦係数	0.3
粒子数	1.67×10^7
ステップ数	104,000

触理論から法線方向の弾性定数を決定し、ラメの定義式を用いてせん断方向の弾性定数を決定する。跳ね返りが最も小さくなるよう第 2 章で述べた Cundall らの提唱する式 (2.9)~式 (2.10) に従い粘性係数を決定する。また、サンドウェッジの軌道は回転軌道を想定し図 5.2 のように決定する。

「目玉」の初期状態 (ゴルフボールがバンカーに埋もれた状態) を 64,000 ステップかけて生成した後、クラブヘッドの先端の最大速度を 5.0 m/s としてスイングを開始した。ボールの速度と軌道の変化をそれぞれ図 5.3 と図 5.4 に、スイング開始から 0.2 秒後までのスナップショットを図 5.5 に示す。砂粒子から力の伝達を受けたゴルフボールは放物運動となるため、図 5.4 の軌跡に対して式 (5.1) による最小 2 乗法によるフィッティングを行い、地面に対するボールの飛び出し角 $\theta=50^\circ$ 、初速度 $v_0=2.3$ m/s を得た。サンドウェッジの打ち出し速度の半分程度の速度がゴルフボールに与えられたとわかった。

$$y = -\frac{g}{2v_0^2 \cos^2 \theta} x^2 + \tan \theta x \quad (5.1)$$

表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを指数的に減速させて計算

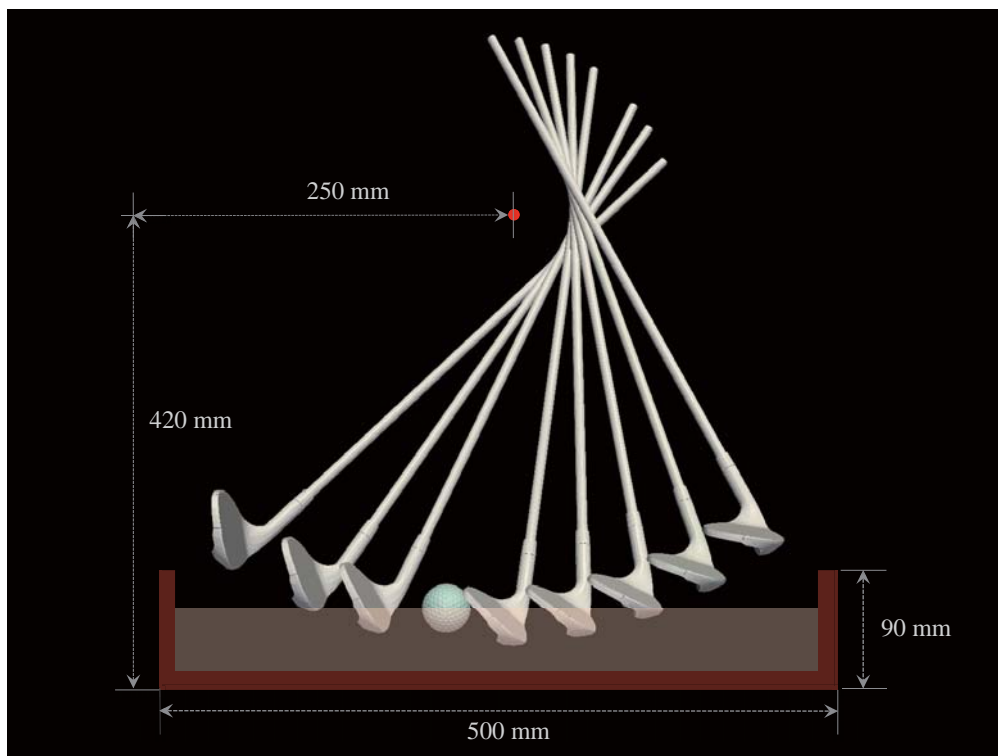


図 5.2 計算配置とサンドウェッジの軌道

時間の半分である 0.1 秒後に停止させて計算した場合のボールの速度と軌道の変化をそれぞれ図 5.6 と図 5.7 に、スイング開始から 0.2 秒後までのスナップショットを図 5.8 に示す。図 5.7 の軌跡に対して式 (5.1) による最小 2 乗法によるフィッティングを行い、地面に対するボールの飛び出し角 $\theta = 49.3^\circ$ 、初速度 $v_0 = 2.19 \text{ m/s}$ を得た。計算条件 5.1 と比較してサンドウェッジによる掻き上げが行われなくなったことにより、ボールがより前方に打ち出されていることが確認できる。

表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを指数的に減速させて計算時間の半分である 0.1 秒後に停止させ、さらにサンドウェッジの初期座標を 10.0 mm だけ垂直方向に低く設定した場合の、ボールの速度と軌道の変化をそれぞれ図 5.9 と図 5.10 に、スイング開始から 0.2 秒後までのスナップショットを図 5.11 に示す。図 5.10 の軌跡に対して式 (5.1) による最小 2 乗法によるフィッティングを行い、地面に対するボールの飛び出し角 $\theta = 55.75^\circ$ 、初速度 $v_0 = 1.67 \text{ m/s}$ を得た。図 5.11 の計算ではサンドウェッジがバンカーに深く入り過ぎたために、砂を介したボールへの力の伝達が十分に行われずスイングに失敗し、ミス・ショットとなっていることが確認できる。64 GPU を用いて計算した場合、データ出力なども含めた全計算時間は 78.9 時間であった。

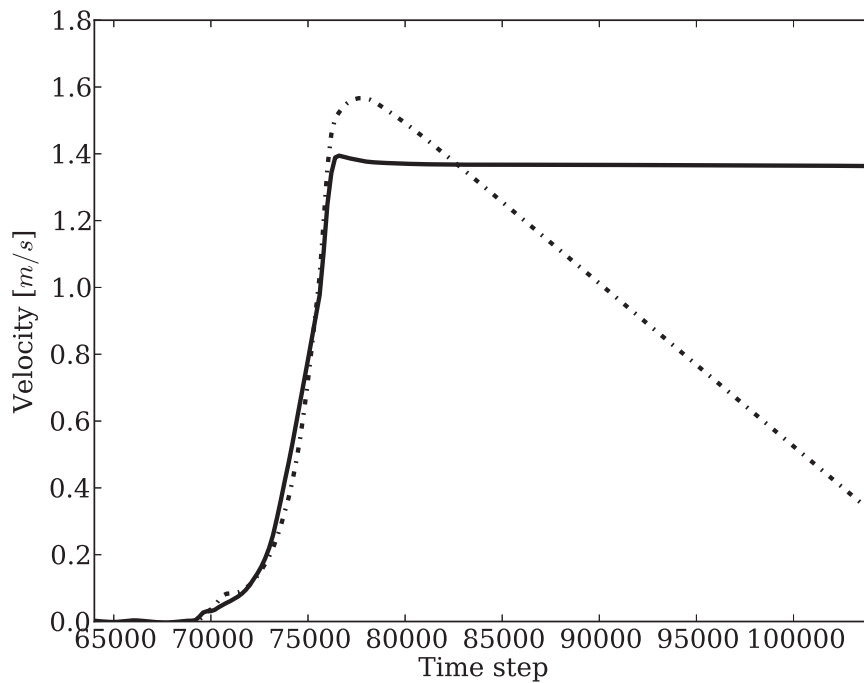


図 5.3 垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化

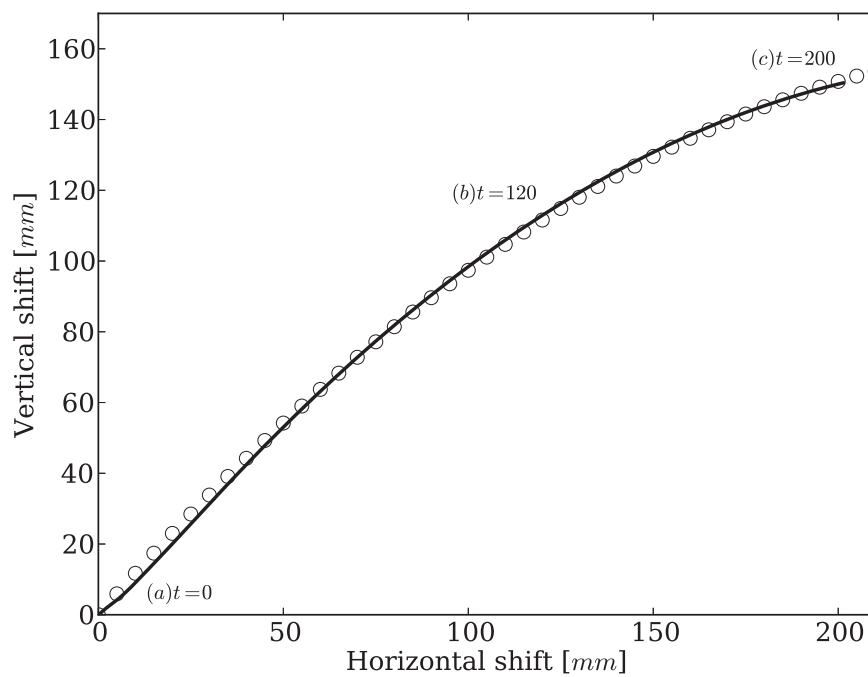


図 5.4 ゴルフボールの軌道



図 5.5 64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算

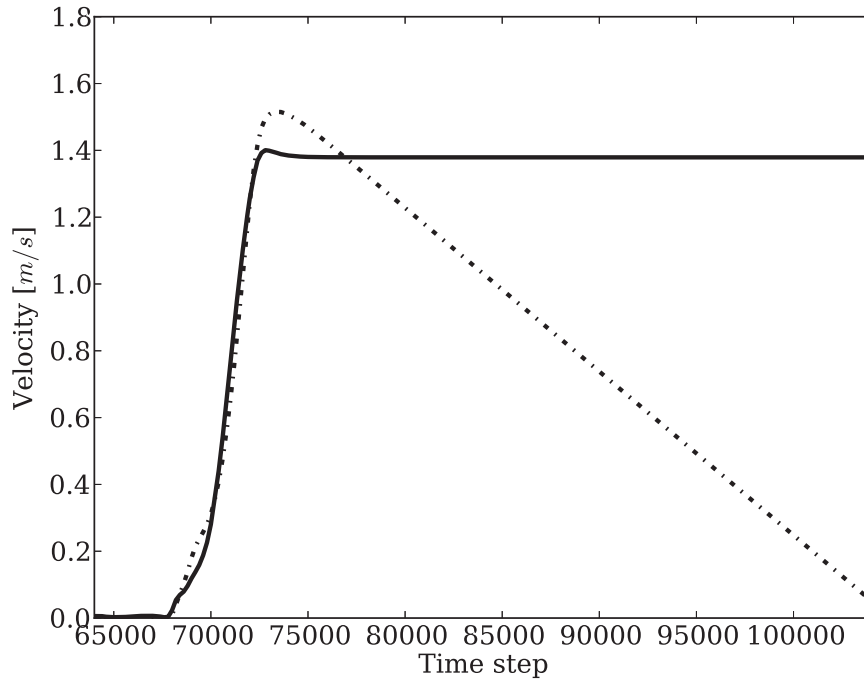


図 5.6 表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを 0.1 秒後に停止させて計算した場合の垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化

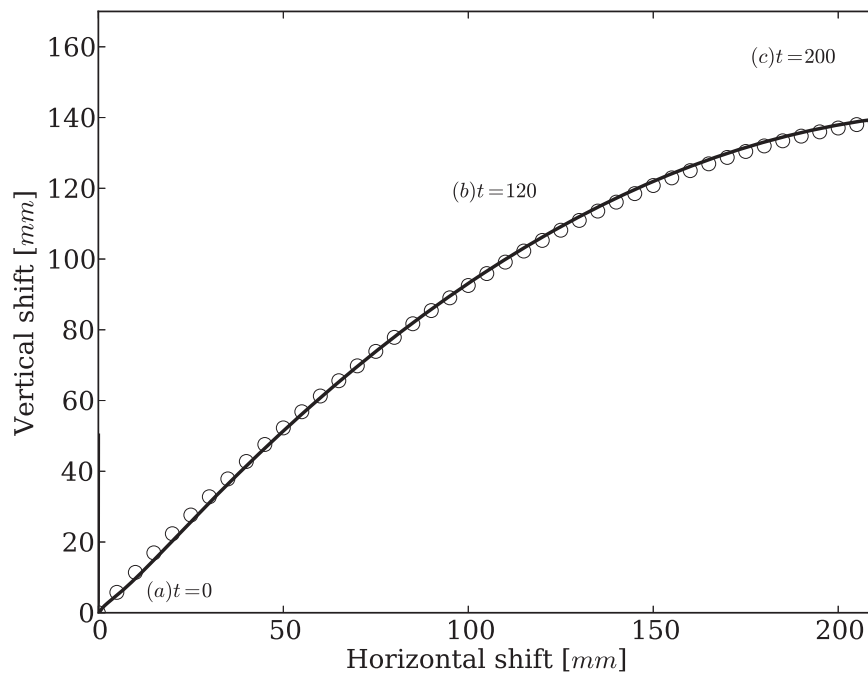


図 5.7 表 5.1 と同一の計算条件のもと、サンドウェッジのスイングを 0.1 秒後に停止させて計算した場合のゴルフボールの軌道

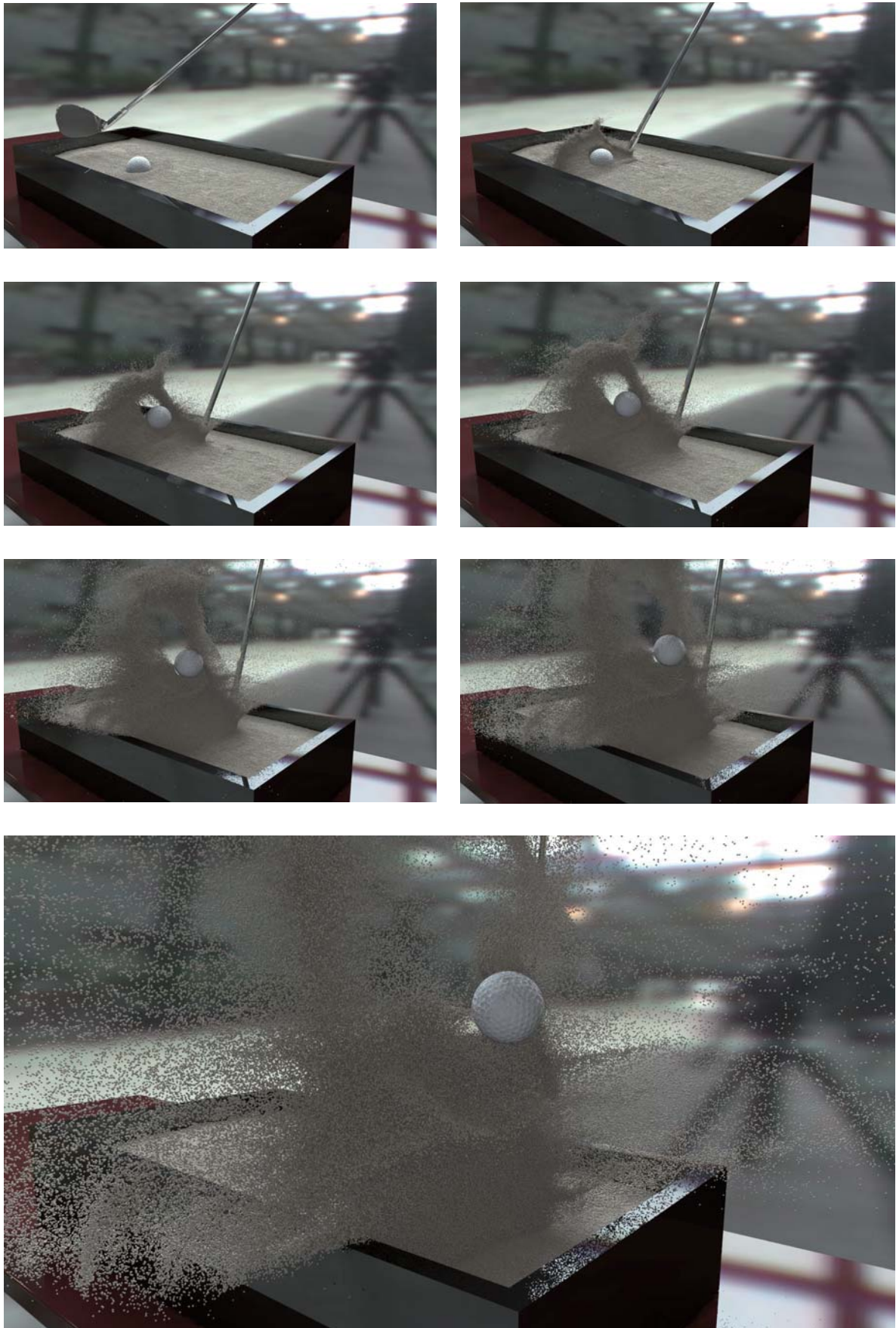


図 5.8 64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (表 5.1 と同一の計算条件のもと、サンドウェッジを 0.1 秒後に停止させた場合)

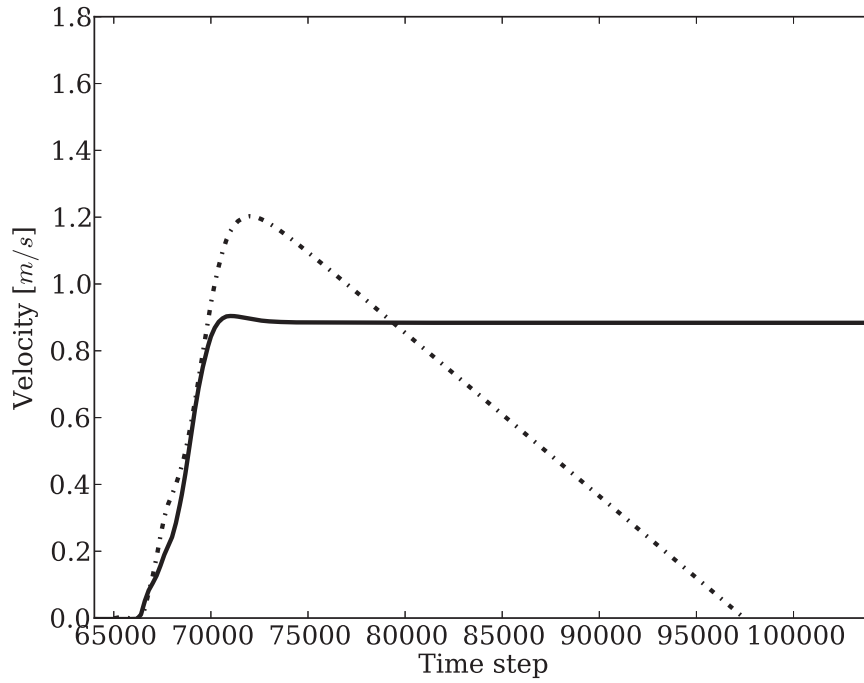


図 5.9 図 5.8 において、初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置して計算した場合の、垂直 (点線) と水平 (実線) 方向のゴルフボールの速度変化

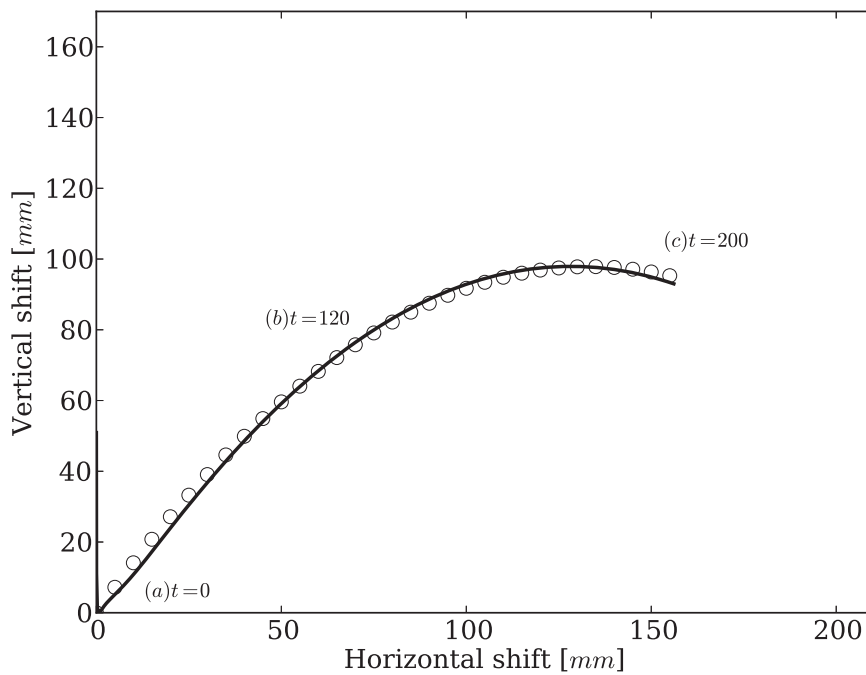


図 5.10 図 5.8 において、初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置して計算した場合のゴルフボールの軌道

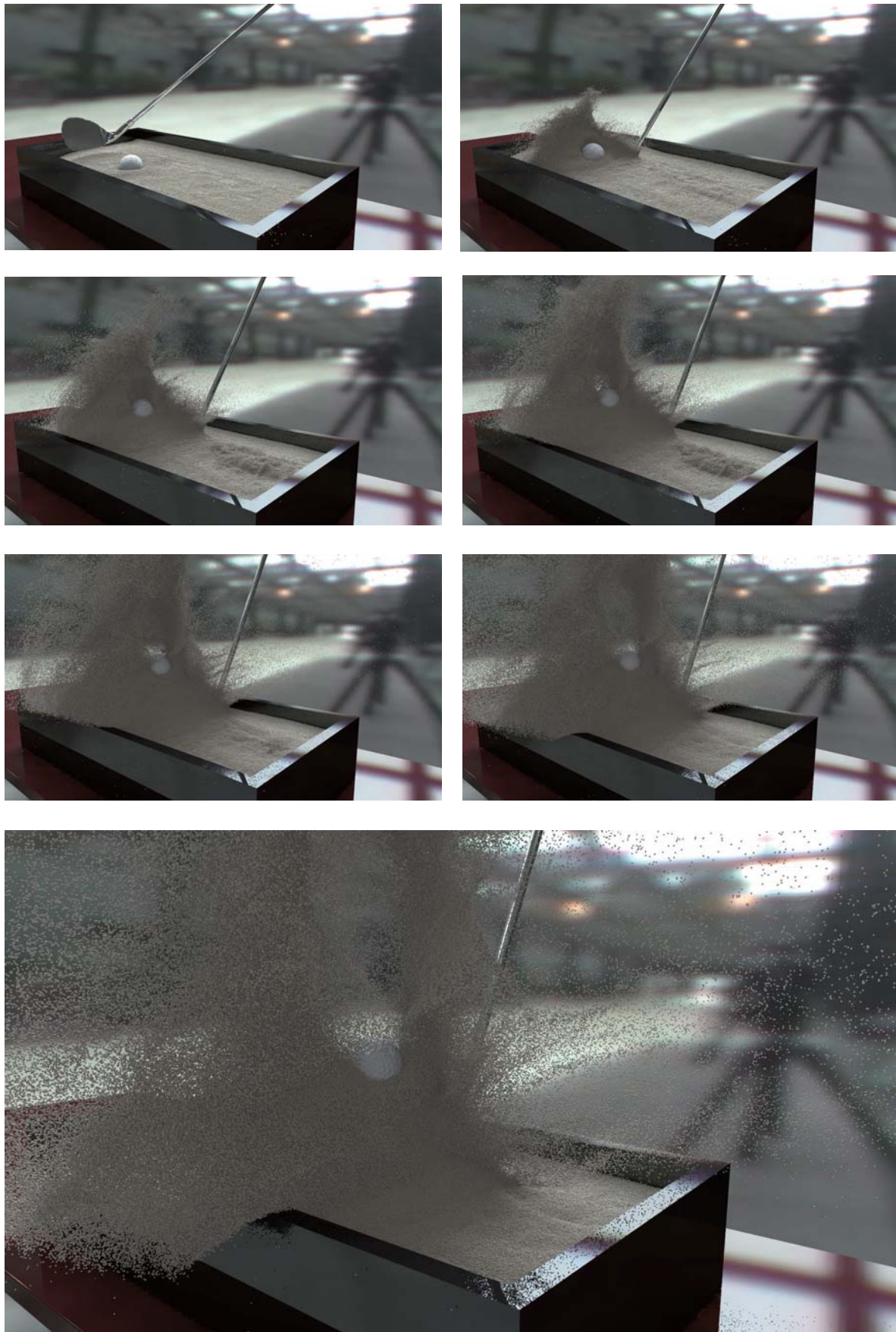


図 5.11 64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (図 5.8 において, 初期にサンドウェッジを鉛直方向に 10.0 mm 低く配置した場合)

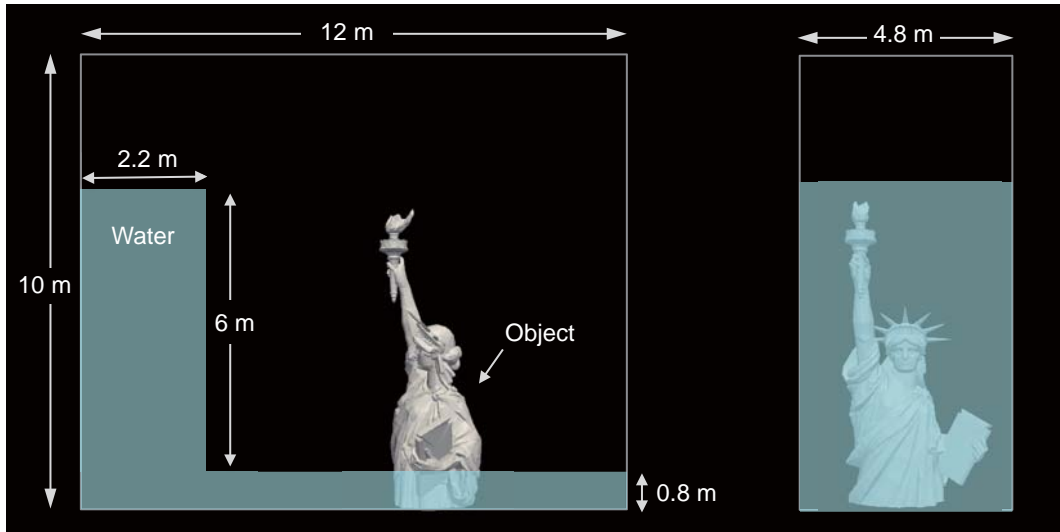


図 5.12 自由の女神を含むダムブレイク計算の配置図

5.3 複雑形状を含むダム崩壊シミュレーション

自由の女神像の CAD データから粒子群による複雑形状の構造物の粒子群データを作成し、図 5.12 のように計算領域に配置する。流体粒子 660 万個、壁粒子 340 万個の合わせて 1,000 万個の粒子を計算に使用し、水柱のダムが崩壊し始めて沈静化するまでの約 3.62 秒を、17,400 ステップをかけて 32 GPU を用いて計算した。計算条件を表 5.2 に、動的負荷分散の様子を図 5.15 に、圧力分布を図 5.14 に示す。良好な圧力分布が確認できる。

表 5.2 計算条件

g	$[m/s^2]$	9.8
ν	$[m^2/s]$	1.0×10^{-6}
ρ	$[kg/m^3]$	1.0×10^3
v_{max}	$[m/s]$	12
l_0	$[m]$	0.025
C_h		2.6
γ		2
C_T		1.0
C_M	$[m/s]$	2/15

表 5.2 と同一計算条件のもと、各方向に粒子数を 2 倍に増やして解像度を 8 倍にし、全

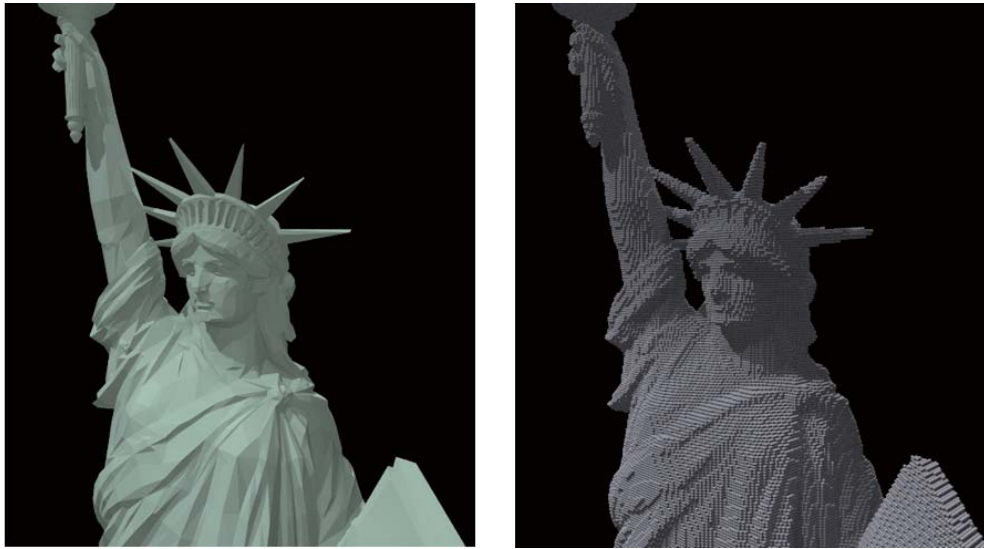


図 5.13 自由の女神の CAD データ (左) と CAD データから生成した粒子群データ (右)

体で 7,250 万粒子 (流体粒子は 5,370 万個) を用いた超高解像度のダム崩壊計算を 80 台の GPU を用いて実現した。計算結果を図 5.16 に示す。また、この時の自由の女神の粒子群データも図 5.13 に示す。高解像度になることで粒子により物体を表現する方法でも複雑形状の細部まで再現できていることが確認できる。

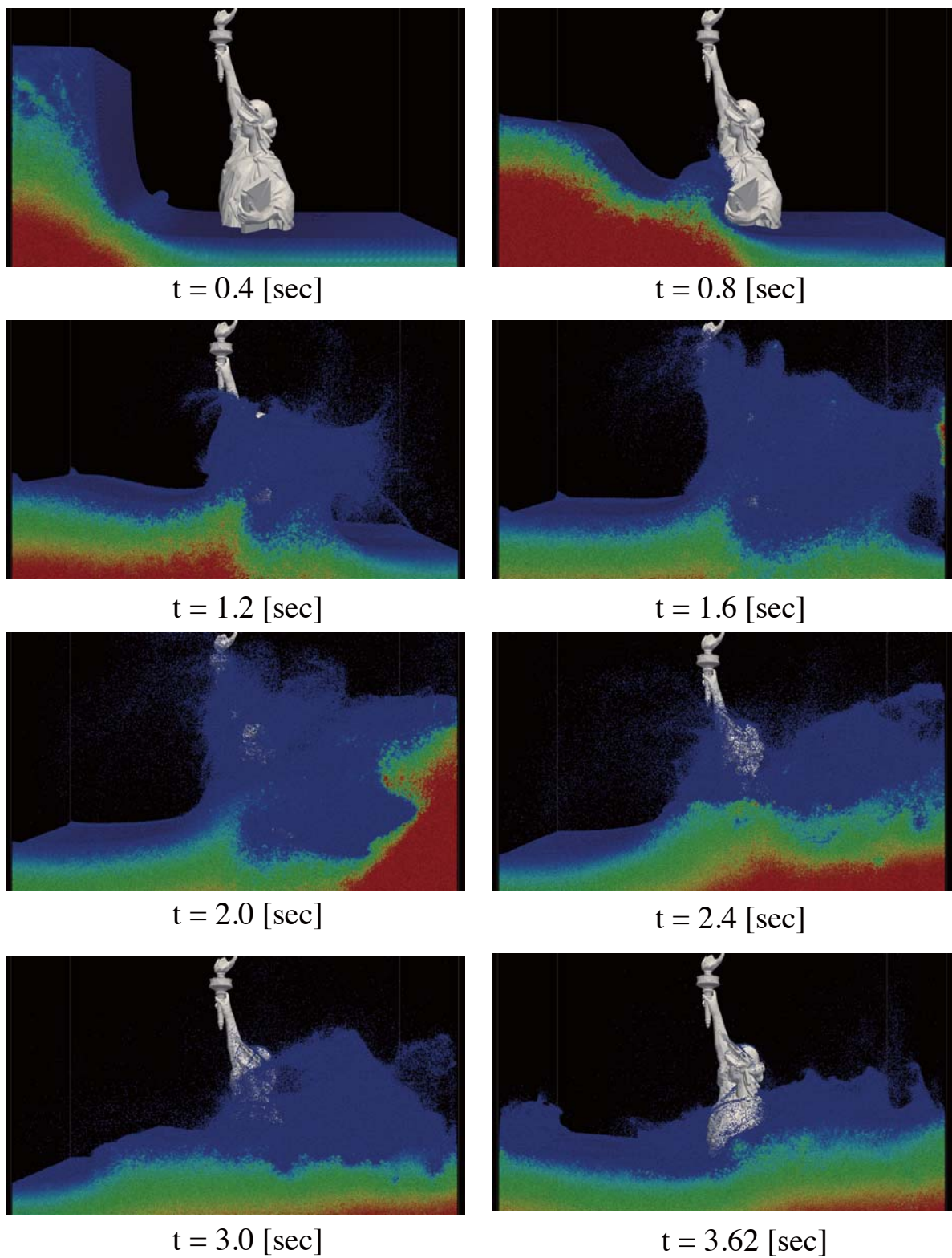
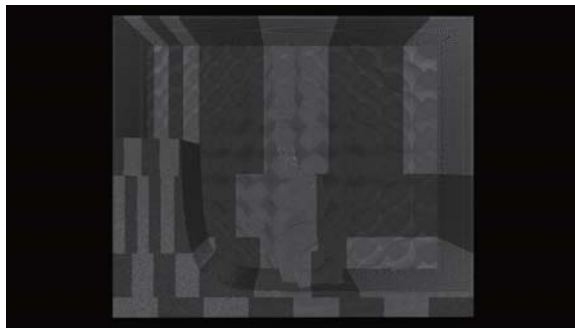
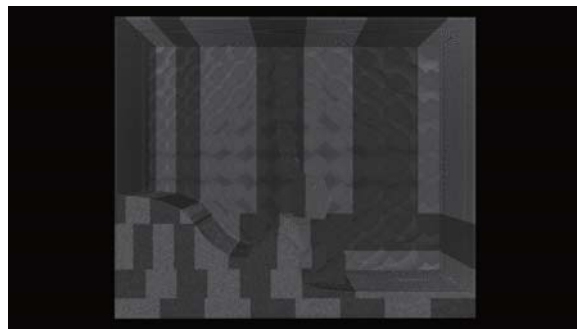


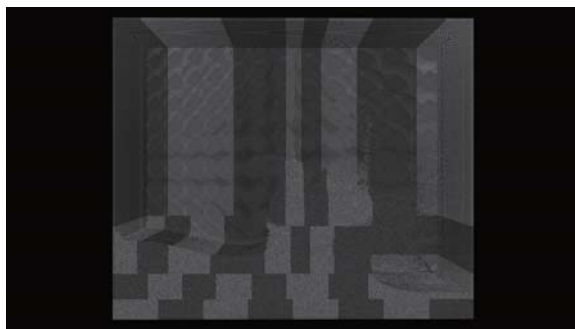
図 5.14 1,000 万粒子を用いた自由の女神を含むダムブレイク計算における圧力分布の変化



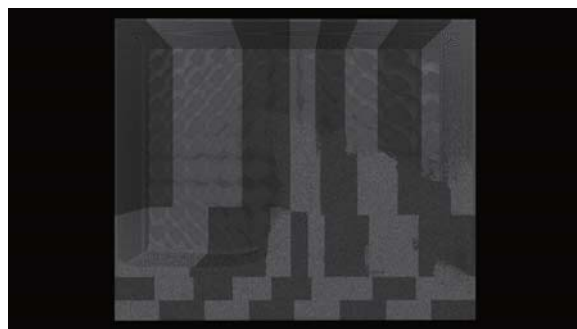
t = 0.4 [sec]



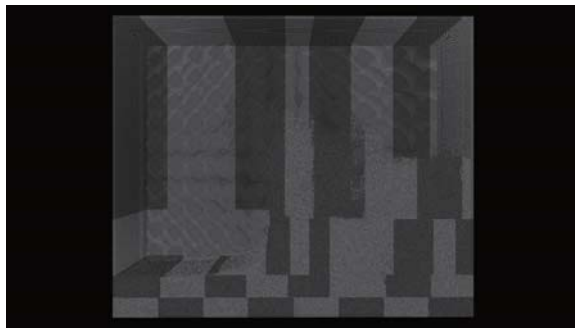
t = 0.8 [sec]



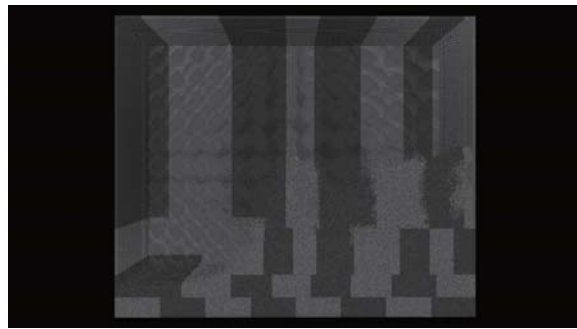
t = 1.2 [sec]



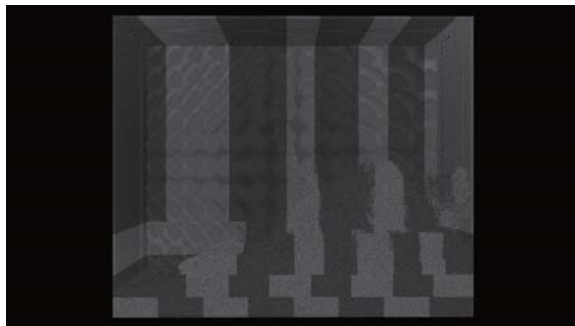
t = 1.6 [sec]



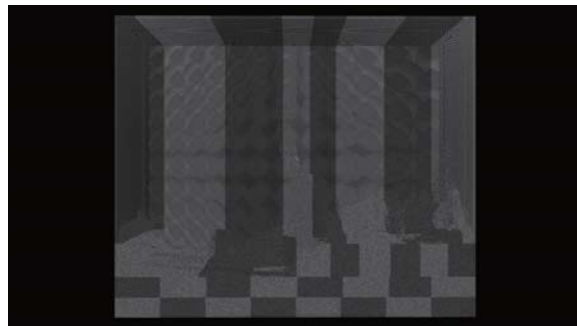
t = 2.0 [sec]



t = 2.4 [sec]



t = 3.0 [sec]



t = 3.62 [sec]

図 5.15 図 5.14 の計算における動的負荷分散の様子

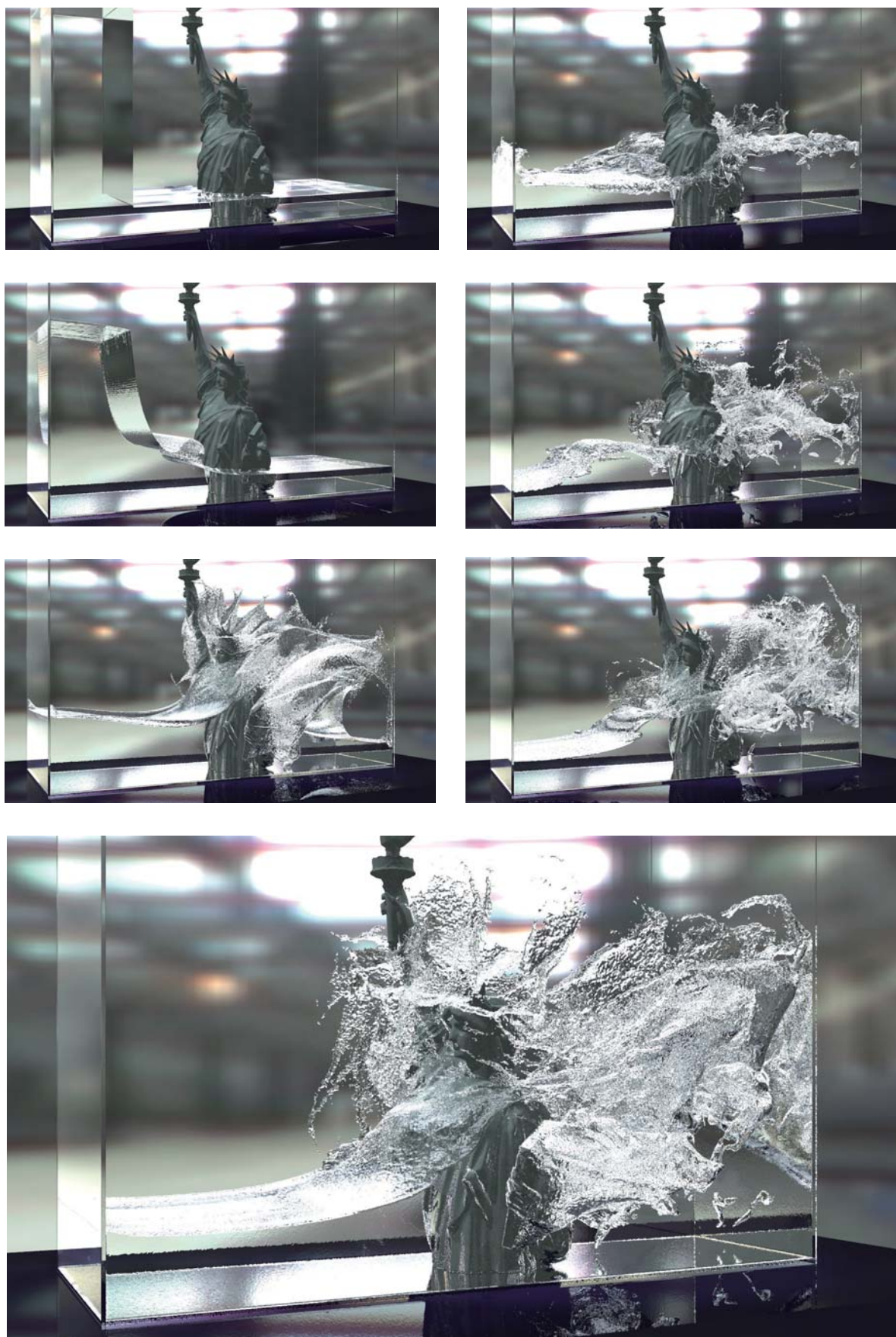


図 5.16 7,250 万粒子による大規模ダムブレイク計算

5.4 建物と流木を含む津波シミュレーション

計算領域の縦横及び高さを $64\text{ m} \times 80\text{ m} \times 20\text{ m}$ とし、深さ 1.4 m の静止した水を張り、そこに合計 945 個の浮遊する物体を配置する。図 5.18 に示すように 4 種類の異なる形状の流木の内部を $124 \sim 472$ 個の物体構成粒子で充填して流木を表現する。それぞれの流木の個数と構成要素数の詳細を図 5.3 に示す。物体を計算開始直後に水面に落下させて不規則な流木の初期配置を生成する。256 台の GPU を用い、水柱の高さは 9 m に設定する。計算条件を表 5.5 に、配置図を図 5.17 に示す。

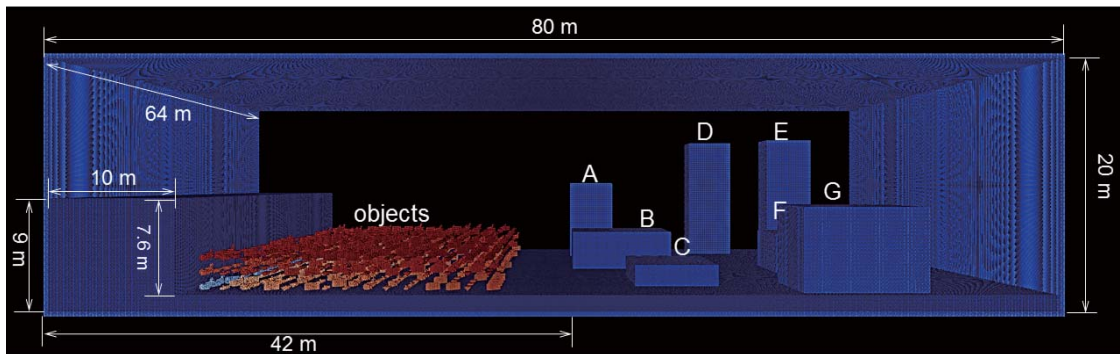


図 5.17 建物を含んだ津波シミュレーションの配置図 (側面から見た図)

計算領域内には図 5.19 のように異なる形状をした 7 棟の建物 A~G を配置する。表 5.5 に各建物のサイズと位置を示す。256 台の GPU を用いて 46.9 時間 (ファイル IO 時間含む) かけて計算した物理時間 13.1 秒後までのスナップショットを図 5.20 に示す。複雑な流木の建物の間の挙動を詳細に確認することができる。

各建物の津波の進行方向に面した箇所 (図 5.19 中の黄色で印をつけられた側面) の底辺から 1.4 m の高さに含まれる領域の壁表面に圧力センサーを設置し、領域内で測定された

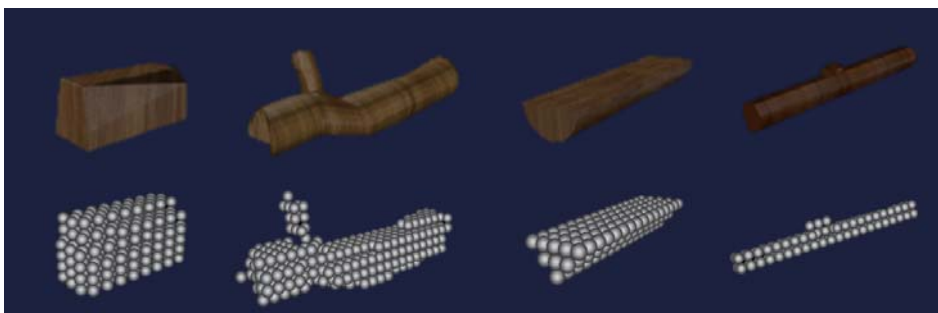


図 5.18 4 種類の流木の CAD データ (上) と、生成された粒子群データ (下)

表 5.3 図 5.18 における各物体の個数とそれぞれの構成粒子数

物体番号	1 つあたりの構成粒子数	物体の個数	物体構成粒子数の合計
0	472	241	113,752
1	124	258	31,992
2	309	211	65,199
3	144	235	33,840
合計:		945	244,783

表 5.4 計算条件

計算領域	64 m × 80 m × 20 m
静水域の深さ	1.4 [m]
初期水柱の高さ	9.0 [m]
粒子数	20,622,493
物体数	945
物体の種類	4
ステップ数	26,300
物理時間	13.1 [s]
GPU s	256
計算時間	46.9 [h]

圧力値の平均値を各時刻でプロットした結果を図 5.21 に示す。一番上段の左側には建物 A~G の測定結果を重ねて表示している。

実際の津波の衝突順序と同様に、進行方向に対して手前の 3 棟 (A~C) が、後方の 4 棟 (D~G) よりも先に津波が衝突して圧力値のピークが発生していた。また、手前の建物 B、建物 C に津波が阻まれ、建物 B、建物 C の背後の隠れた位置にある建物 F の圧力値は、建物 A~G の中で最も低くなるなど、実現象を良く再現できている。

津波の波力を正面から受ける建物 B では、A~G の全ての建物の中で圧力ピークが最も大きく、約 62 kN/m^3 (62 kPa) であった。衝突時、及び衝突前後 50 msec における建物 B の衝突断面の圧力分布の様子を図 5.22 に示す。白線囲み部分は図 5.21 の測定において圧力センサーを設置した部分である。この領域の圧力値は前述のように最大で 62 kPa 程度であった。しかし、実際には図 5.22 の中央の図 (時刻 4.05 秒経過後) における左上の赤い部分のように、物体が建物に直接接触することによりセンサー領域で測定された圧力よりも大きな圧力が建物の壁面上で発生していることが確認できる。図 5.22 の中央の図の赤

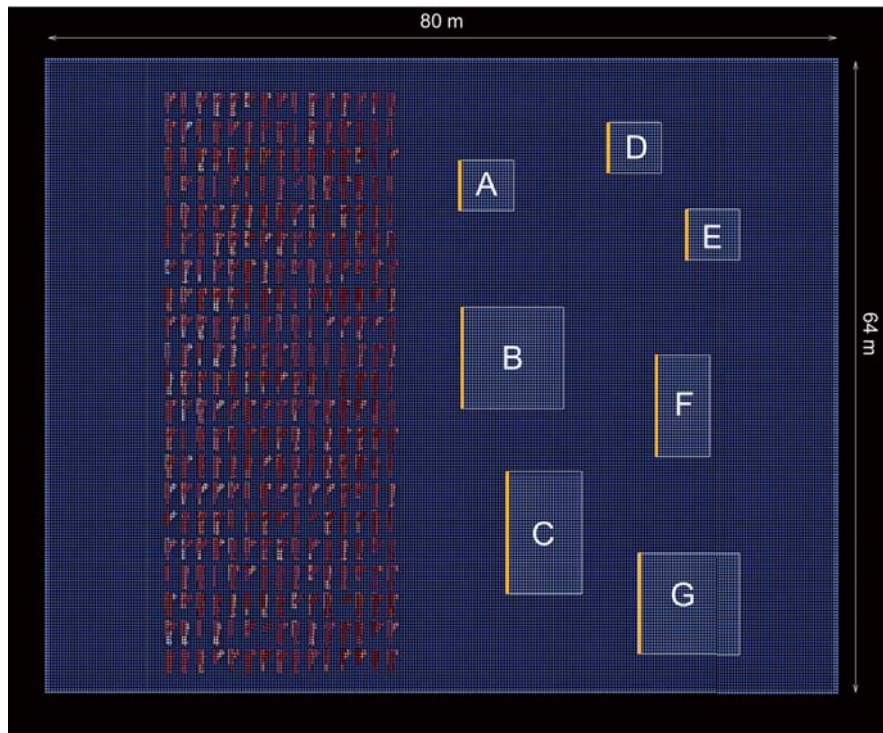


図 5.19 建物を含んだ津波シミュレーションの配置図 (上側から見た図. 黄色は圧力センサーを設置した側面)

表 5.5 各建物のサイズと位置

建物	大きさ (x, y, z)	底面の中心座標 (x, y, z)
A	(5, 5, 10)	(12.5, 44.5, 0)
B	(10, 10, 5)	(30, 47, 0)
C	(12, 7, 3)	(48, 50.5, 0)
D	(5, 5, 15)	(8.5, 59.5, 0)
E	(5, 5, 15)	(17.5, 67.5, 0)
F	(10, 5, 5)	(35, 64.5, 0)
G	(10, 10, 8)	(55, 65, 0)

(単位: m)

色部分の圧力値は約 387 kPa であった. 津波が物体を伴って流れることにより, 流体が衝突したことによる衝撃圧の 6.2 倍以上の圧力が, 集中荷重となって建物に与えられることが分かった. これらが建物の亀裂や破壊の原因となることは容易に理解することができ, 本シミュレーションにより, 建物など構造物の粘り強い施工方法を検討するための有益な知見を得られたと言える. なお, 上述で求めた圧力値のうち, 流木と建物の衝突の際の圧

力値もすべて流体計算から求めた圧力値であり，剛体接触による圧力値の計算は現在に行っていない．すなわち，粒子を剛体連結して表現した流木と建物の衝突計算では DEM のバネとダッシュポットの物理モデルによって求めた接触力から粒子の速度と，座標の修正のみを行っている．今後，DEM のモデルにより求めた接触力から応力を求めるなどして剛体衝突としての圧力値に修正することや，実験との比較により計算された圧力値の妥当性の検証などが課題である．

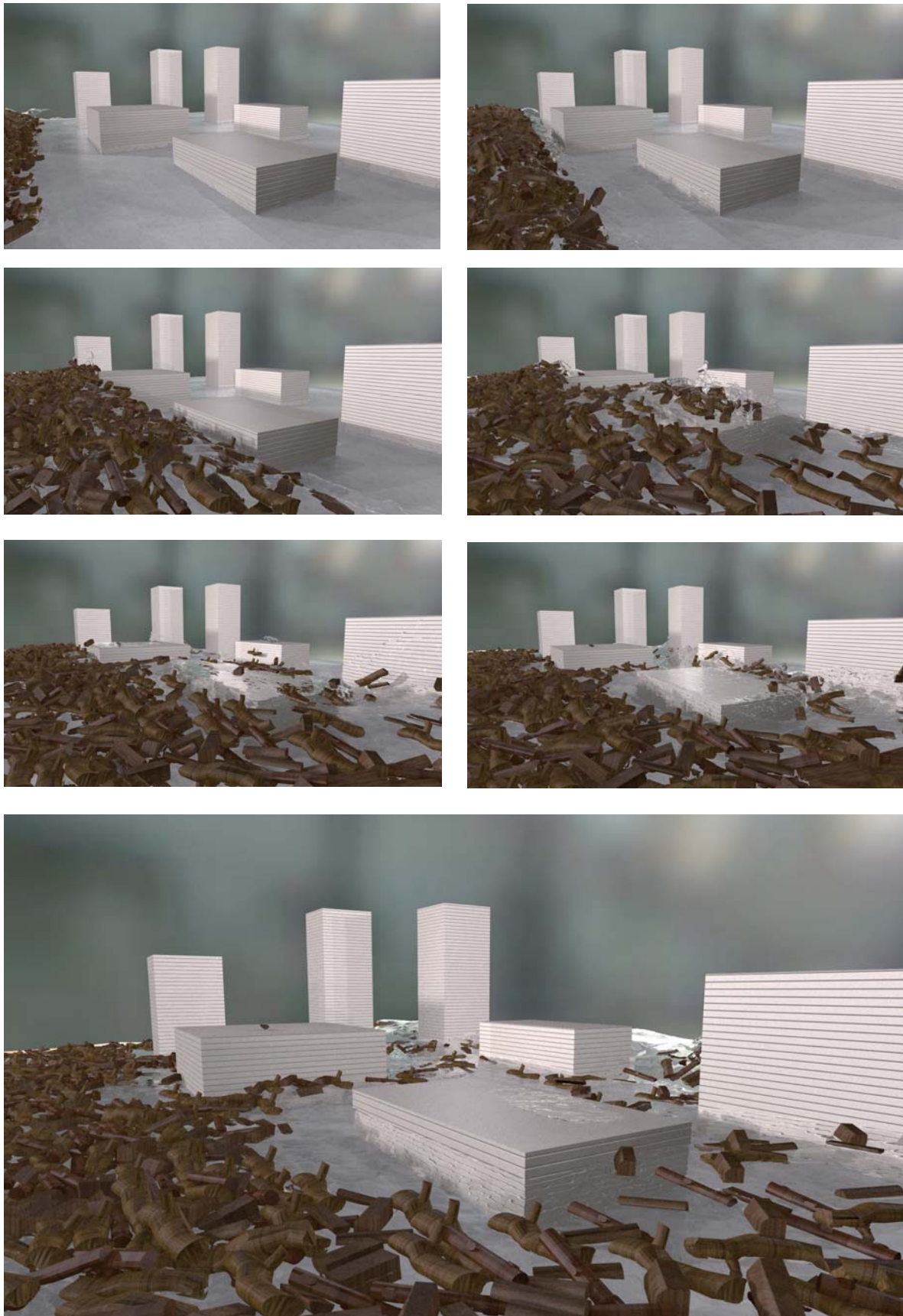


図 5.20 2,062 万粒子, 256 GPU を用いて実行した瓦礫と建物の構造物を含んだ津波シミュレーション

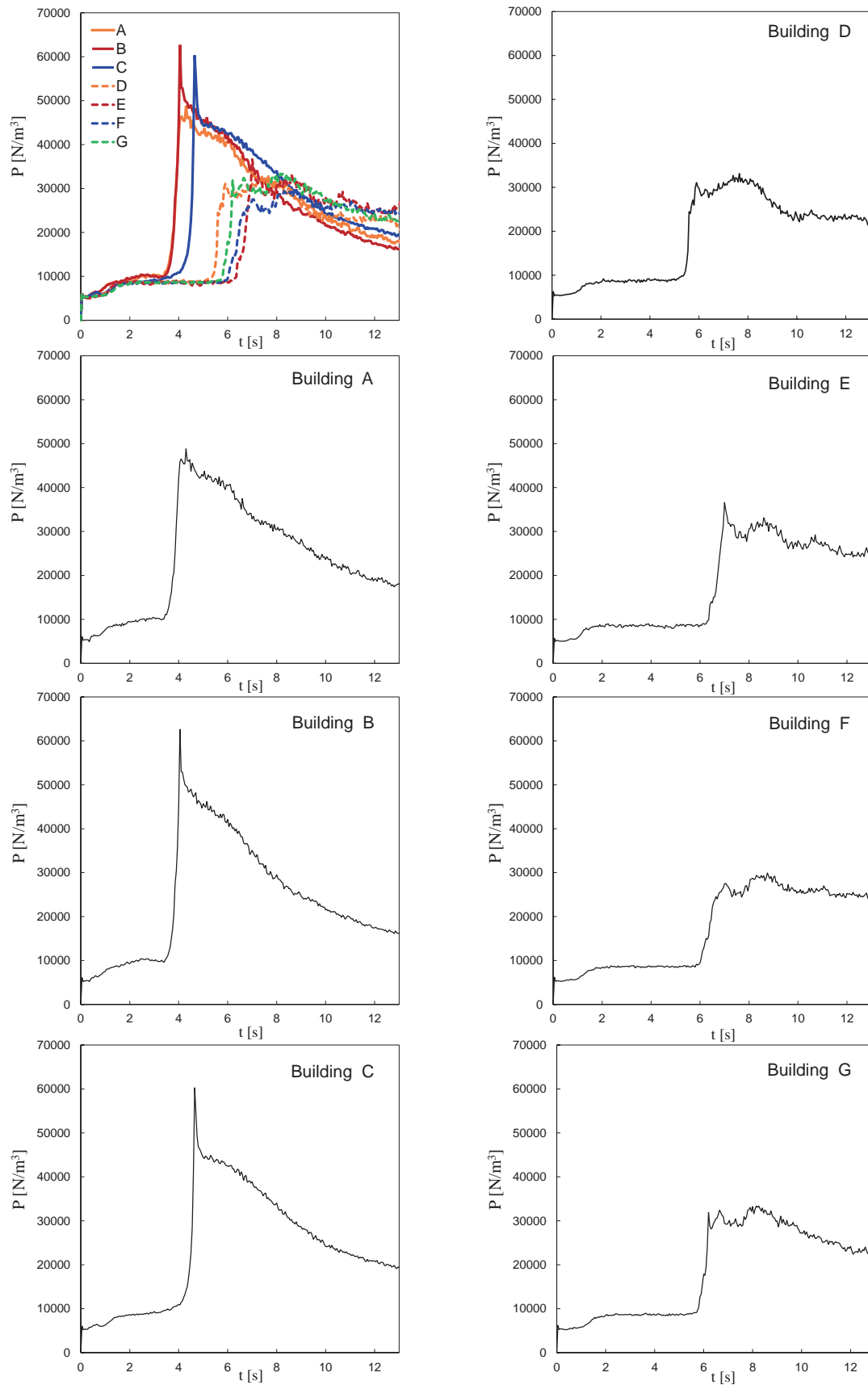


図 5.21 建物 A~G の壁面における圧力値の変化

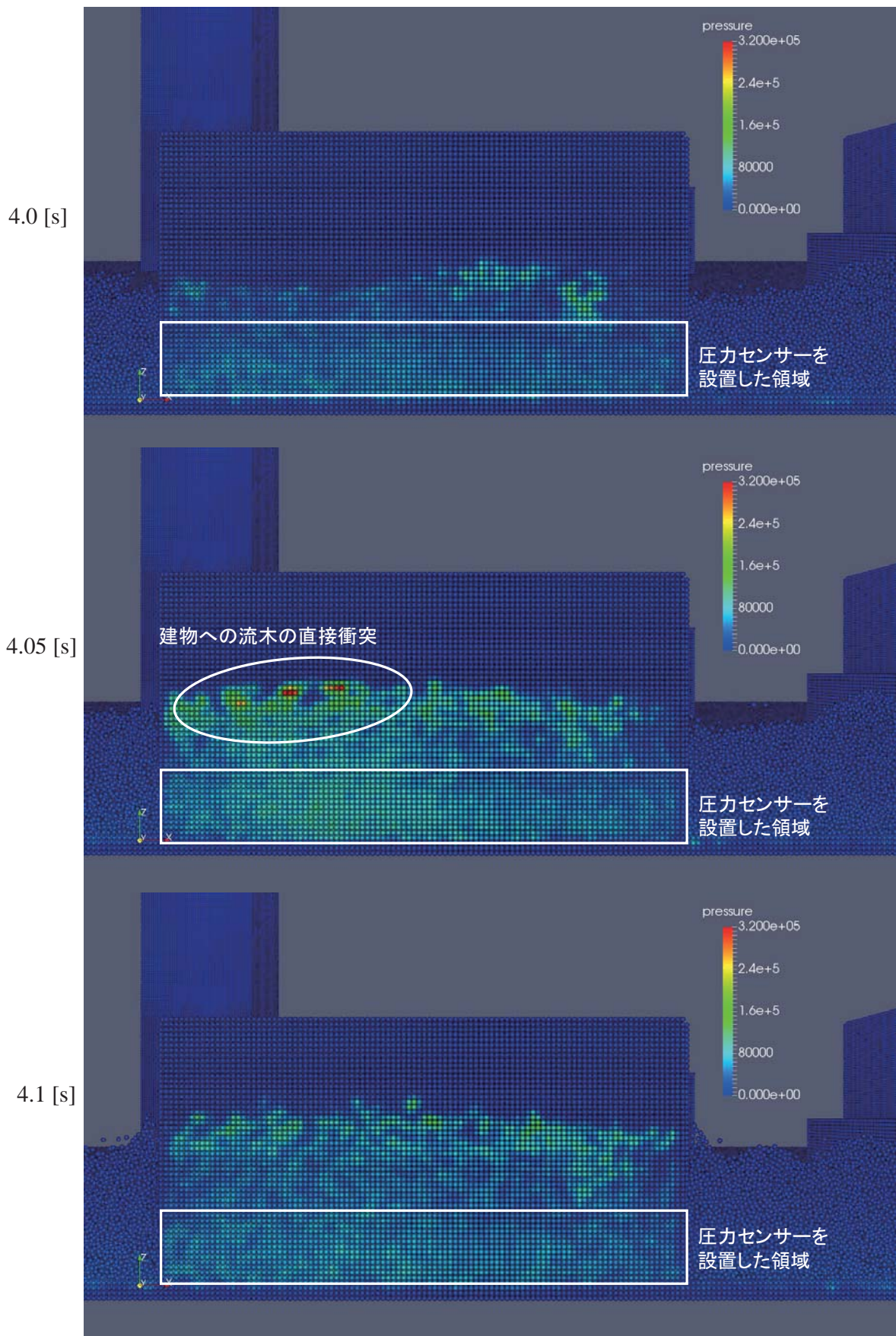


図 5.22 建物 B の壁面における圧力分布 (白線囲み部分は図 5.21 の測定の際のセンサー設置部分)

5.5 多数の瓦礫を含む土石流シミュレーション

計算領域の縦横高さを $180\text{ m} \times 160\text{ m} \times 20\text{ m}$ とし、深さ 2.0 m の静止した水を張り、そこに合計 $10,368$ 個の 12 種類の浮遊する瓦礫 (がれき) を配置する。全部で $117,561,285$ 個 (流体粒子が $93,887,930$ 個、壁粒子が $21,535,585$ 個、物体構成粒子の総数が $2,137,770$ 個) の粒子を用いる。図 5.23 に配置図を、表 5.6 に計算条件の詳細を示す。

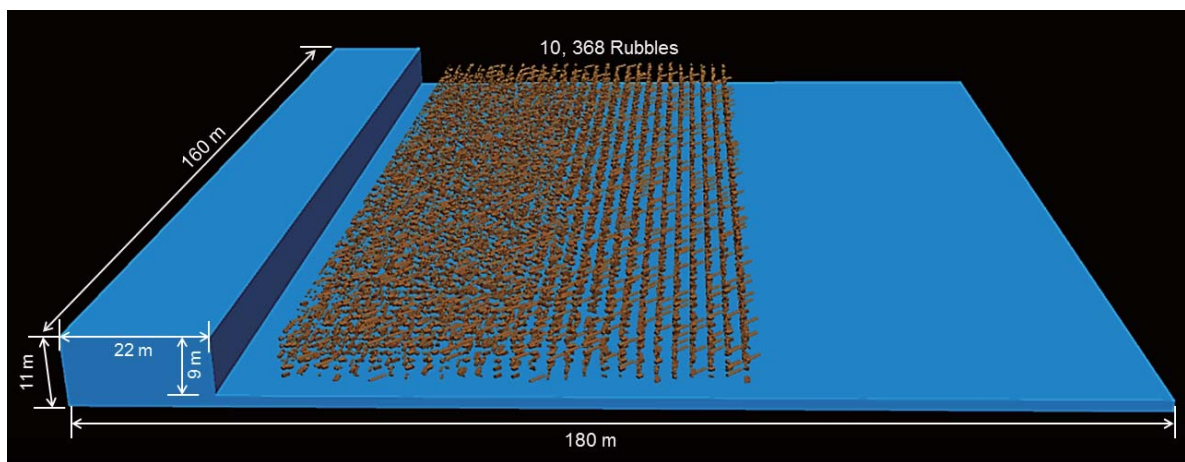


図 5.23 土石流シミュレーションの配置図

表 5.6 計算条件

計算領域	$180\text{ m} \times 160\text{ m} \times 20\text{ m}$
静水域の深さ	2.0 [m]
初期水柱の高さ	11.0 [m]
粒子数	$117,561,285$
瓦礫の数	$10,368$
物体の種類	12
ステップ数	$20,000$
物理時間	10 [s]
GPU _s	256
計算時間	100.0 [h]

図 5.24 に示すように各瓦礫は CAD データから作成した 19 個から 472 個の粒子で構成されている。それぞれの瓦礫の個数と構成要素数の詳細を図 5.7 に示す。高さ 12 m の水

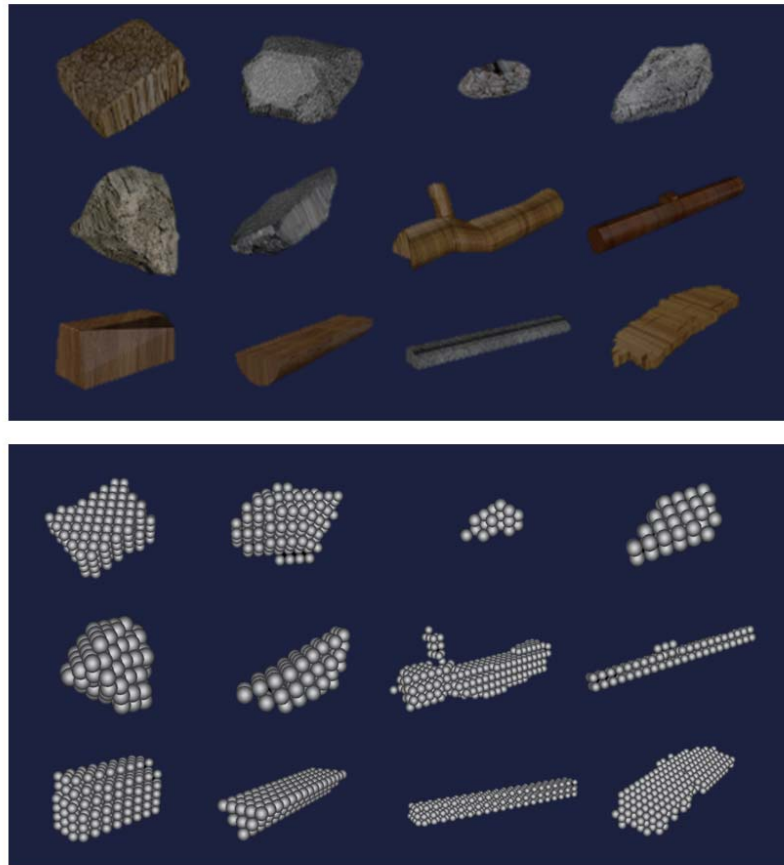


図 5.24 12 種類の瓦礫の CAD データ (上) と、生成された粒子群データ (下)

柱を左側に設置し、瓦礫を水面に落下させて不規則な初期分布を作成した。256 個の GPU を用い、時間刻み幅を 5.0×10^{-4} として 20,000 ステップを 100 時間かけて計算した物理時間で 10 秒経過後までのスナップショットを図 5.25 に示す。第 4.4.1 節でのサスペンション・フローの大規模計算と同様、瓦礫同士の衝突によるエネルギー散逸や、瓦礫と流体との相互作用によって波が碎波し流れが緩やかになり、物理時間で 10 秒経過した段階では波はまだ計算領域の端 (初期の水柱の端から津波の進行方向に 158 m 離れた位置) には到達していないことが分かった。

水柱の崩壊から物理時間で 10 秒経過後までにおける、各時刻における瓦礫の集団全体としての運動エネルギー (各瓦礫のもつ運動エネルギーの総和) をプロットした結果を図 5.26 に示す。図 5.26 に示すように瓦礫の運動エネルギーの変化は (A) 初期値生成段階、(B) 水柱ダムから波力を受けて運動する段階、(C) 引き潮の影響を受ける段階、の 3 段階に分けることができる。(A) では瓦礫を初期に水面に落下させて不規則な初期分布を作成しているため、瓦礫の自由落下に合わせて運動エネルギーが増加し、着水後、徐々に静止

表 5.7 図 5.24 における各瓦礫の個数, 及びそれぞれの構成粒子数

物体番号	1つあたりの構成粒子数	物体の個数	物体構成粒子数の合計
0	245	834	204,330
1	183	867	158,661
2	19	893	16,967
3	44	880	38,720
4	151	836	126,236
5	71	877	62,267
6	472	858	404,976
7	124	869	107,756
8	309	878	271,302
9	144	855	123,120
10	410	856	350,960
11	315	865	272,475
合計:		10,368	2,137,770

状態に向かって運動エネルギーが減少している。(B)では崩壊する水柱から波力を受けて運動エネルギーが増加し、最大で 2.35 MJ となった。水柱の崩壊 9 秒経過後から引き潮が発生することが図 5.25 のスナップショット等から確認でき、その影響を受けて (C) では瓦礫の運動エネルギーは減衰している。

計算時間の内訳を図 5.27 に示す。縦軸は P.132 における図 4.41 の計算手順の各項目に対応している。流体計算部分である 1 番から 6 番は全体の約 32% を占めている。流体構造連成計算の部分である 8 番「物体の運動方程式の時間発展」はこれだけで全体の約 67% を占める。この他、7 番「物体同士の衝突計算」、9 番「領域外粒子の GPU 間通信」が残りの約 1% を占めている。

流体計算部分ではハロー領域の GPU 間通信 (橙色) が、演算部分 (青色) に対して 4 倍になっておりハロー領域の通信量が多い。本シミュレーションでは研究の経緯から 3 次元空間充填曲線による 3 次元領域分割を行ったが、領域形状から分かるように 2 次元空間充填曲線による 2 次元領域分割が適していると考えられ、2 次元領域分割に変更することで図 5.27 のハロー通信時間については削減できる可能性がある。しかし、たとえ流体計算部分が半分程度になったとしても全体の計算時間に対しては 20% 程度の短縮にしかならず、やはり全体の 62% を占める「物体の運動方程式の時間発展 (緑色)」の削減が最も重要な課題である。この部分の詳しい内訳を図 5.27 の上部に示す。緑部分の大部分は GPU 内

の Reduction と GPU 間の Reduction が占めていることが確認できる。

GPU 内及び GPU 間の Reduction には多くの課題がある。GPU 間の Reduction について、現在は物体をマスターとなる GPU で管理しているが、重心を含む GPU 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。重心を含む GPU ごとに物体を管理することが難しい場合にも、例えばスパコンの同一ノード内にある GPU では Direct 通信を導入して CPU のホスト・メモリを介さずに GPU 間の Reduction を直接行い、ノード毎にそれらを取りまとめてマスターに送信するようにすれば、ノード間で通信するプロセス数が少なくなり高速化できる。また、本研究では物体が変形する場合への利用を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り、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。一方、動的負荷分散について、3次元ヒルベルト空間充填曲線を用いた動的領域分割を 50 ステップ毎に実行している。1億 1,756 万個の粒子のうち壁粒子以外の流体粒子と物体構成粒子を所属する小領域ごとに 17 種類の色で塗り分けて示した結果を図 5.29 に示す。各小領域の粒子数は R_{load} が終始 3% 未満の範囲内で均等に保たれていた。

動的負荷分散 1 回あたりの実行時間は図 5.28 に示すように約 1,540 msec であった。1 ステップあたりの全計算時間と比較するとその 1/10 程度であり、動的負荷分散を 50 ステップに 1 回の頻度で実行したことを考慮すればオーバーヘッドはほぼ無視できる。本シミュレーションでは 3次元空間充填曲線を用いた 3次元領域分割を行っており、粒子数及び領域番号テーブルは直方体形状の計算領域に対してそれを含む最小の立方体となる空間に対して 3次元的に確保されている。2次元空間充填曲線による 2次元領域分割を用いる場合よりも空間格子サイズが増加するため、配列の初期化や粒子数テーブルの通信に要する時間は増加している。また、実問題のシミュレーションのため粒子移動も多くなり、負荷分散の実行間隔も広いために次に動的負荷分散を実行するまでの間に粒子分布が大きく変化することでリーフ・マイグレーションのコストが増加している。現状では動的負荷分散 1 回のオーバーヘッドは 1 ステップあたりの実行時間と比較すると相対的には無視できるが、空間格子によるメモリ容量の削減や直方体形状の計算領域への対応など動的負荷分散 1 回あたりの実行コストの削減を検討することは重要である。

本シミュレーションにより、設定する問題サイズ(水柱ダム、瓦礫数、等)に対する瓦礫に付与される運動エネルギーの関係を示すことができ、今後、前節の建物と流木を含む津波シミュレーションと合わせて、現実における津波や土石流の被害状況の予測や、運動エネルギーを軸としたそれらのハザードマップの作成に役立てることが期待できる。

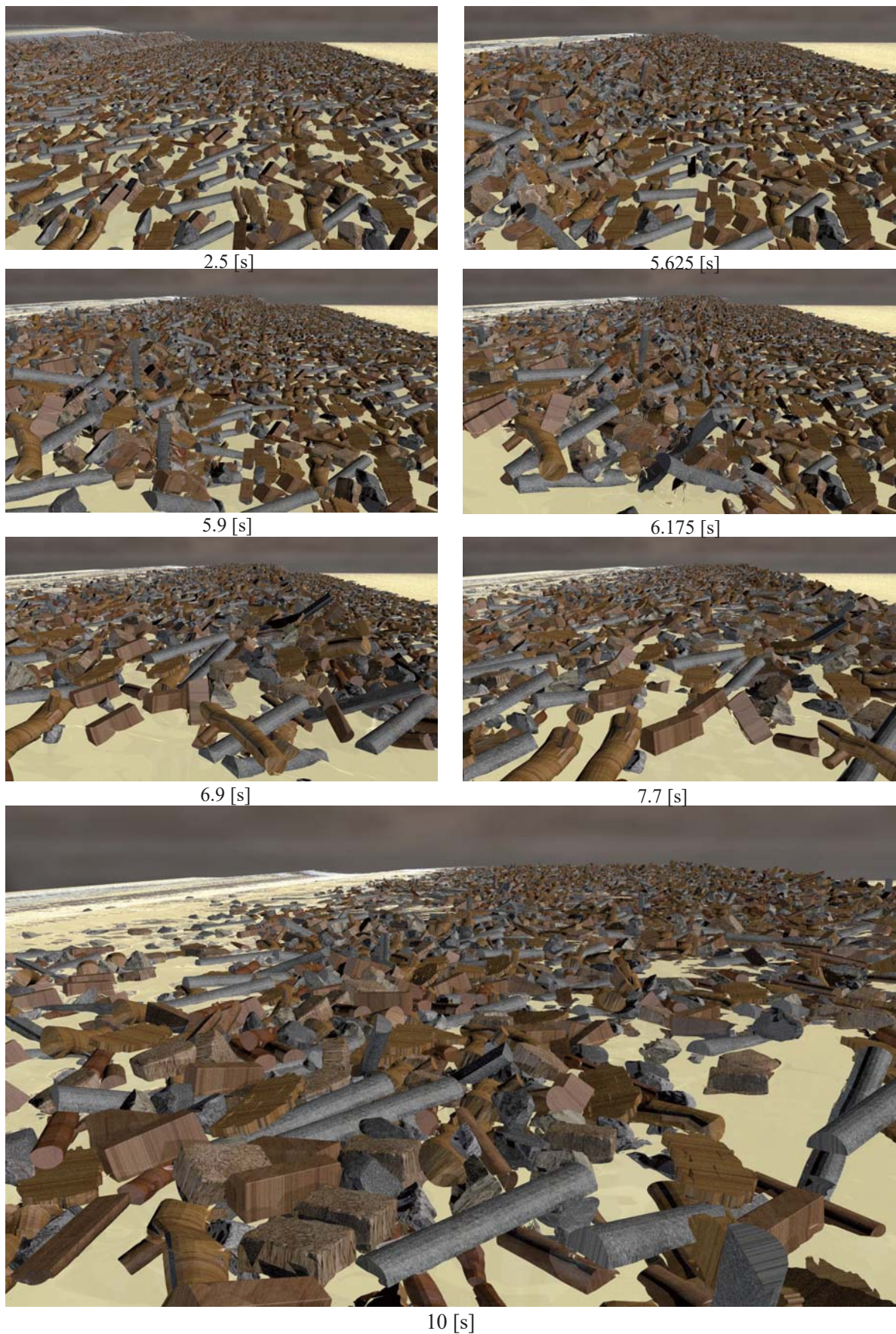


図 5.25 256 台の GPU を用いて計算した 10,368 個の瓦礫と相互作用する 1 億 1,756 万粒子による大規模土石流シミュレーション

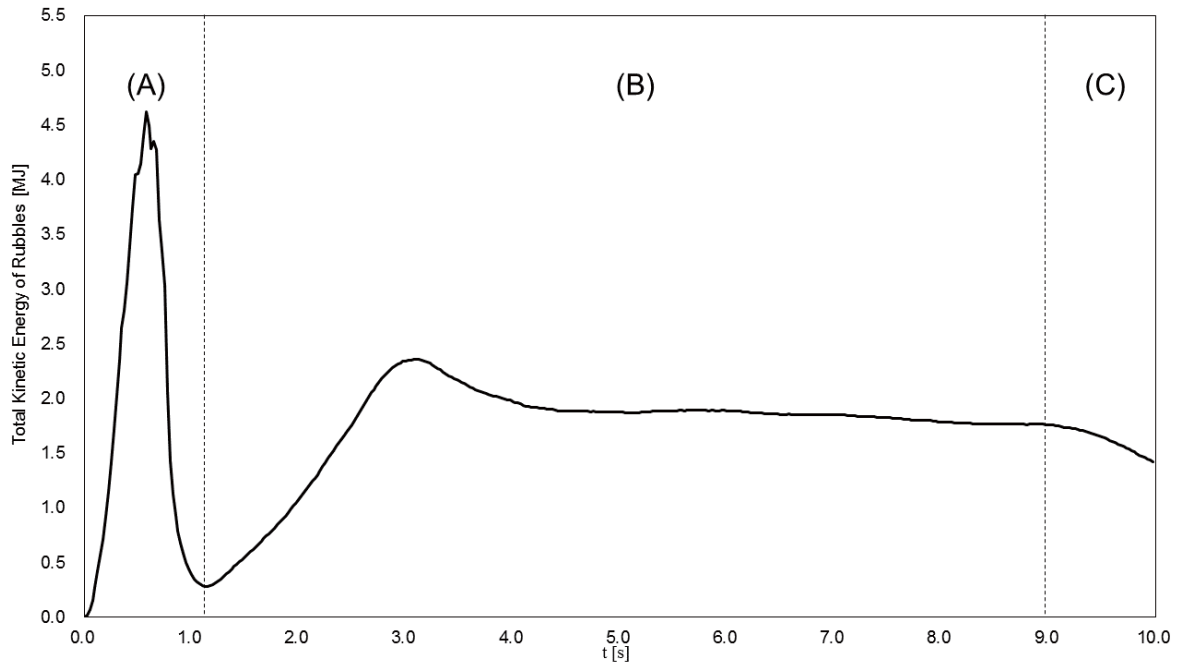


図 5.26 10,368 個の瓦礫の運動エネルギーの総和の時間変化

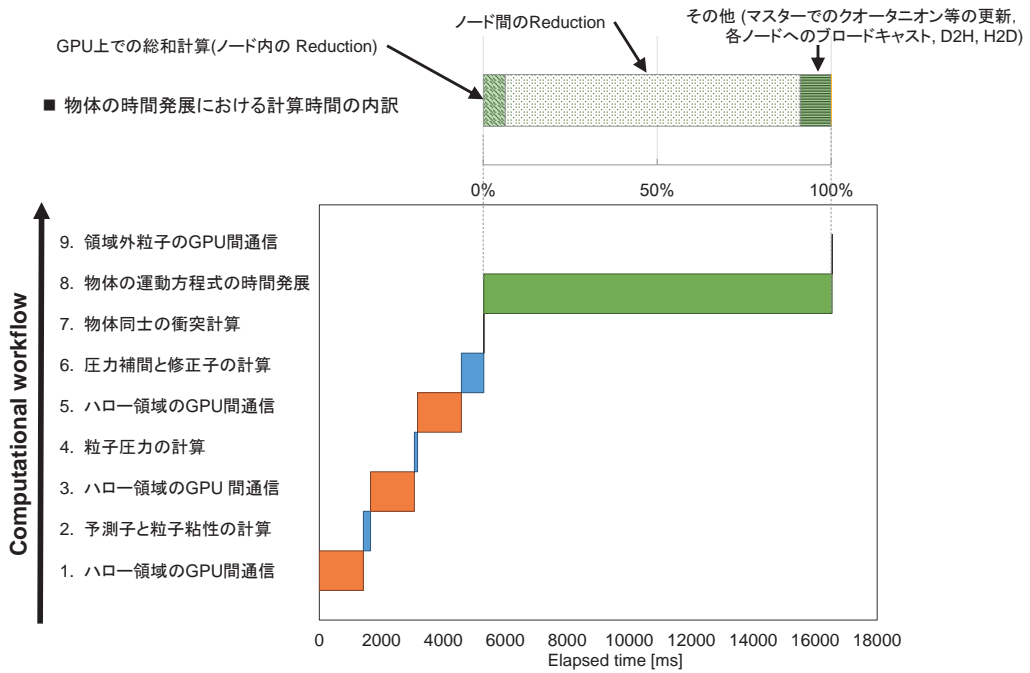


図 5.27 大規模土石流シミュレーションにおける計算時間の内訳

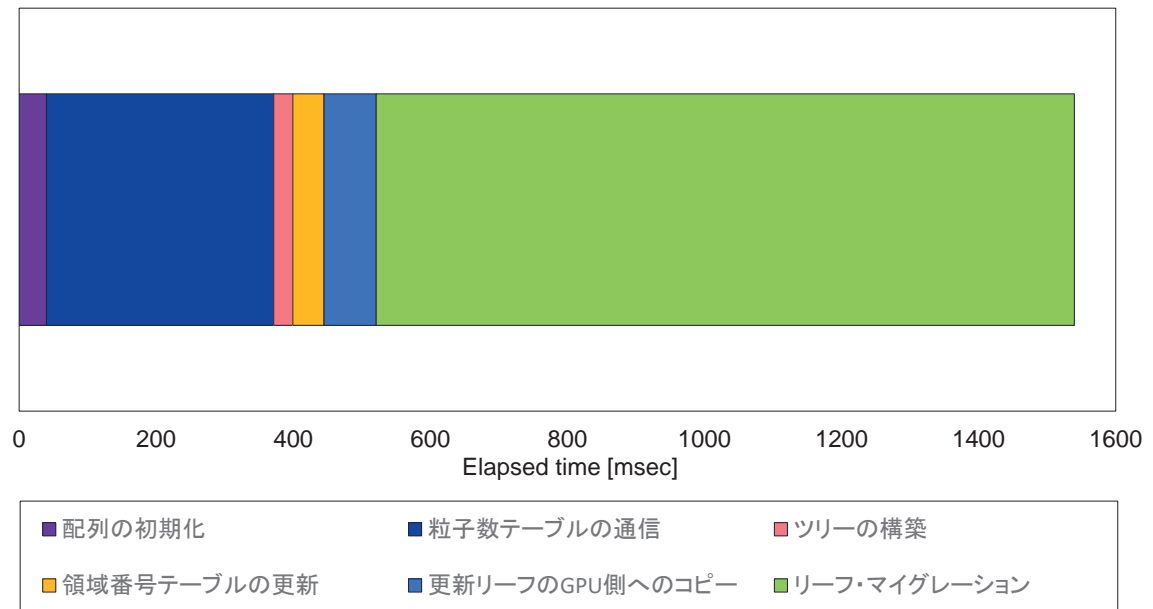


図 5.28 動的負荷分散を 1 回実行するのに要する計算時間の内訳

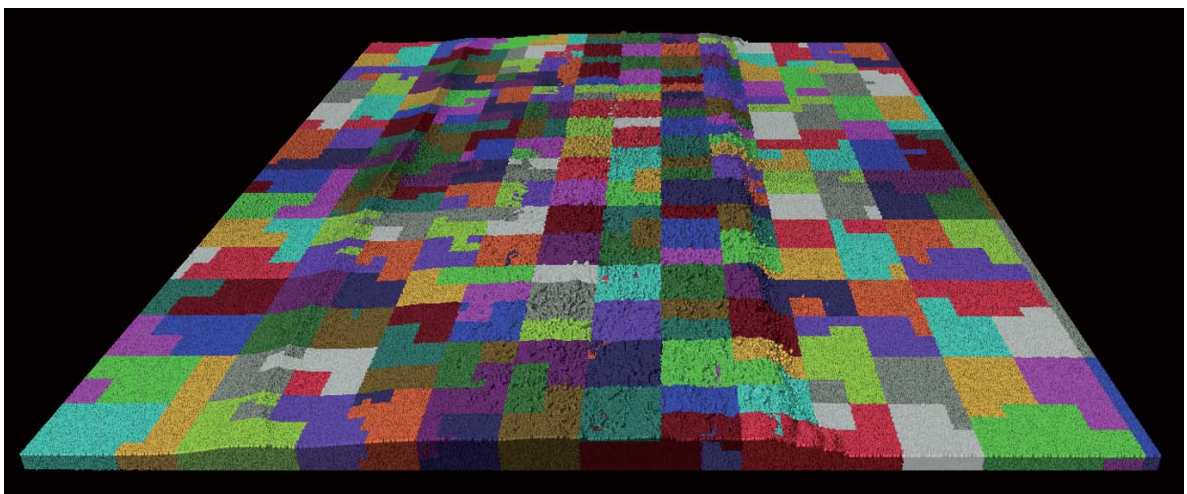


図 5.29 壁粒子以外の流体粒子と物体構成粒子を所属領域ごとに 17 種類の色で塗り分けて示した結果

第 6 章

結論

粉体や流体の粒子法による大規模シミュレーションの重要性は広く認識されている。しかし、それらのシミュレーションでは領域再分割による動的負荷分散の実装は容易ではなく、計算負荷が均一化されないために分散メモリ環境では大規模粒子計算が殆ど行われてこなかった。本研究では GPU スパコンにおいて DEM や SPH 法など近接相互作用に基づく粒子法の大規模シミュレーションを高効率に実行する計算手法を提案し、1 億個の粒子による粉体や流体、瓦礫を伴う津波や土石流の大規模シミュレーションを実現することができ、粉体や流体、及びその連成問題に対して GPU スパコンを用いて粒子法による詳細な解析が可能になることを明らかにした。以下、各章を総括する。

第 3 章では、近傍粒子探索により相互作用計算における演算量を $O(N)$ に抑え、Linked-list 法を用いることにより空間格子に使用するメモリ使用量を大幅に削減した。これらは粒子法計算における高速化の常套手段であるが、本研究ではさらにこれらと併用して、CPU 側からのセル番号によるソートを定期的に行う方法を提案し GPU 計算に特徴的な 32 連続アドレスずつのデータ・ロードの高効率化を試みた。その結果、改良型 SPH 法による流体の単一 GPU 計算に対して約 8.6 倍の高速化を達成することができた。流体構造連成の GPU 計算では多数の物体の力とトルクの総和計算を行う方法についてアルゴリズムの異なる 2 通りの実装を示し、それぞれの適する場合について検討した。

第 3 章の後半では、粒子法の複数 GPU 計算における粒子移動部分の効率的な計算方法を述べた。粒子法の複数 GPU 計算では GPU 間の頻繁なデータ通信の繰り返しにより、粒子計算に利用される GPU 上のメモリは断片化し計算時間が増大してしまう。定期的に粒子を再整列させることにより GPU メモリの断片化を解消する方法を提案し、再整列を行わない場合に時間ステップの 2 乗に比例して増加する積算計算時間を線形増加に抑制し、1 ステップあたりの平均の計算時間を一定にできることを明らかにした。一方、再整列の処理は計算コストが高いためしばしば計算のオーバーヘッドとなる。再整列の頻度と計算時間の関係について調べ、一定速度場の問題に対して再整列の頻度の最適解を求める

モデルを提案した。

第4章では、スライスグリッド法、及び3種類の空間充填曲線に基づく動的負荷分散法を、DEMやSPH法の複数GPU計算で導入する方法を提案した。GPUスパコンにおいてDEMによる粉体計算の強・弱スケーリングを実施し、2次元領域分割のスライスグリッド法の適用限界が256GPU~512GPU程度であることを示した。

改良型SPH法を用いた3次元流体計算の強・弱スケーリングをTSUBAME 2.5上で実施して各動的負荷分散法の比較を行い、2次元ペアノ曲線及び2次元ヒルベルト曲線では弱スケーリングで256GPUを用いた時点で並列化効率95%以上を達成した。2次元モートン曲線では並列化効率は85%であった。同一の問題に対してスライスグリッド法を用いた場合の並列化効率は49%となっており、スライスグリッド法において領域形状が高アスペクト比になり領域間通信量が増加して並列化効率が著しく低下する問題に対して、空間充填曲線を用いることで曲線の局所性により領域接続数が減少し、特にヒルベルト曲線とペアノ曲線では曲線が交差しないためにGPUの台数を増加させても隣接接続関係が殆ど変わらず、並列化効率を大幅に向上できることが明らかになった。強スケーリングにおいても、256GPUを用いた計算において空間充填曲線を用いることによりスライスグリッド法に対して2.3倍以上の性能向上を達成した。一方、理想的な場合と比較すると256GPUを用いた段階で51%まで性能が低下する結果を得た。今後、計算と通信のオーバーラップを導入することによる高速化が課題である。

改良型SPH法を用いた3次元流体計算に対して、2次元空間充填曲線による2次元領域分割と3次元空間充填曲線による3次元領域分割を用いた場合の性能比較を行い、2次元領域分割を用いた場合において、3次元領域分割と比べて領域の表面積が悪化しデータ通信量が増加したとしても隣接小領域との接続数が減少することにより全体の通信コストは削減され結果的に計算時間が短縮される場合があることが分かった。

第4章の後半ではスライスグリッド法による動的負荷分散を流体構造連成問題に適用し、2,304個の立方体形状の物体を含むサスペンション・フローの大規模計算を8,743万粒子を用いて実現した。弱スケーリングを調べた結果、4GPUを用いた場合に対して、256GPU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。ノード内及びノード間のReductionの方法について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。重心を含むGPUごとに物体を管理することが難しい場合にも、例えばスパコンの同一ノード内にあるGPUではDirect通信を導入してCPUのホスト・メモリを介さずにGPU間のReductionを直接行い、ノード毎にそれらを取りまとめてマスターに送信するようになれば、ノード

間で通信するプロセス数が少なくなり高速化できる。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。

第5章では提案する計算手法を実問題に適用し、粉体や流体、及び流体構造連成の大規模シミュレーションを実現した。1,670万個の粒子による64 GPUを用いたゴルフ・バンカーショット解析では、サンドウェッジの位置とスイング速度の異なる複数の計算条件に対してゴルフボールの軌道解析からボールの打ち出し角度と初速を得ることができた。本計算結果自体もサンドウェッジの設計において有益な情報である。現実サイズの粒子による3次元のバンカーショット解析を、GPUスパコンを用いることにより実用的な時間内で実現可能にした意義は大きい。第5章の第5.4節では2,062万個の粒子による256 GPUを用いた945個の4種類からなる流木と7棟の建物を含む津波シミュレーションを実行し、津波が物体を伴って流れることにより、物体が直接建物に衝突した際に津波が衝突したことによる衝撃圧よりも遙かに大きい圧力が集中荷重となって建物に与えられることが分かった。これらが建物の亀裂や破壊の原因となることは容易に理解することができる。第5章の第5.5節では多数の瓦礫を含む土石流シミュレーションを1億1,756万個の粒子による256 GPUを用いた10,368個の12種類からなる瓦礫を含む大規模土石流シミュレーションを達成することができた。瓦礫同士の衝突や流体から物体への作用によりエネルギーが失われ、残念ながら流れが対岸の壁面まで到達しなかった。しかし、瓦礫に対しては集団運動としての運動エネルギーの変化を求めることができ、本シミュレーションにより設定する問題サイズ(水柱ダム、瓦礫数、等)に対する瓦礫に付与される運動エネルギーの関係を示すことができた。

本論文ではDEMやSPH法などの近接相互作用に基づく粒子法の大規模シミュレーションをGPUスパコンにおいて高効率に実行する計算手法を提案した。以下、主題である動的負荷分散を軸に全体を総括する。本研究において得られた新しい知見としては、

- Linked-listを用いたGPUにおける近傍粒子探索に対して定期的なセル番号によるソートを行うことでデータ・ロードの高効率化を実現し計算時間を大幅に短縮できた。
- GPU間で粒子数を均一化する動的負荷分散を行うことで、複数GPU計算でメモリが枯渇する問題や並列化効率が低下する問題を解決し、GPUスパコンで大規模計算が可能になることを明らかにした。特に、領域分割法としてスライスグリッド法を用いた場合に領域形状が高アスペクト比になり領域間通信量が増加して並列化効率の著しく低下することを示し、この問題に対して空間充填曲線による木(ツリー)を用いた格子細分化による動的負荷分散法を用いることで解決し、実行性能を大幅

に向上できることを明らかにした。

- 粒子データの頻繁な GPU 間通信により生じる GPU メモリの断片化の問題に対し、再整列を行わない場合に計算ステップ数の 2 乗に比例して増加する積算計算時間を線形増加に抑え、1 ステップあたりの計算時間を一定にできることを明らかにした。

の 3 点が挙げられ、これについて以下の様に議論する。

各 GPU では近傍粒子探索と Linked-list 法に加えて、近傍粒子探索の空間格子のセル番号による粒子データのソートを行い GPU のデータ・ロードの高効率化を図り高速化を実現した。空間格子のセル番号による粒子データのソートは、各スレッドが自身の担当する粒子の所属セルに構築された Linked-list を辿ってグローバルメモリ上の粒子データにアクセスする際のアクセス効率をセル同士で均等にする処理であるといえる (ここで、アクセス効率とは Linked-list を辿るのに要する時間を Linked-list を構成する粒子数で割った値を意味している)。空間格子の各セル内の粒子数は均等ではないためセルごとのメモリ・アクセスの絶対量は等価ではなく、現在は各セルの粒子数のばらつきは最大収容数の範囲内で認めつつ、ソートにより同一セル内の粒子データを連続させることで各スレッドのグローバルメモリへのアクセス効率を均一にして高速化を実現している。これは空間分割の最小単位であるセル同士のメモリ・アクセスについての負荷分散の問題と考えることができ、アクセス効率についての動的負荷分散は達成されていることになる。もしも領域間だけでなく分割された各小領域内にも空間充填曲線を巡らせて粒子データを管理したとすると、木 (ツリー) の各リーフの保持する粒子数は等しいので、Linked-list 構造を現在のセル単位から木 (ツリー) のリーフ単位に変更することで、アクセス効率だけでなくメモリ・アクセスの絶対量についてもリーフ間で均等になる。

一方、粒子法の複数 GPU 計算では計算領域を空間分割し分割された各小領域に GPU を割り当て、それぞれの GPU が担当する領域内の粒子を計算する。計算領域を均等分割する場合、粒子分布が時間的、空間的に変化して各 GPU が処理する粒子数に偏りが生じ、GPU のメモリ容量を超過する分量の粒子が特定の GPU に流入した場合にメモリ不足を引き起こすことが原因となり計算が続行出来なくなる。メモリ容量が数 GB に制約された GPU を搭載する GPU スパコンにおいては特に深刻な問題であり、このようなメモリが枯渇する問題に加えて、粒子が密集した小領域を担当する GPU の計算負荷が著しく上昇することにより大規模計算は実質的に不可能になる。本研究では GPU 間で動的負荷分散を導入することによりこれらの問題を解消し、複数 GPU を用いた近接相互作用に基づく粒子法シミュレーションを GPU スパコンで実現できることを示した。さらに GPU 間の動的負荷分散について粒子分布を追従するタイプの直感的に理解しやすいスライスグリッド法による動的負荷分散と、空間充填曲線による再帰的な領域分割を行う動的負荷分散というアルゴリズムの異なる 2 種類の動的負荷分散法を比較し、スライスグリッド法の適用限

界は 256 GPU ~ 512 GPU 程度であることを明らかにした。一方、スライスグリッド法の適用限界の領域においても空間充填曲線を用いることによりスライスグリッド法において領域形状が高アスペクト比になることによる並列化効率の著しい低下を解消し大幅な実行性能の向上を達成できた。空間充填曲線を用いることで粒子法の大規模シミュレーションを並列化効率を維持したまま GPU スパコンで実行できることを明らかにした本研究の意義は大きい。さらに本研究では粒子データの頻繁な GPU 間通信によって発生する GPU メモリの断片化により個々の GPU で不要な未参照メモリが発生して実行性能が低下することを示し、それに対して定期的に粒子をメモリの先頭から詰め直す粒子の再整列を行うことで計算ステップ数が増加しても 1 ステップあたりの平均の計算時間を一定に保つことができることを明らかにしている。GPU 間の粒子の移動量が多いところほどメモリの断片化の程度は著しく、GPU メモリの断片化によっても各 GPU の計算負荷の不均一が生じる。並列計算では全体の計算時間は最遅のプロセッサに律速されるため、GPU メモリの断片化に合わせて実行性能が低下する。粒子の再整列は そのような GPU メモリの断片化によって生じる計算負荷の不均一を解消するための動的負荷分散である。

以上、本研究では個々の GPU における CUDA スレッド間でのメモリ・アクセス効率の均一化、GPU 間の粒子数の均一化による動的負荷分散、GPU メモリの断片化に対する動的負荷分散 (粒子の再整列) という複数の階層的な動的負荷分散を合わせて行うことにより、GPU スパコンにおいて近接相互作用に基づく粒子法シミュレーションを高効率に実行できることを明らかにし、大規模シミュレーションによってその有効性を示すことに成功した。現在のスパコンは多数のノードで構成されており、ノード間が高速なネットワークで繋がれていることは序論でも述べた。今後このようなスパコンの構成は変わることはなく、本研究で得られた GPU スパコンにおける高効率な計算手法は、GPU スパコン以外の演算加速器型スパコンにも類比して適用でき有効である。本研究成果により粉体や流体、その連成問題に対し、GPU スパコンを用いて粒子法による詳細な解析が可能であることを明らかにできた。以上をもって本論文の結論とする。

付録 A

ゴルフ・バンカーショットの解析 (別視点から可視化した結果)

第 5.2 節で説明したゴルフ・バンカーショットの解析と同様のシミュレーション (第 5.2 節の表 5.1 と同一の計算条件のもと, サンドウェッジのスイング速度を 2 倍の 10 m/s にした場合) の計算結果を図 A.1 に示す. 第 5.2 節で示した計算結果とは別の視点から可視化している. このシミュレーションではサンドウェッジのスイング軌道の解析は行っていないが, 本問題への理解を助けるため付録として掲載する.

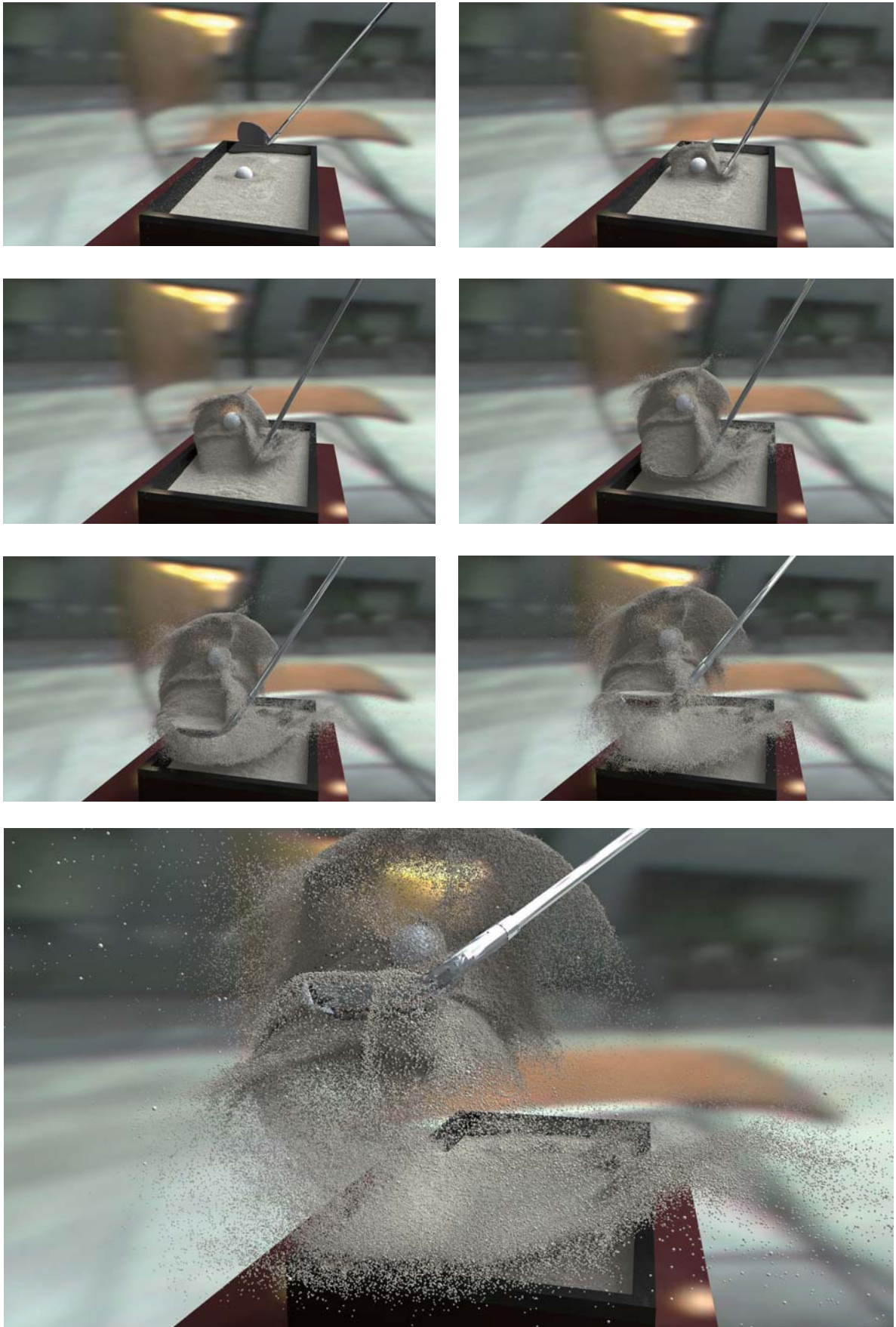


図 A.1 64 GPU を用いた 1,670 万個の粒子による大規模バンカーショット計算 (表 5.1 と同一の計算条件のもと、サンドウェッジのスイング速度を 2 倍にした結果)

謝辞

本研究を遂行するにあたり、熱心なご指導を賜り、最高の研究環境を与えて下さった学術国際情報センター 青木 尊之 教授に心から感謝申し上げます。この5年間で数え切れないほど多くの知恵や技術、様々な貴重な経験を授かりました。これらは私の今後の人生の糧となる大切な宝物であり、いつか先生のように周囲に多くのものを与えられる人へと成長して行けたらと願っています。

コーディング技術やスクリプト処理など研究を進める上で必要となる様々な技術を教えて頂いた、学術国際情報センター 下川辺隆史 助教に深く感謝申し上げます。特に修士課程時代に下川辺さんに教わったノウハウは、今となっては研究を進める上で欠かせないものになっております。学会の出張申請、学振の予算の執行など、秘書の高橋さんには本当にお世話になりました。この場を借りて改めて感謝申し上げます。

個別要素法 (DEM) による粉体計算を導入するにあたり、東北大学 災害科学国際研究所の森口周二 准教授に多くのご助言を賜りました。深く感謝申し上げます。

改良型 SPH 法による流体計算を導入するにあたり、九州大学 マス・フォア・インダストリ研究所の田上大助 准教授、九州大学大学院 数理学府 の井元佑介氏に深く感謝申し上げます。同期の井元さんには研究上で問題が生じたときも常に丁寧で迅速な支援を頂きました。また井元さんも博士論文執筆の非常に忙しい時期であるにもかかわらず、私の急な質問やお願いにも親切に対応して下さいました。本当にありがとうございました。

研究室の先輩である杉原さんには3年次に同室させて頂き、研究の合間に進路や研究方針について多くのアドバイスを頂きました。小野寺さんには修士課程において、自転車で江ノ島に連れて行って頂いたり、楽しい思い出を沢山作っていただきました。チェスターさんには修士課程の頃から学生室で、研究の助言やアドバイスをはじめ様々なことを教えて頂きました。また、プライベートでも剣道の世界大会の観戦に行ったりと公私ともに大変お世話になりました。本当にありがとうございました。粒子法が研究テーマの渡辺勢也君との議論は研究を進める上で大きな助けになりました。そのほか、青木研究室の多くの方々にお世話になりました。この場を借りて感謝申し上げます。また、私が学部4年生のときからいつも応援して下さい、様々な相談に乗って頂いた原佑介氏に感謝申し上げ

ます。

最後になりますが，学部・修士時代と合わせて9年もの長い間，遅い帰宅にもかかわらず絶えず私を応援し支えてくれた両親，時間を割いて招待講演に駆けつけてくれた弟，多くの面で支援し助言と励ましを与え続けてくれた祖父母に心から感謝の意を表したいと思います。

これまで私を支えて下さったすべての方々に，この場を借りて感謝申し上げます。

参考文献

- [1] Daniel Schiochet Nasato, Christoph Goniva, Stefan Pirker, and Christoph Kloss. Coarse Graining for Large-scale DEM Simulations of Particle Flow - An Investigation on Contact and Cohesion Models. *Procedia Engineering*, Vol. 102, pp. 1484–1490, 2015. New Paradigm of Particle Science and Technology Proceedings of The 7th World Congress on Particle Technology.
- [2] 酒井幹夫, 越塚誠一. 固気混相流における離散要素法の粗視化手法の開発. 粉体工学会誌, Vol. 45, No. 1, pp. 12–22, 2008.
- [3] Joseph P. Morris, Patrick J. Fox, and Yi Zhu. Modeling Low Reynolds Number Incompressible Flows Using SPH. *Journal of Computational Physics*, Vol. 136, No. 1, pp. 214–226, 1997.
- [4] S. Koshizuka and Y. Oka. Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear Science Engineering*, Vol. 123, No. 3, pp. 421–434, 1996.
- [5] B. Ben Moussa. On the Convergence of SPH Method for Scalar Conservation Laws with Boundary Conditions. *Methods Appl. Anal.*, Vol. 13, No. 1, pp. 29–62, 2006.
- [6] B. Ben Moussa and J. P. Vila. Convergence of SPH method for scalar nonlinear conservation laws. *SIAM Journal on Numerical Analysis*, Vol. 37, No. 3, pp. 863–887, 2000.
- [7] David A. Fulk and Dennis W. Quinn. An Analysis of 1-D Smoothed Particle Hydrodynamics Kernels. *Journal of Computational Physics*, Vol. 126, No. 1, pp. 165–180, 1996.
- [8] S. Mas-Gallic and P. A. Raviart. A particle method for first-order symmetric systems. *Numerische Mathematik*, Vol. 51, No. 3, pp. 323–352, 1987.
- [9] N. J. Quinlan, M. Basa, and M. Lastiwka. Truncation error in mesh-free particle methods. *International Journal for Numerical Methods in Engineering*, Vol. 66, No. 13, pp. 2064–2085, 2006.
- [10] P. A. Raviart. *Numerical Methods in Fluid Dynamics: Lectures given at the 3rd 1983*

- Session of the Centro Internazionale Matematico Estivo (C.I.M.E.) held at Como, Italy, July 7–15, 1983*, chapter An analysis of particle methods, pp. 243–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.
- [11] T. Ishiyama, K. Nitadori, and J. Makino. 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.
- [12] Volker Springel. The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, Vol. 364, pp. 1105–1134, May 2005.
- [13] Christian Konrad. On domain decomposition with space filling curves for the parallel solution of the coupled Maxwell/Vlasov equations. Research Report RR-6693, INRIA, 2008.
- [14] Marios D. Dikaiakos and Joachim Stadel. A Performance Study of Cosmological Simulations on Message-passing and Shared-memory Multiprocessors. In *Proceedings of the 10th International Conference on Supercomputing, ICS '96*, pp. 94–101, New York, NY, USA, 1996. ACM.
- [15] Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John L. Hennessy. Load Balancing and Data Locality in Adaptive Hierarchical N-body Methods: Barnes-Hut, Fast Multipole, and Radiosity. *Journal Of Parallel and Distributed Computing*, Vol. 27, No. 2, pp. 118–141, 1995.
- [16] Alexander Heinecke, Wolfgang Eckhardt, Martin Horsch, and Hans-Joachim Bungartz. Parallelization of md algorithms and load balancing. In *Supercomputing for Molecular Dynamics Simulations*, SpringerBriefs in Computer Science, pp. 31–44. Springer International Publishing, 2015.
- [17] P. A. Cundall and O. D. L. Strack. A discrete numerical model for granular assemblies. *Geotechnique*, Vol. 29, No. 1, pp. 47–65, 1979.
- [18] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics-theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, Vol. 181, pp. 375–389, November 1977.
- [19] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astron. J.*, Vol. 82, pp. 1013–1024, December 1977.
- [20] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, Vol. 68, pp. 1703–1759, August 2005.
- [21] Jiun-Shyan Chen, Chunhui Pan, Cheng-Tang Wu, and Wing Kam Liu. Reproducing Kernel Particle Methods for large deformation analysis of non-linear structures. *Computer*

-
- Methods in Applied Mechanics and Engineering*, Vol. 139, No. 1-4, pp. 195–227, 1996.
- [22] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering*, Vol. 37, No. 2, pp. 229–256, 1994.
- [23] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: Diffuse approximation and diffuse elements. *Computational Mechanics*, Vol. 10, No. 5, pp. 307–318, 1992.
- [24] Y. Krongauz and T. Belytschko. A Petrov-Galerkin Diffuse Element Method (PG DEM) and its comparison to EFG. *Computational Mechanics*, Vol. 19, No. 4, pp. 327–333, 1997.
- [25] T. Hamada, R. Yokota, K. Nitadori, T. Narumi, K. Yasuoka, and M. Taiji. 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, November 2009.
- [26] 青木尊之, 額田彰, 第二 I/O 編集部. はじめての CUDA プログラミング. I/O books. 工学社, 2009.
- [27] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2014. Version 6.0.
- [28] Michael Bader, Arndt Bode, Hans-Joachim Bungartz, Michael Gerndt, Gerhard R. Joubert, and Frans Peters, editors. *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, Vol. 25 of *Advances in Parallel Computing*, April 2014.
- [29] 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. 日本計算工学会論文集, 2012.
- [30] K. Murotani, S. Koshizuka, M. Ogino, R. Shioya, and Y. Nakabayashi. Development of distributed parallel explicit 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, November 2014.
- [31] J.M. Domínguez, A.J.C. Crespo, D. Valdez-Balderas, B.D. Rogers, and M. Gázquez-Gesteira. New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, Vol. 184, No. 8, pp. 1848–1860, 2013.
- [32] MPI Forum. Message Passing Interface (MPI) Forum Home Page. <http://www.mpi-forum.org/> (2009).
- [33] Ji Qiang, Robert D. Ryne, Salman Habib, and Viktor Decyk. An Object-Oriented Parallel Particle-in-Cell Code for Beam Dynamics Simulation in Linear Accelerators. *Journal of Computational Physics*, Vol. 163, No. 2, pp. 434–451, 2000.

- [34] Ji Qiang and Xiaoye Li. Particle-field decomposition and domain decomposition in parallel particle-in-cell beam dynamics simulation. *Computer Physics Communications*, Vol. 181, No. 12, pp. 2024–2034, 2010.
- [35] George Karypis. METIS and ParMETIS. In David A. Padua, editor, *Encyclopedia of Parallel Computing*, pp. 1117–1124. Springer, 2011.
- [36] K.D. Devine, E.G. Boman, L.A. Riesen, U.V. Catalyurek, and C. Chevalier. Getting started with zoltan: A short tutorial. In *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009. Also available as Sandia National Labs Tech Report SAND2009-0578C.
- [37] C. Chevalier and F. Pellegrini. PT-Scotch: A Tool for Efficient Parallel Graph Ordering. *Parallel Comput.*, Vol. 34, No. 6-8, pp. 318–331, 2008.
- [38] George Karypis and Vipin Kumar. Parallel Multilevel K-way Partitioning Scheme for Irregular Graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, Supercomputing '96*, Washington, DC, USA, 1996. IEEE Computer Society.
- [39] E. Kijispongse and S. U-ruekolan. Dynamic load balancing on GPU clusters for large-scale K-Means clustering. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pp. 346–350, June 2012.
- [40] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.
- [41] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, Vol. 18, No. 9, pp. 509–517, 1975.
- [42] John Dubinski. A parallel tree code. *New Astronomy*, Vol. 1, No. 2, pp. 133–147, 1996.
- [43] Michael Bader. *Space-Filling Curves - An Introduction with Applications in Scientific Computing*, Vol. 9 of *Texts in Computational Science and Engineering*. Springer-Verlag, 2013.
- [44] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *High-Performance Computing, 1997. Proceedings. Fourth International Conference on*, pp. 230–235, December 1997.
- [45] Ji Xu, Huabiao Qi, Xiaojian Fang, Liqiang Lu, Wei Ge, Xiaowei Wang, Ming Xu, Feiguo Chen, Xianfeng He, and Jinghai Li. Quasi-real-time simulation of rotating drum using discrete element method with parallel GPU computing. *Particuology*, Vol. 9, No. 4, pp. 446–450, 2011. Multiscale Modeling and Simulation of Complex Particulate Systems.
- [46] Yuan Tian, Ji Qi, Junjie Lai, Qingguo Zhou, and Lei Yang. A heterogeneous CPU-GPU implementation for discrete elements simulation with multiple GPUs. In *Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA), 2013 International*

- Joint Conference on*, pp. 547–552, November 2013.
- [47] K. L. Johnson. Normal contact of elastic solids-Hertz theory. In *Contact Mechanics*, pp. 84–106. Cambridge University Press, 1985. Cambridge Books Online.
- [48] Stefan Luding. Introduction to discrete element methods : basic of contact force models and how to perform the micro-macro transition to continuum theory. *European Journal of Environmental and Civil Engineering*, Vol. 12, No. 7-8, pp. 785–826, 2008.
- [49] John Vince. *Quaternions for Computer Graphics*. Springer Publishing Company, Incorporated, 2011.
- [50] Jace Miller. An Introduction to Quaternions and their Applications to Rotations in Computer Graphics. Technical report, 2006.
- [51] 越塚誠一. 粒子法シミュレーション：物理ベース CG 入門. 培風館, 2008.
- [52] 大槻敏, 楠見晴重, 松岡俊文. 3次元個別要素法による岩盤斜面の崩壊挙動及び亀裂進展シミュレーション. 土木学会論文集C, Vol. 64, No. 3, pp. 607–615, 2008.
- [53] J. Andreas Bærentzen. Robust Generation of Signed Distance Fields from Triangle Meshes. In *Volume Graphics*, pp. 167–175. Eurographics Association, 2005.
- [54] 越塚誠一, 柴田和也, 室谷 浩平. 粒子法入門. 丸善, 2014.
- [55] J.J. Monaghan. Simulating Free Surface Flows with SPH. *Journal of Computational Physics*, Vol. 110, No. 2, pp. 399–406, 1994.
- [56] 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.
- [57] Ahmad Shakibaenia and Yee-Chung Jin. MPS mesh-free particle method for multiphase flows. *Computer Methods in Applied Mechanics and Engineering*, Vol. 229-232, pp. 13–26, July 2012.
- [58] U Ghia, K.N Ghia, and C.T Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, Vol. 48, No. 3, pp. 387–411, 1982.
- [59] L. Lobovský, E. Botia-Vera, F. Castellana, J. Mas-Soler, and A. Souto-Iglesias. Experimental investigation of dynamic pressure loads during dam break. *Journal of Fluids and Structures*, Vol. 48, pp. 407–434, July 2014.
- [60] 青木尊之. GPU コンピューティングによる大規模シミュレーション. プラズマ・核融合学会論文誌, Vol. 90, No. 12, pp. 755–763, 2014.
- [61] G. S. Grest, B. Dünweg, and K. Kremer. Vectorized link cell Fortran code for molecular dynamics simulations for a large number of particles. *Computer Physics Communications*, Vol. 55, pp. 269–285, October 1989.
- [62] M. Gomez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Dominguez, and

- A. Barreiro. SPHysics-development of a free-surface fluid solver - Part 2: Efficiency and test cases. *Computers and Geosciences*, Vol. 48, No. 0, pp. 300–307, 2012.
- [63] Kevin London, Shirley Moore, Philip Mucci, Keith Seymour, and Richard Luczak. The PAPI Cross-Platform Interface to Hardware Performance Counters. Department of Defense Users' Group Conference Proceedings, pp. 18–21, 2001.
- [64] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM*, Vol. 52, No. 4, pp. 65–76, 2009.
- [65] NVIDIA Visual Profiler. <http://developer.nvidia.com/nvidia-visual-profiler>.
- [66] Thrust Quick Start Guide/ : <http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/ThrustQuickStartGuide.pdf>(2012).
- [67] Optimizing Parallel Reduction in CUDA/ : http://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf(2012).
- [68] 武田隆顕. 大規模粒子シミュレーションデータ可視化ツール zindaiji3 の開発について (宇宙科学情報解析論文誌 第二号). 宇宙航空研究開発機構研究開発報告, Vol. 12, pp. 1–7, March 2013.
- [69] David K. Buck and Aaron A. Collins. POV-Ray - The Persistence of Vision Raytracer.
- [70] H. Sagan and J. Holbrook. *Space-Filling Curves*. Springer-Verlag New York, 1994.
- [71] Hans Sagan. 空間充填曲線とフラクタル. *Space-filling curves*. シュプリンガー・フェアラーク東京, 1998.
- [72] 堀井宏祐, 小泉孝之, 辻内仲好, 三木光範, 日高重助, 折戸啓太. 並列粒子要素法によるバンカーショット解析. 情報処理学会論文誌. 数理モデル化と応用, Vol. 44, No. 14, pp. 91–99, 2003.

研究業績一覧

査読付論文

- (1) 都築怜理, 青木尊之, "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", 情報処理学会主催 2015 年ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, 三美印刷, pp.111-119, 2015 (最優秀論文賞受賞)
- (2) 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 ACS 論文誌, 三美印刷, Vol.6, No.3, pp.82-93, 2013

国際学会発表 (ポスター発表)

- (1) S. Tsuzuki, T. Aoki, "Large-scale granular simulations using Dynamic load balance on a GPU supercomputer" , SC14 Regular, Electronic, and Educational Poster, International Conference for High Performance Computing, Networking, Storage and Analysis 2014 (SC'14), 32, New Orleans, LA, November, 2014, (**Acceptance Rate 39.**%)
- (2) S. Tsuzuki, "Dynamic load balance for a granular simulation using billion particles on GPU supercomputer", The Fourth AICS International Symposium, Computer and Computational Sciences for Exa-scale Computing, 7, Kobe, Japan, December, 2013

国際学会発表 (口頭発表)

- (1) S. Tsuzuki, T. Aoki, "Large-scale Particle-based Simulations for Granular and Fluid Dynamics using Dynamic Load Balance on a GPU supercomputer", Particles 2015, 59, Barcelona, Spain, September, 2015
- (2) S. Tsuzuki, T. Aoki, "Large-scale SPH Simulations using Dynamic Load Balance on a

- GPU Supercomputer", Marine 2015, 452, Roma, Italy, June, 2015
- (3) S. Tsuzuki, T. Aoki, "A large-scale particle simulations using dynamic load balance on GPU supercomputer", 11th world congress on computational mechanics (WCCM), 3355, Barcelona, Spain, July, 2014
 - (4) S. Tsuzuki, T. Aoki, "Large-scale Particle Simulations using Dynamic Load Balance on TSUBAME 2.0 Supercomputer", 5th Asia Pacific Congress on Computational Mechanics (APCOM 2013), 1537, Singapore, December, 2013
 - (5) S. Tsuzuki, T. Aoki, T. Shimokawabe, "A large-scale particle simulations based on dynamic load balance on GPU-rich supercomputer", Particles 2013, 218, Stuttgart, September, 2013
 - (6) S. Tsuzuki, T. Aoki, T. Shimokawabe, X. Wang, "Passive scalar computation of billion particles on GPU supercomputer", 10th world congress on computational mechanics (WCCM), 473, Sao Paulo, Brazil, July, 2012

招待講演

- (1) 都築怜理, "動的領域分割を用いた GPU スパコンにおける流体構造連成の大規模粒子法シミュレーション", 芝浦工業大学第 49 回数理学科談話会, 大宮, 11 月, 2015
- (2) 都築怜理, "GPU スパコンによる動的領域分割を用いたサスペンション・フローの大規模粒子法シミュレーション", 日本応用数学会 2015 年度年会, K1072, 金沢, 9 月, 2015
- (3) 都築怜理, "動的負荷分散による SPH/DEM 大規模粒子法シミュレーション", GTC Japan 2014, 5, 東京, 7 月, 20
- (4) 都築怜理, "近接相互作用に基づく大規模粒子計算 (個別要素法) における効率的な動的負荷分散手法の提案と GPU スパコンでの実装", 日本応用数学会 2013 年度年会, 9080, 福岡, 9 月, 213

国内学会・シンポジウム等における発表

- (1) 都築怜理, 青木尊之, "GPU スパコンにおける動的負荷分散を用いた大規模流体構造連成シミュレーション" 日本計算工学会・第 20 回計算工学講演会, b-5-6, 茨城, 6 月, 2015
- (2) 都築怜理, 青木尊之, "サスペンション・フローの超大規模シミュレーション", 粉体工学会 2015 年度春期研究発表会講演論文集, 一般-16, 東京, 5 月, 2015

- (3) 都築怜理, 青木尊之, "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 回数値流体シンポジウム, 55, 東京, 12 月, 2014
- (4) 都築怜理, 青木尊之, "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 27 回計算力学講演会, 24222, 盛岡, 11 月, 2014
- (5) 都築怜理, 青木尊之, "GPU スパコンによる大規模粒子法 (DEM, SPH) シミュレーション", 第 6 回アクセラレーション技術発表討論会, 沖縄, 6 月, 2014
- (6) 都築怜理, 青木尊之, "GPU による大規模粒子法シミュレーションの実問題への適用", 日本計算工学会, 第 19 回計算工学講演会, c-3-5, 広島, 6 月, 2014
- (7) 都築怜理, 青木尊之, "GPU スパコンを用いたバンカーショットの大規模 DEM 計算", 粉体工学会 2014 年度春期研究発表会講演論文集, 一般-8, 京都, 5 月, 2014
- (8) 都築怜理, 青木尊之, "GPU スパコンを用いた大規模 DEM シミュレーションによる粉体の攪拌解析", 第 26 回計算力学講演会, 1216, 佐賀, 11 月, 2013
- (9) 都築怜理, 青木尊之, "GPU で演算加速された 1 億個の粒子による大規模粉体シミュレーション", 粉体工学会 2013 年度秋期研究発表会講演論文集, 一般-12, 大阪, 10 月, 2013
- (10) 都築怜理, 青木尊之, 下川辺隆史, 小野寺直幸, "GPU スパコンにおける大規模粒子法計算の動的負荷分散", 第 18 回 計算工学講演会, c-1-3, 東京, 6 月, 2013
- (11) 都築怜理, 青木尊之, 下川辺隆史, 小野寺直幸, "GPU による 1 億個の粒子を用いた大規模個別要素法シミュレーション", 第 18 回 計算工学講演会, e-11-1, 東京, 6 月, 2013
- (12) 都築怜理, 青木尊之, 王嫻, 宮下達路, "パッシブ・スカラー粒子の大規模 GPU 計算", 第 17 回 計算工学講演会, c-5-4, 京都, 5 月, 2012
- (13) 都築怜理, 青木尊之, 下川辺隆史, 王嫻, "複数ノードの GPU による大規模パッシブ・スカラー粒子計算の強スケーリングと動的負荷分散", 第 133 回 HPC 研究発表会, 38, 兵庫, 3 月, 2012
- (14) 都築怜理, 青木尊之, 王嫻, "複数 GPU を利用したパッシブ・スカラー粒子計算の高速化", 第 25 回数値流体シンポジウム, 70, 大阪, 12 月, 2011

受賞 (学外, 国内)

- (1) ビジュアル・コンピューティング賞, 日本計算工学会・第 20 回計算工学講演会, 2015 年 6 月 9 日
- (2) 最優秀論文賞, 情報処理学会・2015 年ハイパフォーマンスコンピューティングと計算科学シンポジウム, 2015 年 3 月 28 日

- (3) IEEE Computer Society Japan Chapter 優秀若手研究賞, 情報処理学会, 2015 年 3 月 28 日
- (4) ベスト CFD グラフィックス・アワード動画部門 第 1 位, 日本流体力学会・第 28 回 数値流体シンポジウム, 2014 年 12 月 4 日
- (5) ベストグラフィックスアワード最優秀賞, 日本計算工学会・第 19 回計算工学講演会, 2014 年 6 月 5 日
- (6) 若手優秀講演フェロー賞, 日本機械学会・計算力学部門, 2014 年 5 月 29 日
- (7) コンピュータサイエンス領域奨励賞, 情報処理学会, 2012 年 10 月 11 日

受賞 (学内)

- (1) 最優秀論文発表賞, 東京工業大学大学院 総合理工学研究科 創造エネルギー専攻 修士論文発表会, 2013 年 2 月 12 日
- (2) Best Presentation Award, 東京工業大学大学院 総合理工学研究科 創造エネルギー専攻 修士課程中間発表会, 2012 年 9 月 24 日

職歴

- (1) 日本学術振興会 平成 28 年度 ポスト・ドクトラル特別研究員 (採用内定)
- (2) 日本学術振興会 特別研究員 (DC2) (2014 年 4 月-2016 年 3 月)