Vol.2017-HPC-162 No.15
2017/12/19

連結リストを用いたMPS法における近傍粒子探索の高速化と並列計算機への移植

宮島 敬明[1,a)]　窪田 健一[1]　藤田 直之[1]

概要：Moving Particle Semi-implicit (MPS) 法は流体力学などの分野で利用される粒子系シミュレーションの一種である。計算対象となる物体や流体などの連続体を仮想粒子に分割し、各粒子と近傍粒子との相互作用を元に物理量を算出する。近傍粒子の探索は、MPS 法の演算の主要なボトルネックであり、負荷の低減のためにいくつかのアルゴリズムが提案されている。今回、宇宙航空研究開発機構 (JAXA) が研究開発中の内製 MPS プログラム “p-flow” に連結リストを用いた近傍粒子探索を導入し、並列計算機 (Intel Xeon Skylake-SP, Intel Xeon Phi KNL, NVIDIA Tesla P100(NVlink, PCIe)) へ移植・最適化を行った。本研究報告では、連結リストの導入によるデータ構造の複雑化が処理時間に与える影響、並列計算機に応じた移植と最適化手法、それぞれの処理時間の比較の 3 点について議論する。224,910 粒子の水柱崩壊問題をベンチマークに利用、粒子密度の計算を対象とし、評価には NVIDIA Tesla P100 (PCIe) ×2, Tesla P100 (NVlink) ×2, Intel Xeon Gold 6150 ×2, Intel Xeon Phi 7210 ×1 を利用した。p-flow の場合、連結リストの導入によるデータ構造の複雑化は処理時間に大きな影響を与えなかった。また、並列計算機へ 2 種類の最適化を行った際の処理時間は、それぞれ 7.8[ms]、6.8[ms]、35.1[ms]、104.1[ms] であった。

1. はじめに

MPS 法は大変形を伴う非圧縮性流体のシミュレーションに用いられ、粒子系シミュレーションに分類される [1]。空間を格子状に分割し、各格子が物理量を保持するオイラー的な手法の格子法と異なり、移動する粒子が物理量を持つラグランジュ的な手法である。図 1 に MPS 法による水柱崩壊問題のシミュレーション結果を示す。各粒子の速度などの物理量は近傍粒子との相互作用を元に計算されるため、全粒子がそれぞれの近傍粒子を探索する必要がある。この近傍粒子探索の処理は、ランダムアクセスと間接参照、不定回ループなどを持ち、計算時間の大半を占める。MPS 法の計算精度は使用する粒子数によって大きく左右される。室谷らの京を持ちいた先行研究では、4.0km × 3.5km の津波のシミュレーションに 2 億 6 千万個の粒子を利用している [2]。各粒子が独立して近傍粒子を探索し相互作用を計算するため、並列度は粒子数に比例して増加するが、同時に計算時間も増加する。その結果、計算時間の大半が近傍粒子探索となり、この部分の高速化が MPS 法全体の高速化の鍵となる。MPS 法の GPU への実装はいくつかの先行研究が存在する [5]。また、近傍粒子探索は、N 体問題や分子動力学法などの他の粒子系シミュレーションでも計算時間の大半を占めるため、多くの研究が行われている。

我々は、航空分野への MPS 法の応用を目指し、内製の MPS プログラム “p-flow” を研究・開発中である。オリジナルの p-flow は、Fortran 95 で記述され、Structure of Array(SoA) 形式のデータ構造を持っていた。近傍粒子探索には計算領域を分割した探索手法（バケット法）が用いられていたが、探索は単純なインデックスを用いたものであった。また、MPI によって領域分割が行われているが、OpenMP などを用いたスレッドレベルの並列化は行われていなかった。本研究報告では、データ構造を変更し、さらに並列計算機への移植・最適化を行った際の評価を示す。具体的には、バケット法の探索に Array of Structure(AoS) 形式の連結リストを導入し、それを Intel Xeon Skylake-SP, Intel Xeon Phi KNL, NVIDIA Tesla P100(NVlink, PCIe) へ移植・最適化し、計算時間を比較した。最適化には MPS 法特有の物理背景を考慮することで、メモリアクセスとキャッシュを有効利用した。また、Xeon/Xeon Phi と Tesla P100 には並列度の違いを考慮し、それぞれ異なる最適化を施すこととした。移植・最適化に際しては移植性を優先し、OpenMP と OpenACC を用いた。

1　宇宙航空研究開発 航空技術部門
数値解析技術研究ユニット 計算情報基盤セクション
〒 182-8522 東京都調布市深大寺東町 7-44-1
a)　miyajima.takaaki@jaxa.jp

ⓒ 2017 Information Processing Society of Japan

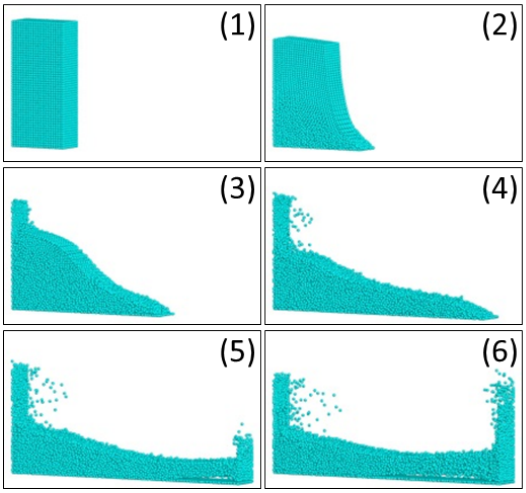

図 1: MPS 法のシミュレーション例：水柱崩壊問題

2. Moving Particle Semi-implicit 法

p-flow は大規模シミュレーションを目的に研究開発されており、陽解法の MPS 法 (Explicit MPS) を採用している。本節では、陽解法の MPS 法について述べる。

2.1 支配方程式

MPS 法は以下の計算ステップを目的のシミュレーション時間に到達するまで繰り返す。なお、$\boldsymbol{r}_j - \boldsymbol{r}_i$ は対象粒子 i と近傍粒子 j との距離計算を示す。

Proc 0) シミュレーションの初期値を設定
Proc 1) 仮速度の計算
Proc 2) 粒子の仮位置を計算
Proc 3) 粒子数密度と圧力の計算
Proc 4) 圧力勾配の計算
Proc 5) 粒子の位置を計算
Proc 6) 次の時間ステップへ（Proc 1~6 を繰り返す）

以下に各計算ステップの概要を示す。

Proc 0) シミュレーションの初期値を設定

MPS 法では初期値として、近傍粒子の距離の重み平均 λ^0 と初期の粒子数密度 n^0、初期の密度 ρ を求める。

$$\lambda^0 = \frac{\sum_{j \neq i'} (|\boldsymbol{r}_j^0 - \boldsymbol{r}_{i'}^0|)^2 \omega(|\boldsymbol{r}_j^0 - \boldsymbol{r}_{i'}^0|)}{\sum_{j \neq i'} \omega(|\boldsymbol{r}_j^0 - \boldsymbol{r}_{i'}^0|)} \tag{1}$$

$$n^0 = \sum_{j \neq i'} \omega(|\boldsymbol{r}_j^0 - \boldsymbol{r}_{i'}^0|) \tag{2}$$

Proc 1) 仮速度の計算

計算ループでは、まず下式を用いて各粒子の仮速度を求める。右辺第 2 項と第 3 項はそれぞれ粘性と重力である。

$$\boldsymbol{u}_i^* = \boldsymbol{u}_i^k + \Delta t \left(\left(\nu \frac{2d}{\lambda^0 n^0} \sum_{j \neq i} (\boldsymbol{u}_j^k - \boldsymbol{u}_i^k) \omega(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right) + g \right) \tag{3}$$

ただし、k はタイムステップ、t は実時間、i と j は粒子の番号、ν は動粘性率、d はシミュレーションの次元数、g は重力加速度、$\boldsymbol{u}_i^k$ は時刻 k での粒子 i の速度、$\boldsymbol{u}_i^*$ は時刻 k での粒子 i の仮速度である。

Proc 2) 粒子の仮位置を計算

続いて、Proc 1 で得られた仮速度を用いて各粒子の仮位置を求める。

$$\boldsymbol{r}_i^* = \boldsymbol{r}_i^k + \Delta t \boldsymbol{u}_i^* \tag{4}$$

ただし、$\boldsymbol{r}_i^k$ は時刻 k での粒子 i の位置である。

Proc 3) 粒子数密度と圧力の計算

その後、各粒子の次のタイムステップの圧力を求める。

$$n_i^* = \sum_{j \neq i} \omega(|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|) \tag{5}$$

$$P_i^{k+1} = c^2 \frac{\rho^0}{n^0} (n_i^* - n^0) \tag{6}$$

ただし、P_i^{k+1} は時刻 k での粒子 i の圧力、c は音速、n_i^* は時刻 k での粒子 i の仮の粒子数密度である。

Proc 4) 圧力勾配の計算

そして、各粒子の圧力から圧力勾配を求める。

$$\langle \nabla P \rangle_i^{k+1} = \frac{d}{n^0} \sum_{j \neq i} \left(\frac{(P_i^{k+1} + P_j^{k+1})(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} \omega_{grad}(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right) \tag{7}$$

ただし、ω_{grad} は後述する重み関数である。

Proc 5) 粒子の位置を計算

最後に、次のステップの最終的な速度と位置を求める。

$$\boldsymbol{u}_i^{k+1} = \boldsymbol{u}_i^* - \Delta t \left(\frac{1}{\rho} \nabla P \right)_i^{k+1} \tag{8}$$

$$\boldsymbol{r}_i^{k+1} = \boldsymbol{r}_i^* - \Delta t \left(\frac{1}{\rho} \nabla P \right)_i^{k+1} \tag{9}$$

ただし、$\boldsymbol{u}_i^{k+1}$ と $\boldsymbol{r}_i^k$ は時刻 $k+1$ での粒子 i の粒子の速度と位置である。

Proc 6) 次の時間ステップへ

次のタイムステップの計算を開始するために、粒子の最大速度 $\boldsymbol{u}$ から実時間 Δt を求める。

2.2 近傍粒子探索

MPS 法は式 5 に代表されるように、1 タイムステップの中で複数回の近傍粒子探索を行う。式 5 では粒子数密度を求めるために、近傍粒子を探索し、見つけた粒子の物理量を対象の粒子に足しこむ。各粒子によって近傍粒子は異なり、それぞれが時々刻々移動するため、毎回計算を行わなければならない。近傍粒子探索は全計算の中で最も高負荷で、Smoothed Particle Hydrodynamics(SPH) 法や N 体問題に類似している。探索を簡単化するために、バケット法や双方向リスト法、ハッシュ法、ブックキーピング法などの手法が提案されている [4] [6] [3]。

p-flow が採用するバケット法は、計算領域をバケットと呼ばれる格子に区切り、隣接するバケット内部の粒子のみを探索の

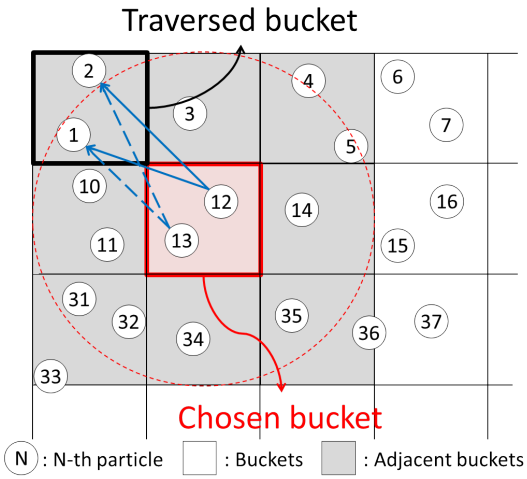

図 2: バケット法を用いた近傍粒子探索：計算領域は矩形のバケットに区切られ、計算対象の粒子（粒子番号 12）は隣接するバケットのみから近傍粒子を探索する

対象とすることで計算量を減少させる手法である。図 2 にバケット法を用いた近傍粒子探索の概要を示す。計算対象の粒子の値は、隣接するバケット内の各近傍粒子の距離に応じた物理量を足しこんで総和することで求められる。近傍粒子探索は探索と距離計算、重みの計算、物理量の総和によって成り、計算領域の全粒子がそれぞれにこの処理を行う。

重みの計算は、式 5 や式 10、式 11 のように 1 タイムステップの中で複数回行われる。図 2 に示す赤い点線の円は影響範囲と呼ばれ、円の内部にある粒子は距離に応じて重みが加えられ、物理量を計算対象の粒子に加算する。重みは距離のみで決定され、式 10 が圧力勾配の計算、式 11 がそれ以外の計算に用いられる

$$\omega_{grad}(r) = \frac{r_e}{r} - \frac{r}{r_e} \tag{10}$$

$$\omega(r) = \frac{r_e}{r} - \frac{r}{r_e} - 2 \tag{11}$$

ただし、r_e は影響範囲の半径、r は粒子 i と j の距離である。

渡辺らは、個別要素法（DEM: Distinct Element Method）におけるバケット法の実装について詳細な報告を行っている [4]。個別要素法は、粉体のシミュレーションを目的に開発され、MPS 法に類似した近傍粒子探索を行う。バケット法の探索に、連結リスト法やハッシュ法を利用した際のメモリ使用量と計算時間を比較し、連結リストがハッシュ法よりも計算時間のうえで優れていることを示した。特に、本研究で評価した数十万粒子の範囲では、12~21%程度の差が出ることが示されている。個別要素法と MPS 法は、各バケットの内部に入る粒子の個数が異なるため（個別要素法では最大 8 個、MPS 法では最大 33 個）、結果をそのまま利用することはできないが重要な先行研究であると考える。なお、本研究では粒子登録法は導入していないため、さらなる高速化が可能な粒子登録法と連結リスト法やハッシュ法を組み合わせる手法については、今後の課題とする。

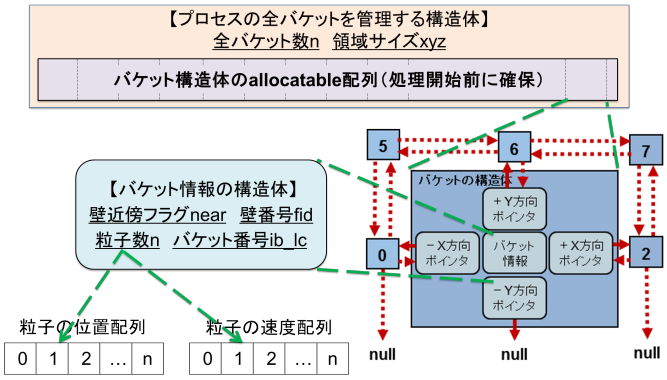

図 3: p-flow のバケット管理用データ構造

3. p-flow: JAXA 内製 MPS プログラム

本節では、JAXA 内製の MPS プログラム “p-flow” について述べる。我々の以前の報告 [7] で対象としていたオリジナルの p-flow は、陽解法の MPS 法を採用して計算負荷を減らすとともに、単純な領域分割によってフラット MPI 化を実現していた。実装は比較的単純なもので、領域全体のバケットを定義する静的配列を各プロセスが保持するメモリ効率の悪いものであった。近傍粒子探索には、当該粒子の所属するバケット番号を元に隣接するバケットの番号を算出していた。これを 3.1 節に示す、双方向リストを利用した各プロセスが必要とするバケットのみで定義された動的配列を保持する形式に変更した。

3.1 バケット管理用データ構造

バケットの管理を行うデータ構造は、省メモリ化のために各プロセスが自身のバケットのみの情報を持ち、ParMetis を利用した動的負荷分散を実現するために、図 3 に示すデータ構造に変更した。

- バケット構造体（図右下）：各バケットの接続関係と付随する情報を保持する構造体
- バケット情報の構造体（図左中）：各バケットの情報を管理する構造体

バケット構造体はバケットの数に応じて動的に確保され、各バケット構造体は連結リストによって他のバケットとの接続関係が保持されている。連結リストの実装はポインタではなく、バケットの番号を利用している。また、付随する情報を保持するバケット情報の構造体も保持している。バケット情報の構造体は、バケット内に存在する粒子の数やインデックス、気流や壁の遠近などの情報を持ち、それぞれが別の静的に確保される構造体となっている。全体として、複数の構造体がネストした Array of Structure (AoS) 形式のデータ構造である。なお、図左下に示す粒子の各物理量は単精度浮動小数点の要素で構成され、粒子の個数分だけの要素数を持った Structure of Array (SoA) 形式のデータ管理用構造体とは独立した配列として宣言されている。物理量は以下の 9 種類である。粒子の位置 r_i、粒子の速度 u_i、粒子の仮速度 u_i^*、粒子の圧力 P_i、粒子の仮位置 r_i^*、圧力勾配 $\langle \nabla P \rangle_i$、粒子数密度 n_i、粒子番号 i。

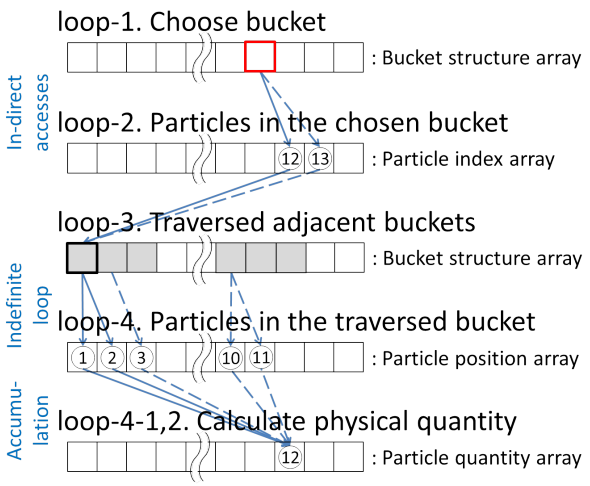

図 4: メモリアクセスパターン

3.2 近傍粒子探索の実装とメモリアクセスパターン

p-flow の近傍粒子探索の実装を粒子数密度の計算（式 5）と図 2 を例に示す。近傍流探索の計算は以下の 4 重ループからなり、最内ループで粒子の距離と重みを元に粒子数密度の計算と足しこみが行われる。他の近傍粒子探索も同様のループ構造を持つが、最内ループでの物理量の計算が異なる。

(1) loop-1: 対象バケットの選択（図 2 における赤いバケット）
(2) loop-2: 対象バケット内の対象粒子の選択（図 2 の粒子 12）
(3) loop-3: 隣接する 3×3×3 のバケットを選択・走査（図 2 では簡単化のために二次元としたため、3×3）
(4) loop-4: 隣接バケット内の粒子を選択し、距離と重み、物理量の計算・足しこみを行う

近傍粒子探索を実装する上で考慮すべき点は、loop-2 で選択された対象バケット内の全ての粒子が loop-3 と loop-4 の計算のために同じ粒子にアクセスするということである。図 2 では、粒子 12 と 13 はまず左上の隣接バケットを選択し、loop-4 に該当するバケット内の粒子 1 と 2 について計算を行う。その計算が終わり次第、上段真ん中のバケットに移る。また、loop-1 と 2,3 は処理に依存性がなく、完全に独立した処理として実行が可能である。loop-4 は足しこみ処理を行うため、単純な並列化では Read-After-Write(RAW) ハザードを起こしてしまうことがあるため、注意が必要である。

図 4 にこのループのメモリアクセスパターンを示す。バケット管理用構造体と各物理量の配列を各ループで間接参照し、それぞれが非連続的なメモリアクセスを行っている。物理量の配列への非連続的なアクセスは、定期的な粒子のソートにより連続的なアクセスへと改善する手法が存在する。loop-4 については、各バケット内の粒子数が異なるため不定数ループとなっているため、そのままではベクトル化が難しい。また、粒子の位置は各タイムステップで変わるため、キャッシュの利用率を上げることも簡単ではない。将来的には、粒子登録法や分子動力学の分野で用いられる Verlet リスト?) などを導入することで、これらの問題を改善することが可能である。

3.3 予備評価

予備評価として、SoA 形式のバケット管理用構造体を持つオリジナルの p-flow (SoA p-flow) と AoS 形式のバケット管理用構造体を持つ本稿の p-flow(AoS p-flow) の計算時間を比較した。評価環境は、2 ソケットの Intel Xeon Gold 6150 @ 2.7GHz（各 18 物理コア、36 論理スレッド）と、192GB の DDR4-2400 メモリを搭載したシングルノードのマシンである。データセットは、水柱崩壊問題を 70×70×14 バケットで 224,910 個の粒子を用いて解くものである。時間計測には *MPI_Wtime* 関数を用い、最初の 200 タイムステップの平均をとった。MPI プロセス数は 72 とした。なお、AoS p-flow は動的負荷分散を行わない設定とした。

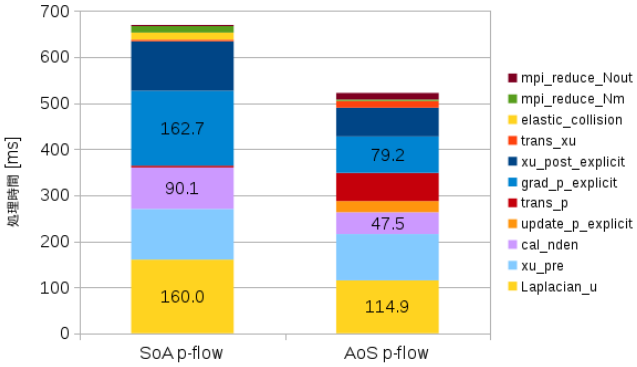

図 5: SoA p-flow と AoS p-flow のプロファイル結果

図 5 に SoA p-flow と AoS p-flow のプロファイル結果を示す。本研究報告の対象である、AoS p-flow では Laplacian_u と cal_nden, grad_p_explicit の 3 つのサブルーチンの合計の計算時間が全計算時間の 58%と 43%を占めていた。それぞれ、計算ステップ 1 と 4、3 に相当し、内部で近傍粒子探索を行っている。これら 3 つ近傍粒子探索を行うサブルーチンは SoA p-flow よりも AoS p-flow の方が、それぞれ 28.2%, 47.3%, 51.3%ほど短時間で処理が完了している。また、4 節で評価に利用した cal_nden サブルーチンの近傍粒子探索の処理だけを比較すると、SoA p-flow で 53.2[ms]、AoS p-flow で 47.0[ms] であり、AoS の方が 11.7%ほど短時間で処理が完了している。また、プロセス間で粒子のやり取りを行うサブルーチン xu_pre, xu_post_explicit は全計算時間の 30%と 29%を占めていた。

4. 最適化手法

AoS p-flow の近傍粒子探索を CPU（Xeon/Xeon Phi）と GPU（Tesla P100）で実装するにあたり、メモリアクセスパターンと各アーキテクチャの構造を考慮した 2 種類の実装を行った。それぞれ「1 スレッド / 1 バケット」と「1 スレッドブロック / 1 バケット」と呼び、詳細を以下に述べる。実装に際して、CPU には OpenMP、GPU には OpenACC を利用した。

4.1 1 バケットの計算に必要なデータサイズ

各バケットが必要とするデータサイズは以下の通りとなる。1 バケットが選択・走査する隣接バケットは 27（= 3×3×3）個

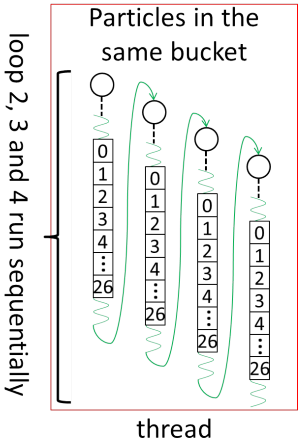

図 6: 1 スレッド / 1 バケット

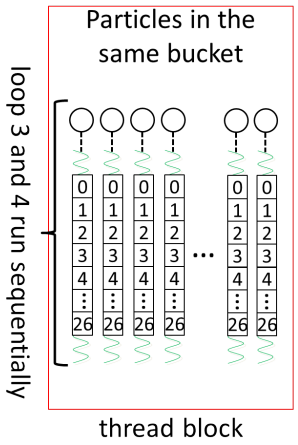

図 7: 1 スレッドブロック / 1 バケット

で、各バケットは最大 33 個の粒子を持つ。つまり、近傍粒子の最大個数は 891（= 33×27）個となる。粒子の各物理量は単精度浮動小数点形式で表されるため、3,564 （= 891×4）bytes となる。粒子数密度の計算には 4 つの物理量（x,y,z 座標と粒子数密度）が必要なため、物理量には 14,256（= 4×3,564）bytes が必要となる。各バケット構造体は 72 bytes なので、隣接バケットの情報に 1,944（ = 27×72）bytes が必要となる。これらを合計した、16,200（= 14,256+1,944） bytes のキャッシュ容量が各スレッドあたりに必要となる。バケットの選択・走査とバケット内の粒子へのアクセスは、スレッドあたりのキャッシュサイズが 16,200 bytes より大きければ初回のメモリアクセスを除いて全てキャッシュされる。これは最大データサイズであり、実際には粒子数が 33 よりも少ないケースがあるため、データサイズは 16,200 bytes よりも小さくてすむと考えられる。

4.2 最適化 1: 1 スレッド / 1 バケット

本最適化では、図 6 に示すように各バケット（loop-1 の 1 イテレーション）を CPU/GPU の各スレッドに割り当てる。loop-2~4 は各スレッドによって逐次的に処理される。表 1 に示すとおり、CPU の場合は各スレッドあたりのキャッシュサイズは、16,200 bytes より大きい。しかし、Tesla P100 の場合、L1 データキャッシュは 32 CUDA スレッドで共有されるため、CUDA スレッドあたりのキャッシュサイズは 445 bytes である。これは必要なキャッシュサイズを満たさない。また、全 CUDA スレッドが処理を行うため Occupancy は高いが、スレッドブロック内の CUDA スレッドの処理が不均衡になるため、同期待ちのコストが大きくなる。

CPU については、OpenMP の “!$omp parallel do”ディレクティブを loop-1 に追加した。また、“schedule”節を追加することで、バケット間で異なる粒子数による負荷の不均衡を分散した。5 節では 3 つのスケジューリングポリシー（static, dynamic, guided）について評価を示す。GPU については、OpenACC の “!$acc parallel loop gang vector”ディレクティブを loop-1 に追加した。

4.3 最適化 2: 1 スレッドブロック / 1 バケット

本最適化では、図 7 に示すように各バケットを GPU の各スレッドブロックに割り当てる。loop-2 の同じバケットの各粒子が同じスレッドブロックの CUDA スレッドに割り当てられる。loop-3 と 4 は各 CUDA スレッドにより逐次的に実行される。これにより、各 CUDA スレッドは同一の隣接バケット内の粒子にアクセスするため、スレッドブロック内での処理のバラツキと同期待ちはなくなる。また、スレッドブロック内で物理量を共有することができる。4.2 節で述べたように各バケットは 16,200 bytes のキャッシュサイズが必要であるが、物理用を共有することができるため、Tesla P100 の L1 キャッシュ（24KB）に全データを保持することが可能である。もし物理量がメモリ上で連続的に配置されていれば、Tesla P100 は 256bit でデータをフェッチするため、8 つ分の物理量を同時にキャッシュすることができる。なお、フェッチの回数は低減されるが、コアレスドアクセスにはなっていないため、メモリバンド幅の利用率は低いままであると考えられる。また、1 バケットの内部に存在する粒子の数は最大で 33 個であり、平均すると 10 個程度であるため、スレッドブロックサイズを 32 に設定するべきであると考えた。バケット内の粒子が最大の 33 であった場合、ブロックサイズが 32 であればワープは 2 回実行されるが、ブロックサイズが 64 であれば 1 回の実行で完了する。5 節では、スレッドブロックサイズを 16,32,64,128 にした場合の評価を示す。

GPU については、OpenACC の “!$acc parallel vector_length(N)”と “!$acc loop gang” ディレクティブを loop-1 に、“!!$acc loop vector” ディレクティブを loop-2 に追加した。また、“!$acc loop seq” を loop-4 に追加し、逐次処理とした。“vector_length(N)”句はスレッドブロックサイズを変更するためのものである。CPU に本最適化と同様の最適化を行うべく、最内ループに “!$omp simd” ディレクティブを追加したが、SIMD 化されなかったため、今後の研究課題とする。

5. 評価

本節では、4 節で提案した最適化手法を Intel Xeon Skylake-SP, Intel Xeon Phi KNL, NVIDIA Tesla P100(NVlink, PCIe) に適用した際の評価を示す。まず最適化に関して Xeon/Xeon Phi の評価と考察を行い、次に P100(NVlink, PCIe) の評価と考察を行う。最後に Xeon/Xeon Phi と P100(NVlink, PCIe) の比較を行う。評価には次の 4 種類の計算ノードを利用した、Intel Xeon Gold 6150, Xeon Phi KNL 7210, NVIDIA Tesla P100 (PCIe), P100 (NVlink)。Xeon Phi を除き、全ての計算ノードが 2 つの CPU と GPU を搭載している。表 1 に各計算ノードの詳細を記す。Xeon と Xeon Phi には Intel Fortran compiler ver.17.0.4 を用い、コンパイルオプションは “-qopenmp -O3 -xCOMMON-AVX512 -fp-model fast=2”を使用した。また、スレッドのアフィニティは “granularity=fine,compact”とした。Xeon Phi のメモリモードは Flat-Quadrant とし、“numactl –membind=1”コマンドにより MCDRAM を明示的に利用した。MPI ライブラリには IntelMPI 5.0.2 を用い、コンパイルオプションは “-l -n 2”とした。Tesla P100 (NVlink) と P100 (PCIe)

には PGI Fortran compiler ver.17.7 を用いた、コンパイルオプションは “-acc -ta=nvidia:cc60, fastmath” とした。また、MPI ライブラリには OpenMPI 1.10.5 を用い、コンパイルオプションは “-bind-to socket -npersocket 1 -n 2” とした。ホスト CPU は、Tesla P100 (NVlink) が Intel Xeon E5-2695v4[*1]、P100 (PCIe) が Intel Xeon Gold 6150 である。時間計測とデータセットは、3.3 節と同様である。なお、Tesla P100(NVlink,PCIe) については CPU-GPU のデータ転送は除外している。

表 1: 各計算ノードのスペック

計算ノード	単精度性能 [TFLOPS]	周波数 [GHz]
Gold 6150	N/A	2.7
KNL 7210	5.3	1.3
P100 (PCIe)	9.3	1.1
P100 (NVlink)	10.6	1.3
スレッド数	L1 キャッシュ [KB]	メモリ帯域 [GB/s]
36	512	256
256	32	480
3,584	24	732
3,584	24	732

5.1 Xeon と Xeon Phi

表 2 に 4 節の最適化 1 を適用し、ループのスケジューリング・ポリシーを変更した際の評価を示す。表中の “KNL 7210 (Single)” は KNL 7210 を 1 ノード利用して時間計測した実際値、“KNL 7210 (Dual†)” は KNL 7210 を 2 ノード利用した際に得られるであろう計算時間の推測値を示す。なお、“KNL 7210 (Dual†)” の値は、“KNL 7210 (Sinlge)” の半分の値とした。これは、我々の以前の報告 [7] から得られた、近傍粒子探索の部分に通信が存在しないこと、計算時間が粒子数にほぼ比例することを考慮したものである。“Gold 6150 (Dual)” は、Xeon Gold 6150 を 2 ノード利用して時間計測した実際値である。Xeon/Xeon Phi のどちらについてもスケジューリング・ポリシーは “dynamic”（チャンクサイズ=1）が最も優れていた。Xeon Gold 6150 の場合は “dynamic” が “guided” よりも 14.8%速く、KNL 7210 の場合は 36.1%速かった。つまり、KNL 7210 の方が負荷分散により敏感であると考えられる。また、どちらのプロセッサについても “static” は “guided” よりも約 3%高速であった。

ハイブリッド並列について考察を行う。“Hybrid+dynamic” と “Flat” を比較すると、KNL 7210 の場合はどちらの設定でも計算時間が一緒であったが、Xeon Gold 6150 の場合は “Hybrid+dynamic” の方が “Flat” よりも 36%ほど高速であった。更に、“Hybrid+static” が “Flat” よりも 3.9%ほど高速であった。1 ノード内で領域を 72 個（プロセス）に細かく分割した “Flat” では、OpenMP の “static” スケジューリングと似た効果が得られたと考える。本研究報告の対象から外れるため詳しく述べないが、多数のプロセスによって領域分割を行うとプロセス間通信の所要時間が大幅に増加してしまうため、計算時間全体には悪影響を与えてしまう。Xeon Gold 6150(Dual) と KNL 7210(Dual) を比較した場合、Xeon Gold 6150 が KNL 7210 よりも 1.48 倍ほど高速である。

[*1] 東京大学 情報基盤センターの Reedbush スーパーコンピュータシステムを利用した。

表 2: スケジューリングのポリシーと計算時間 [ms]

並列化手法	Hybrid	Hybrid	Hybrid	Flat
ポリシー	dynamic	static	guided	N/A
プロセス数	2	2	2	72 or 256
スレッド数	36	36	36	N/A
KNL 7210 (Single)	104.1	117.2	121.1	105.4
KNL 7210 (Dual†)	52.1	58.6	60.6	52.7
Gold 6150 (Dual)	35.1	46.0	47.8	47.8

5.2 Tesla P100 (NVlink, PCIe)

表 3 に 4 節の最適化 1,2 を適用し、ブロックサイズを変更した際の評価を示す。最適化 1 は最適化 2 よりも約 7 倍ほど計算時間が長い。最適化 2 のブロックサイズは、64 の方が 32 よりもわずかながら高速であった。節で述べたように、バケット内の粒子数は 33 以下かつ平均で 10 個程度であったため、ブロックサイズが 32 の方が 0.2%ではあるが高速であることは予想外であった。MPS 法は時々刻々と粒子の位置が変わり、データセットによってそのパターンもことなるため、水柱崩壊問題の最初に 200 ステップに限った結果である可能性もある。詳しい解析は今後の課題とする。

表 3: 最適化手法とブロックサイズ、計算時間 [ms]

最適化手法	最適化 1	最適化 2			
ブロックサイズ	64	16	32	64	128
P100 (PCIe, Dual)	59.3	13.4	7.82	7.80	8.3
P100 (NVlink, Dual)	52.8	11.6	6.79	6.77	7.2

5.3 Xeon と Xeon Phi、Tesla P100 の比較

図 8 に各計算ノードの計算時間を比較したグラフを示す。P100 (NVlink) が最も高速であり、Xeon Gold 6150 の 5.2 倍、KNL 7210（推測値）の 7.8 倍ほど高速である。Xeon と KNL 7210 のどちらもベクトル化を行えていないため、P100 と同様な単に多数のスレッドを持った計算ユニットとして利用しているにすぎない。最内ループを単にベクトル化しても、粒子の物理量へのランダムアクセス (gather/scatter) がボトルネックとなることが予想される。

6. 結論

本研究報告では、JAXA 内製の陽解法 MPS プログラム “p-flow” の近傍粒子探索を対象に 2 種類の最適化を提案し、Intel Xeon Skylake-SP, Intel Xeon Phi KNL, NVIDIA Tesla P100(NVlink, PCIe) で実装・評価を行った。各ハードウェア構成を考慮し、Xeon/Xeon Phi には「1 スレッド / 1 バケット」を Tesla P100 には「1 スレッドブロック / 1 バケット」を適用した。評価には粒子数が 224,910 個の水柱崩壊問題を用いた。各

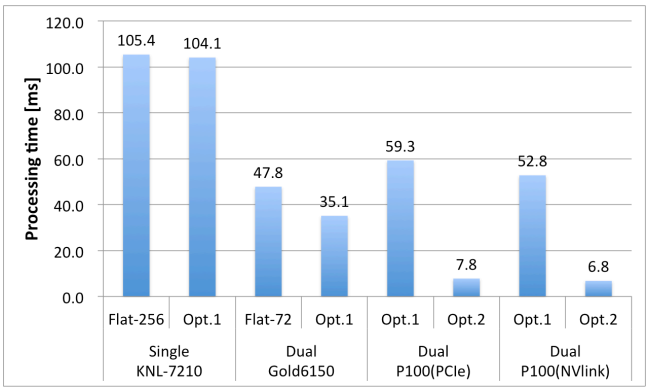

図 8: 各計算ノードの最速の値の比較

計算ノードの計算時間は、Intel Xeon Skylake-SP が 104.1[ms]、Intel Xeon Phi KNL が 35.1[ms]、NVIDIA Tesla P100(NVlink) が 6.8[ms]、NVIDIA Tesla P100(PCIe) が 7.8[ms] であった。

今後の課題として、提案した最適化の詳細な解析と Xeon/Xeon Phi の SIMD ユニットを利用するベクトル化手法の提案が挙げられる。また、OpenACC を利用した今回の最適化と CUDA を利用したものとの比較も行いたい。

謝辞

本研究は JSPS 科研費 16K16064 の助成を受けたものです。

参考文献

[1] Koshizuka, S. and Oka, Y.: Moving particle semi-implicit method for fragmentation of incompressible fluid, *Nuclear Science and Engineering*, Vol. 123, pp. 421–434 (1996).

[2] Murotani, K., Koshizuka, S., Tamai, T., Shibata, K., Mitsume, N., Yoshimura, S., Tanaka, S., Hasegawa, K., Nagai, E. and Fujisawa, T.: Development of Hierarchical Domain Decomposition Explicit MPS Method and Application to Large-scale Tsunami Analysis with Floating Objects, *Journal of Advanced Simulation in Science and Engineering*, Vol. 1, No. 1, pp. 16–35 (online), DOI: 10.15748/jasse.1.16 (2014).

[3] Murotani, K., Masaie, I., Matsunaga, T., Koshizuka, S., Shioya, R., Ogino, M. and Fujisawa, T.: Performance improvements of differential operators code for MPS method on GPU, *Computational Particle Mechanics*, Vol. 2, No. 3, pp. 261–272 (online), DOI: 10.1007/s40571-015-0059-2 (2015).

[4] Seiya, W., Takayuki, A., Satori, T. and Takashi, S.: Neighbor-particle Searching Method for Particle Simulation Based on Contact Interaction Model for GPU Computing, *IPSJ Transactions on Advanced Computing Systems*, Vol. 8, No. 4, pp. 50–60 (2015).

[5] Sota, Y., Watanabe, A. and Kojima, T.: Accerelation of the moving paricle semi-implicit method through multi-GPU parallel computing with dynamic domain decomposition, *Journal of Japan Society of Civil Engineers, Ser. A2 (Applied Mechanics (AM))*, Vol. 69, No. 2 (2013).

[6] Sun, H., Tian, Y., Zhang, Y., Wu, J., Wang, S., Yang, Q. and Zhou, Q.: A Special Sorting Method for Neighbor Search Procedure in Smoothed Particle Hydrodynamics on GPUs, *Parallel Processing Workshops (ICPPW), 2015 44th International Conference on*, pp. 81–85 (online), DOI: 10.1109/ICPPW.2015.46 (2015).

[7] 宮島敬明，藤田直之：OpenACC を用いた MPS 法の近傍粒子探索の最適化，技術報告 19，宇宙航空研究開発 航空技術部門 数値解析技術研究ユニット計算情報基盤セクション (2017).