

# グローバル-ローカル ID マップアルゴリズムの数式解説 (atom\_map.cpp)

Open DEM Japan

2025年6月30日

分散メモリ並列計算において、各粒子にはシミュレーション全体で一意的な **グローバル ID**  $t \in \mathbb{N}$  が割り当てられる。各 MPI プロセスは **所有粒子**  $N_{\text{own}}$  個と **ゴースト粒子**  $N_{\text{ghost}}$  個を保持し、自プロセスの **ローカル添字**  $i \in \{0, \dots, N_{\text{own}} + N_{\text{ghost}} - 1\}$  で管理する。グローバル ID からローカル添字への写像

$$m : \mathbb{N} \rightarrow \{-1, 0, 1, \dots, N_{\text{own}} + N_{\text{ghost}} - 1\}, \quad m(t) = \begin{cases} i & (t \text{ が粒子 } i \text{ の ID のとき}) \\ -1 & (\text{対応粒子なし}) \end{cases} \quad (1)$$

を高速に構築・検索することが本ファイルの目的である。

## 1. 配列方式 (map\_style = 1)

全粒子の最大 ID を

$$T_{\text{max}} = \max_{0 \leq i < N_{\text{own}}} t_i \quad (2)$$

とおく。長さ  $T_{\text{max}} + 1$  の整数配列  $A = (A(0), A(1), \dots, A(T_{\text{max}}))$  を確保し、初期値を

$$A(k) = -1 \quad (0 \leq k \leq T_{\text{max}}) \quad (3)$$

とする。所有粒子とゴースト粒子を合わせた総数  $N_{\text{all}} = N_{\text{own}} + N_{\text{ghost}}$  を逆順に走査し、

$$A(t_i) \leftarrow i, \quad \forall i = N_{\text{all}} - 1, \dots, 0 \quad (4)$$

と代入することで写像 (1) を得る。逆順走査は「近いゴースト粒子より遠いゴースト粒子を上書き」、さらに「ゴースト粒子より所有粒子を上書き」するため、接触検索などで **最寄りの** 実体を優先的に取得できる。配列方式では

$$\text{検索計算量} = O(1), \quad \text{メモリ使用量} = O(T_{\text{max}}) \quad (5)$$

である。

## 2. ハッシュ方式 (map\_style = 2)

MB 式によりハッシュ表サイズを

$$M = 2 \max(N_{\text{own}} + N_{\text{ghost}}, N_{\text{avg}}), \quad M \geq 1000, \quad (6)$$

ただし  $N_{\text{avg}} := \lceil N_{\text{total}}/P \rceil$  (全粒子数をプロセス数  $P$  で割った平均) と定める。連鎖法を用いるオープンハッシュであり、バケット数  $B$  は

$$B = p, \quad p > M \text{ を満たす最小の素数} \quad (7)$$

とする。素数探索は、

$$p = n + 1 + \delta, \quad \delta = \min\{2k + 1 \mid (n + 1 + 2k + 1) \text{ が素数}\}, \quad (8)$$

により偶数回避と平方根試験で求められる (max 32-bit 符号付整数まで)。

ハッシュ関数は剰余

$$h(t) = t \bmod B, \quad (9)$$

エントリは  $(t, m(t))$  の組を持つ。衝突時は単方向リンクリストを辿り、存在すればローカル添字を書き換え、存在しなければ空きリストの先頭を新規ノードに割当てて末尾に接続する。写像構築の計算量は

$$O(N_{\text{all}}) \quad (\text{平均一定個の衝突探索}), \quad (10)$$

検索も期待  $O(1)$  である。メモリは

$$O(B + M) = O(M) \quad (11)$$

となり、配列方式より粒子 ID が疎な場合に有利である。

### 3. 再初期化判定

粒子数がハッシュ容量を超えたとき

$$N_{\text{all}} > M \implies \text{再初期化} \quad (12)$$

を行い、(6)–(8) を再評価してハッシュ表全体を再構築する。

### 4. 同一 ID 検出

各粒子  $i$  について

$$s(i) = \begin{cases} j & (j < i \wedge t_j = t_i) \\ -1 & (\text{他に同一 ID 無し}) \end{cases} \quad (13)$$

を保存し、同一 ID の重複を高速検出する。配列・ハッシュ両方式において写像構築と同時に  $s(i) \leftarrow m(t_i)$  を実施すれば追加コストはない。

以上により、atom\_map.cpp は粒子 ID の疎密に応じて (5)–(11) の複雑度をもつ 2 種の写像実装を切替え、バランスよくメモリと計算量を最適化していることがわかる。