

Sum-CRCW PRAM における 超高速並列ソーティングアルゴリズム

安藤 謙友¹ 和田 幸一¹

¹ 法政大学理工学部応用情報工学科

概要

ソーティング問題を解くアルゴリズムとしてヒストグラム計算と接頭辞和計算からなる H-P ソートが知られている。本研究では Sum-CRCW PRAM における H-P ソートを基にしたソーティングアルゴリズムとして、計算時間のオーダを変えずにプロセッサ数を削減する。また、プロセッサ数のオーダを変えずに計算時間を削減するソーティングアルゴリズム設計手法を示す。その結果、 $2 \leq i \leq \log^* n$ を満たす整数 i について、 $O(\frac{n \log^{(i)} n}{i})$ プロセッサを用いて $O(i)$ 時間で計算可能なアルゴリズムと、 $O(\frac{n}{\log^{(i)} n + i})$ プロセッサを用いて $O(\log^{(i)} n + i)$ 時間で計算可能なアルゴリズムを示す。

1 はじめに

PRAM は並列に動作可能な複数のプロセッサと各プロセッサがアクセス可能な共有メモリからなる並列計算モデルである。^[1,30] このモデルは複数のプロセッサが同一番地の共有メモリに対して同時に読み出しが可能かどうか、同時に書き込みが可能かどうかで細分化されており、CRCW (Concurrent Read, Concurrent Write) PRAM は、同時に読み出しも書き込みも可能なモデルである。CRCW PRAM において、同時書き込みの際に書き込まれた値の和を格納するモデルである Sum-CRCW PRAM を本研究では扱う。

Sum-CRCW PRAM におけるソーティング問題を解くアルゴリズムは次のような研究がされている。Eisenstat は $O(\frac{n}{\log^* n})$ プロセッサを用いて $O(\log^* n)$ 時間で計算可能なアルゴリズムを示した。^[2] これはコスト最適なアルゴリズムの中では現在最速のアルゴリズムである。また、Frei, Wada は接頭辞和計算について、 $O(n \log n)$ プロセッサを用いて $O(1)$ 時間で計算可能なアルゴリズムを示しており、これによって、H-P ソートは同じ計算量になることが示される。本研究では、[3] のアルゴリ

ズムを計算時間のオーダを変えずにプロセッサ数を削減する方法を示す。また、プロセッサ数を増やさずに計算時間を削減する方法を示す。これらによって、 $2 \leq i \leq \log^* n$ を満たす整数 i について、 $O(\frac{n \log^{(i)} n}{i})$ プロセッサを用いて $O(i)$ 時間で計算可能なアルゴリズムと、 $O(\frac{n}{\log^{(i)} n + i})$ プロセッサを用いて $O(\log^{(i)} n + i)$ 時間で計算可能なアルゴリズムを示す。

2 準備

2.1 H-P ソート

H-P ソートはヒストグラム計算と接頭辞和計算からなるソーティングアルゴリズムである。^[2] ここで要素 $a[0], a[1], \dots, a[n-1]$ を持つ長さ n の配列 a に対する接頭辞和配列 s の k 番目の要素は次のように定義される。

$$s[k] = \sum_{0 \leq i \leq k} a[i] \quad (0 \leq k < n)$$

n 要素の入力配列 a に対して H-P ソートは以下の 4 ステップからなる。

- (1) a のヒストグラム $h1$ を計算する
- (2) $h1$ の接頭辞和 $s1$ を計算する
- (3) $s1$ のヒストグラム $h2$ を計算する

(4) $h2$ の接頭辞和 $s2$ を計算する

これにより計算した $s2$ が a のソートされた配列となる。Sum-CRCW PRAM において、ヒストグラム計算は入力配列のサイズに等しいオーダーのプロセッサ数があれば定数時間で計算可能であるため、H-P ソートで考慮すべき点は接頭辞和の計算量である。ここで $|h1|$ と $|h2|$ についてヒストグラム計算に必要な配列サイズを考えると、 $|h1|$ は a の要素の最大値と最小値の差となり、 $|h2|$ は $s1$ において現れる数が高々 n であることから $h1$ と同様に高々 n となる。よって a の要素の最大値と最小値の差を d とすると、(2) と (4) では入力サイズが $O(n + d)$ である接頭辞和計算を考えればよく、この接頭辞和計算に必要なプロセッサ数と計算時間が H-P ソート全体の計算量と等しくなる。よって、以降では接頭辞和計算の計算量について考える。

2.2 接頭辞和に関する結果

本研究で扱う接頭辞和計算に関する補題を示す。

補題 1. ^[3] 要素数 k の接頭辞和は Sum-CRCW PRAM 上で $O(k \log k)$ プロセッサを用いて $O(1)$ 時間で計算できる。

また、PRAM のアルゴリズムに対するプロセッサ数を調整する手段として、Brent の定理から得られる次の補題を扱う。

補題 2. ^[4] プロセッサ数 $O(p)$ を用いて $O(t)$ 時間で計算できる PRAM のアルゴリズムが存在するならば、 $p' < p$ を満たす p' について $O(p')$ プロセッサを用いて $O(\frac{pt}{p'})$ 時間で計算できる PRAM のアルゴリズムが存在する。

補題 1 と補題 2 から次の補題を得る。

補題 3. 要素数 k の接頭辞和は Sum-CRCW PRAM 上で $O(k)$ プロセッサを用いて $O(\log k)$ 時間で計算できる。

3 提案手法

3.1 アルゴリズムの設計と解析

本研究で用いる接頭辞和計算の基本となるアルゴリズムは [2] や [3] でも用いられていた、配列を分割して子問題に分ける手法である。長さ n の配列 a に対して d 分割するアルゴリズム A_0 は以下の 5 ステップからなる。以下、 n は d で割り切れるとする。

- (1) a を要素数が d であるような新たな $\frac{n}{d}$ 個の配列 $b_0, b_1, \dots, b_{\frac{n}{d}-1}$ に順序を保って分割する
- (2) $b_0, b_1, \dots, b_{\frac{n}{d}-1}$ それぞれの接頭辞和である $s_0, s_1, \dots, s_{\frac{n}{d}-1}$ を計算する
- (3) $s_0, s_1, \dots, s_{\frac{n}{d}-1}$ の末尾の接頭辞和である長さ $\frac{n}{d}$ の配列 s' を計算する
- (4) $i \neq 0$ である s_i の各要素に $s'[i-1]$ を加算する
- (5) $s_0, s_1, \dots, s_{\frac{n}{d}-1}$ をその順で結合する

アルゴリズム A_0 の各ステップについて、より詳細な動作を図 1 に示す。

$$\begin{aligned}
 & [a_0, a_1, \dots, a_{d-1}, a_d, \dots, a_{2d-1}, \dots, a_{n-1}] \\
 (1) & \left[\begin{matrix} b_0 \\ a_0, a_1, \dots, a_{d-1} \end{matrix} \right] \left[\begin{matrix} b_1 \\ a_d, a_{d+1}, \dots, a_{2d-1} \end{matrix} \right] \dots \left[\begin{matrix} b_{\frac{n}{d}-1} \\ a_{n-d}, \dots, a_{n-1} \end{matrix} \right] \\
 (2) & \left[\begin{matrix} s_0 \\ a_0, a_0 + a_1, \dots, \sum_{i=0}^{i=d-1} a_i \end{matrix} \right] \left[\begin{matrix} s_1 \\ a_d, a_d + a_{d+1}, \dots, \sum_{i=d}^{i=2d-1} a_i \end{matrix} \right] \dots \\
 (3) & \left[\begin{matrix} s' \\ \sum_{i=0}^{i=d-1} a_i, \sum_{i=0}^{i=2d-1} a_i, \dots, \sum_{i=0}^{i=n-d-1} a_i, \sum_{i=0}^{i=n-1} a_i \end{matrix} \right] \\
 (4) & \left[\begin{matrix} s_0 \\ a_0, a_0 + a_1, \dots, \sum_{i=0}^{i=d-1} a_i \end{matrix} \right] \left[\begin{matrix} s_1 \\ \sum_{i=0}^{i=d} a_i, \sum_{i=0}^{i=d+1} a_i, \dots, \sum_{i=0}^{i=2d-1} a_i \end{matrix} \right] \dots \\
 (5) & [a_0, a_0 + a_1, \dots, \sum_{i=0}^{i=d-1} a_i, \sum_{i=0}^{i=d} a_i, \dots, \sum_{i=0}^{i=2d-1} a_i, \dots, \sum_{i=0}^{i=n-1} a_i]
 \end{aligned}$$

図 1 アルゴリズム A_0

n 要素に対するアルゴリズム A_0 について、(1) と (5) は定数時間で計算可能である。(2) において d 要素の接頭辞和の 1 つを $P_0^{(2)}(d)$ プロセッサを用いて $T_0^{(2)}(d)$ 時間で計算できるとし、(3) における $\frac{n}{d}$ 要素の接頭辞和を $P_0^{(3)}(\frac{n}{d})$ プロセッサを用いて $T_0^{(3)}(\frac{n}{d})$ 時間で計算できるとする。(4) は $O(n)$ プロセッサを用いて $O(1)$ 時間で計算可能であるため、 A_0 が用いるプロセッサ数を P_0 、計算時間を T_0 とすると次

のような計算量となる.

$$\begin{aligned} P_0(n) &= \frac{n}{d}P_0^{(2)}(d) + P_0^{(3)}\left(\frac{n}{d}\right) + O(n) \\ T_0(n) &= T_0^{(2)}(d) + T_0^{(3)}\left(\frac{n}{d}\right) + O(1) \end{aligned}$$

次に (2) の接頭辞和計算に A_0 を用いたアルゴリズム A_1 を考える. A_1 が用いるプロセッサ数を P_1 , 計算時間を T_1 とすると次のような計算量となる.

$$\begin{aligned} P_1(n) &= \frac{n}{d}P_0(d) + P_0^{(3)}\left(\frac{n}{d}\right) + O(n) \\ T_1(n) &= T_0(d) + T_0^{(3)}\left(\frac{n}{d}\right) + O(1) \end{aligned}$$

さらに $i \geq 2$ について, A_i が用いるプロセッサ数を P_i , 計算時間を T_i とし, (2) の接頭辞和計算に A_{i-1} を用いたアルゴリズム A_i のアルゴリズムは次のような計算量となる.

$$\begin{aligned} P_i(n) &= \frac{n}{d}P_{i-1}(d) + P_0^{(3)}\left(\frac{n}{d}\right) + O(n) \\ T_i(n) &= T_{i-1}(d) + T_0^{(3)}\left(\frac{n}{d}\right) + O(1) \end{aligned}$$

ここで $d = \lfloor \log n \rfloor$ とし, A_0 の (2) と (3) の計算に補題 1 のアルゴリズムを適用すると, 各アルゴリズムは次のような計算量となる.

$$\begin{aligned} P_0(n) &= O(n \log \log n), \quad T_0(n) = O(1) \\ P_1(n) &= O(n \log \log \log n), \quad T_1(n) = O(1) \\ &\vdots \\ P_{i-2}(n) &= O(n \log^{(i)} n), \quad T_{i-2}(n) = O(i) \end{aligned}$$

補題 2 を用いると, $2 \leq i \leq \log^* n$ について, $O(\frac{n \log^{(i)} n}{i})$ プロセッサを用いて $O(i)$ 時間で計算可能なアルゴリズムが得られる. この結果は特に $i = \log^* n$ のとき $O(\frac{n}{\log^* n})$ プロセッサを用いて $O(\log^* n)$ 時間で計算可能なアルゴリズムを得る. これにより, 計算時間のオーダを変えずにプロセッサ数を削減した.

同様に $d = \lfloor \log n \rfloor$ とし, A_0 の (2) の計算を補題 3 のアルゴリズム, (3) の計算を補題 1 のアルゴリズムを適用すると次のような計算量となる.

$$\begin{aligned} P_0(n) &= O(n), \quad T_0(n) = O(\log \log n) \\ P_1(n) &= O(n), \quad T_1(n) = O(\log \log \log n) \\ &\vdots \\ P_{i-2}(n) &= O(n), \quad T_{i-2}(n) = O(\log^{(i)} n) \end{aligned}$$

補題 2 を用いると, $2 \leq i \leq \log^* n$ について, $O(\frac{n}{\log^{(i)} n})$ プロセッサを用いて $O(\log^{(i)} n)$ 時間で計算可能なアルゴリズムが得られる. この結果も同様に, $i = \log^* n$ のとき $O(\frac{n}{\log^* n})$ プロセッサでを用いて $O(\log^* n)$ 時間で計算可能なアルゴリズムを得る. これにより, プロセッサ数のオーダを変えずに計算時間を削減した.

3.2 考察

$2 \leq i \leq \log^* n$ について, $O(\frac{n}{\log^{(i)} n})$ プロセッサを用いて $O(\log^{(i)} n + i)$ 時間で計算可能なアルゴリズムが, 従来のコスト最適なアルゴリズムの計算時間である $O(\log^* n)$ より高速になり得るかを考える.

計算時間には, i の増加に伴い減少する項 $\log^{(i)} n$ と i の増加に伴い増加する項 i が存在する. よって, この方法で作成することができるアルゴリズムの中で最も高速な計算時間となると, $\log^{(i)} n = i$ となる. ここで, $\log^{(i)} n \leq \log^* n$ となるような i の範囲を考える. $\log^* n$ の値が増加する境界の値 $1, 2, 4, 16, 65536, \dots$ のみについて $n = 2 \uparrow \log^* n$ と表すことができる. ここで, \uparrow はクヌースの矢印

表記 [5] であり, $a \uparrow b = a^{a^{\cdot^{\cdot^{\cdot}}}}$ (a の数が b 個) となる. $n = 2 \uparrow \log^* n$ であることから $\log^{(i)} n = 2 \uparrow (\log^* n - i)$ となる. $\log^{(i)} n \leq \log^* n$ となると, 同様に $\log^* \log^* n$ の値が増加する境界の値のみについて $\log^* n = 2 \uparrow \log^* \log^* n$ であることを考えると, $\log^* n - i \leq \log^* \log^* n$ となる. このことから得られる $\log^* n - \log^* \log^* n \leq i \leq \log^* n$ の範囲で計算時間を評価すると, i の項が影響して $O(\log^* n)$ となるため, このアルゴリズムは従来の計算時間である $O(\log^* n)$ より高速にならない.

4 まとめ

本研究では, $2 \leq i \leq \log^* n$ を満たす整数 i について, $O(\frac{n \log^{(i)} n}{i})$ プロセッサを用いて $O(i)$ 時間で計算可能なアルゴリズムと, $O(\frac{n}{\log^{(i)} n + i})$ プロセッサを用いて $O(\log^{(i)} n + i)$ 時間で計算可能なアルゴリズムを示した. これにより, 定数時間でソーティング問題を解くアルゴリズムに必要なプロセッサ数を削減したが, 一方でコスト最適なアルゴリズムは従来の $O(\log^* n)$ 時間よりも高速にはできなかった. 特にプロセッサ数が $O(n)$ で定数時間でソーティングが可能であるかどうかを示すことが今後の課題である.

参考文献

- [1] 浅野他共訳, T. コルメン他共著. アルゴリズムイントロダクション [第3巻] 精選トピックス. 近代科学社, (1995).
- [2] S.C.Eisenstat. $O(\log^* n)$ algorithms on a Sum-CRCW PRAM. *Computing*.(2007), Vol.79, p.93-97.
- [3] Fabian Frei, Koichi Wada. Efficient deterministic MapReduce algorithms for parallelizable problems. *Journal of Parallel and Distributed Computing*. (2023), Vol.177, p.28-38.
- [4] J.Jaja. An Introduction to Parallel Algorithms. ADDISON-WESLEY, (1992), p.26.
- [5] フィッシュ. 巨大数論 第2版. インプレス R&D, (2017).