

多次元プロセッサ配列上の 並列ソートと時間計算量

嵯峨幸治
Koji SAGA

佐渡一広
Kazuhiro SADO

五十嵐善英
Yoshihide IGARASHI

群馬大学工学部情報工学科

1. はじめに

近年集積回路技術の進歩により、多重プロセッサ上で効率良く実行できる並列アルゴリズムの研究が盛んである。特に並列ソーティングアルゴリズムは、実用的に重要であるばかりでなく理論的にも大変興味ある問題であるため、様々な研究が行われている。Ajtai^[1]は、 n 個の要素を $O(\log n)$ 時間でソートする並列アルゴリズムを提案しているが、構造が複雑で実現は困難である。Thompson と Kung^[14]が提案した網状結合プロセッサ配列 (mesh-connected processor array) は、VLSI 向きモデルとして注目されている。このモデル上での効率の良い並列ソーティングアルゴリズムと計算時間に関する研究が、文献 [4~12] に発表されている。本論文では、3次元配列モデルで実現する並列ソーティングアルゴリズムの設計と効率評価を行う。本アルゴリズムは、要素が完全にソートされた8つの配列のマージを再帰的に行う。 n^3 個の要素を $15n + \log_2 n - 25$ 単位時間でソートでき、同じモデル上で $19n - 19$ 単位時間で実現する Schimmler のアルゴリズム^[11]より効率が良い。また、Kunde^[4]は漸近的に $10.5n + O(n^{2/3} \log_2 n)$ 単位時間でソートできるアルゴリズムを提案しているが、実用的な大きさの n に対しては、Schimmler のアルゴリズムと同程度の速さである。更に Kunde は、3次元配列モデルで n^3 個の要素をソートするいかなる並列ソーティングアルゴリズムも、少なくとも $5n$ ステップかかることを導いているが、我々は並列マージソートおよび並列擬似マージソートを用いたクラスの計算時間の下限値を導く。これらは $7.25n - 4\log_2 n - 8$ 単位時間、 $5.25n - \log_2 n - 6$ 単位時間である。

2. 3次元配列上のソーティングアルゴリズム

プロセッサ配列 $A[1..N]$ を考える。各プロセッサ $A[i]$ ($1 \leq i \leq N$) には、あらかじめデータが入っているとする。手続き $BUBBLE(A[i..j], k)$ は、配列 $A[i..j]$ に対し、要素を k 回非減少順の並列バブルを行う。手続き $BUBBLE$ の2~8行目の計算時間を単位比較置換時間 t_c とする。よって、手続き $BUBBLE(A[i..j], k)$ の計算時間は、 kt_c である。また、配列 $A[i..j]$ の要素を k 回非増加順の並列バブルを行う手続きを $\overline{BUBBLE}(A[i..j], k)$ とする。これは、手続き $BUBBLE$ の4行目と7行目の不等号の向きを逆にすればよい。

シャフルとは、数列 $a_1, a_2, \dots, a_{2n}$ を数列 $a_1, a_{n+1}, a_2, a_{n+2}, \dots, a_n, a_{2n}$ に置換する操作であり、手続き $SHUFFLE$ で示す。手続き $SHUFFLE$ の2~6行目の計算時間を単位単純置換時間 t_e とする。よって、 $(j-i+1)$ 個の要素を手続き $SHUFFLE(A[i..j])$ によりシャフルする計算時間は、 $\lfloor (j-i)/2 \rfloor t_e$ である。

```

procedure BUBBLE(A[i..j], k);
begin
1.   for s:=1 to k do
2.     if odd(s) then
3.       for all t:= $\lceil (i+1)/2 \rceil$ .. $\lfloor j/2 \rfloor$  do in parallel
4.         if A[2t-1] > A[2t] then
5.           exchange A[2t-1], A[2t]
6.       else
7.         for all t:= $\lceil i/2 \rceil$ .. $\lfloor (j-1)/2 \rfloor$  do in parallel
8.           if A[2t] > A[2t+1] then
              exchange A[2t], A[2t+1]
end

procedure SHUFFLE(A[i..j]);
begin
1.   for s:=1 to  $\lfloor (j-i)/2 \rfloor$  do
2.     if odd(s) then
3.       for all t:= $\lfloor s/2 \rfloor$ .. $\lfloor s/2 \rfloor$  do in parallel
4.         exchange A[i+ $\lfloor (j-i)/2 \rfloor$ +2t], A[i+ $\lfloor (j-i)/2 \rfloor$ +2t+1]
5.     else
6.       for all t:=-(s/2)+1.. $\lfloor s/2 \rfloor$  do in parallel
              exchange A[i+ $\lfloor (j-i)/2 \rfloor$ +2t-1], A[i+ $\lfloor (j-i)/2 \rfloor$ +2t]
end

```

$n \times n \times n$ の網状結合プロセッサ配列 $A[1..n, 1..n, 1..n]$ は、各プロセッサが隣接するプロセッサのみに接続し、プロセッサは各々を区別するために一定の番号が付けられている。 n^3 個の非減少順にソートされた系列を $A[1,1,1]$, $A[1,1,2]$, ..., $A[n,n,n]$ の順に並べる面優先-行優先順 (plane-major row-major order) や $A[1,n,n]$, $A[1,n,n-1]$, ..., $A[n,n,n]$ の順に並べる面優先-逆行優先順 (plane-major reverse row-major order) がその代表である (図1参照)。本論文では、 $A[1..n, 1..n, 1..n]$ の第 j 行第 k 列の系列を $A[1..n, j, k]$ と書き、この系列を x 方向の系列と呼ぶ。同様に、第 i 平面第 j 行の系列や第 i 平面第 k 列の系列を各々 $A[i, j, 1..n]$, $A[i, 1..n, k]$ と書き、各々 y 方向の系列, z 方向の系列と

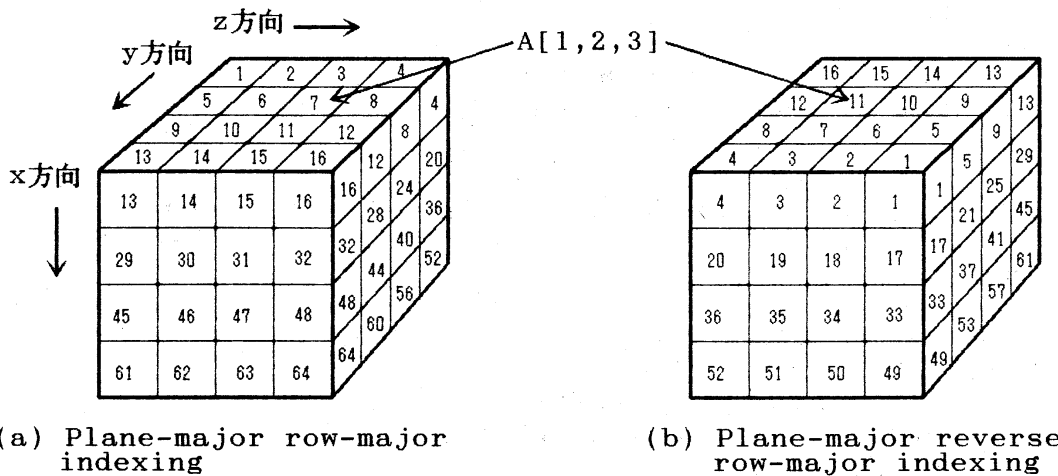


Fig. 1 Processor indexing schemes.

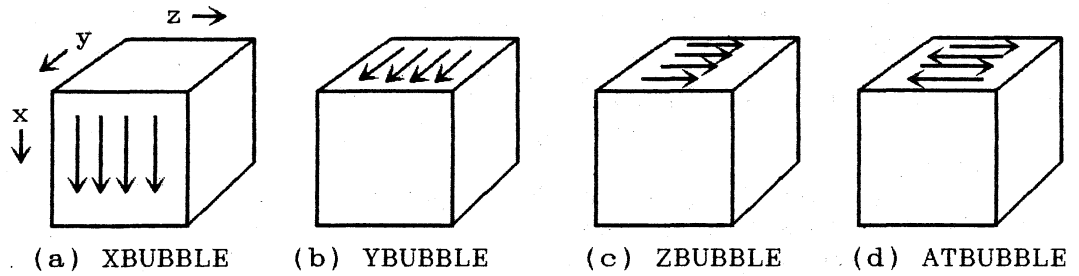


Fig.2 BUBBLE operations

呼ぶ。また、プロセッサ $A[i, j, k]$ ($p_1 \leq i \leq p_2$, $q_1 \leq j \leq q_2$, $r_1 \leq k \leq r_2$) から構成される配列を $A[p_1 \dots p_2, q_1 \dots q_2, r_1 \dots r_2]$ と書く。

手続き ZSHUFFLE($A[p_1 \dots p_2, q_1 \dots q_2, r_1 \dots r_2]$) の 1~3 行目は、系列 $A[i, j, r_1 \dots r_2]$ での並列シャフルを行っている。4~6 行目は、 $A[p_1+2i-1, j, r_1+2k-2]$ と $A[p_1+2i-1, j, r_1+2k-1]$ ($1 \leq i \leq (p_2-p_1+1)/2$, $q_1 \leq j \leq q_2$, $1 \leq k \leq (r_2-r_1+1)/2$) の要素を単純置換する。手続き YSHUFFLE($A[p_1 \dots p_2, q_1 \dots q_2, r_1 \dots r_2]$) は、系列 $A[i, q_1 \dots q_2, k]$ について同様の操作を行う。

手続き XBUBBLE は、系列 $A[p_1 \dots p_2, j, k]$ を x 方向に s 回非減少順の並列バブルを行う (図 2(a) 参照)。同様に、系列 $A[i, q_1 \dots q_2, k]$ を y 方向に非減少順に並列バブルを行う手続きを YBUBBLE (図 2(b) 参照)、非増加順に並列バブルを行う手続きを $\overline{\text{YBUBBLE}}$ とする。また、系列 $A[i, j, r_1 \dots r_2]$ を z 方向に非減少順に並列バブルを行う手続きを ZBUBBLE (図 2(c) 参照)、非増加順に並列バブルを行う手続きを $\overline{\text{ZBUBBLE}}$ とする。手続き ATBUBBLE は、系列 $A[i, j, r_1 \dots r_2]$ を z 方向に、 j が奇数のときは非減少順に、偶数のときは非増加順に s 回の並列バブルを行う (図 2(d) 参照)。手続き PLANEUBUBBLE は、 $\text{sign} = 0$ のとき、系列 $A[i, q_1 \dots q_2, r_1 \dots r_2]$ を y 方向に非減少順に s_j 回、さらに z 方向に非減少順に s_k 回の並列バブルを行う。 $\text{sign} = 1$ のときは、各々の方向で非増加順に並列バブルを行う。

```

procedure ZSHUFFLE( $A[p_1 \dots p_2, q_1 \dots q_2, r_1 \dots r_2]$ );
begin
1.   for all  $i := p_1 \dots p_2$  do in parallel
2.     for all  $j := q_1 \dots q_2$  do in parallel
3.       SHUFFLE( $A[i, j, r_1 \dots r_2]$ )
4.     for all  $j := q_1 \dots q_2$  do in parallel
5.       for all  $i := 1 \dots (p_2 - p_1 + 1)/2$  and  $k := 1 \dots (r_2 - r_1 + 1)/2$  do in parallel
6.         exchange  $A[p_1 + 2i - 1, j, r_1 + 2k - 2], A[p_1 + 2i - 1, j, r_1 + 2k - 1];$ 
end

procedure XBUBBLE( $A[p_1 \dots p_2, q_1 \dots q_2, r_1 \dots r_2], s$ );
begin
1.   for all  $j := q_1 \dots q_2$  do in parallel
2.     for all  $k := r_1 \dots r_2$  do in parallel
3.       BUBBLE( $A[p_1 \dots p_2, j, k], s$ )
end

```

```
procedure ATBUBBLE(A[p1..p2, q1..q2, r1..r2], s);
```

```
begin
```

1. for all i := p₁..p₂ do in parallel
2. for all j := q₁..q₂ do in parallel
3. if odd(j) then
4. BUBBLE(A[i, j, r₁..r₂], s)
5. else
6. BUBBLE(A[i, j, r₁..r₂], s)

```
end
```

```
procedure PLANEUBUBBLE(A[p1..p2, q1..q2, r1..r2], sj, sk, sign);
```

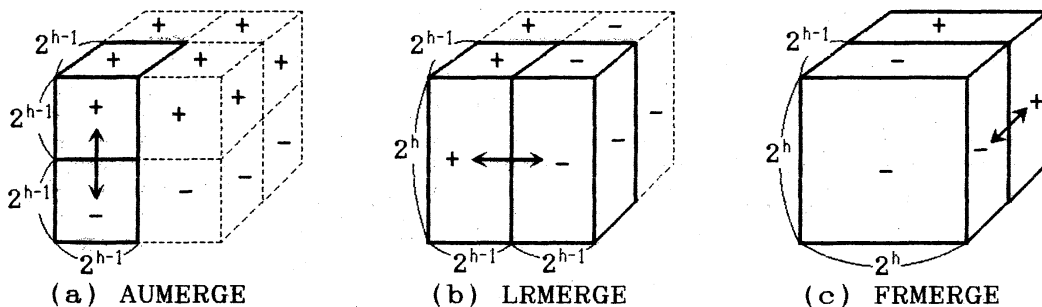
```
begin
```

1. if sign = 0 then begin
2. YBUBBLE(A[p₁..p₂, q₁..q₂, r₁..r₂], s_j);
3. ZBUBBLE(A[p₁..p₂, q₁..q₂, r₁..r₂], s_k)
4. end
5. else begin
6. YBUBBLE(A[p₁..p₂, q₁..q₂, r₁..r₂], s_j);
7. ZBUBBLE(A[p₁..p₂, q₁..q₂, r₁..r₂], s_k)

```
end
```

```
end
```

2x2x2 プロセッサ配列上で8個の要素をソートする手続き EIGHTSORT は, $\text{sign} = 0$ のとき要素を面優先-行優先順に, $\text{sign} = 1$ のとき面優先-逆行優先順にソートする. 手続き AUMERGE は, 面優先-行優先順と面優先-逆行優先順にソートされている大きさの等しい上下に向かい合う2つの配列をマージし, すべての要素を面優先-行優先順 (面優先-逆行優先順) にソートする. 同様に手続き LRMERGE は, 左右に向かい合う2つの配列を, 手続き FRMERGE は前後に向かい合う2つの配列をマージする. 手続き CMERGE は, 上記の3つの手続きを組み合わせる面優先-行優先順と面優先-逆行優先順にソートされている各々4つの配列をマージし, すべての要素を面優先-行優先順 (面優先-逆行優先順) にソートする. 手続き CMERGE の1, 2行目で隣接する上下の配列のマージを (図3(a) 参照), 3, 4行目で隣接する左右の配列のマージを (図3(b) 参照), 5行目で隣接する前後の配列のマージを行う (図3(c) 参照). 以上の手続きを組合わせて配列 $A[1..2^h, 1..2^h, 1..2^h]$ の要素をソートする手続き CUBESORT ができる.



+ : sorted plane-major row-major order
 - : sorted plane-major reverse row-major order
 Fig.3 A merging process by CMERGE

```

procedure EIGHTSORT(A[p..p+1, q..q+1, r..r+1], sign);
begin /* p, q and r are odd */
1.  ATBUBBLE(A[p..p+1, q..q+1, r..r+1], 1);
2.  for all i := 0 .. 1 do in parallel
      PLANEUBUBBLE(A[p+i, q..q+1, r..r+1], 1, 1, i)
3.  XBUBBLE(A[p..p+1, q..q+1, r..r+1], 1)
4.  PLANEUBUBBLE(A[p..p+1, q..q+1, r..r+1], 1, 1, sign)
end

procedure AUMERGE(A[p..p+2t-1, q..q+t-1, r..r+t-1], sign);
begin
1.  XBUBBLE(A[p..p+2t-1, q..q+t-1, r..r+t-1], 2t);
2.  PLANEUBUBBLE(A[p..p+2t-1, q..q+t-1, r..r+t-1], t, t, sign)
end

procedure LRMERGE(A[p..p+2t-1, q..q+t-1, r..r+2t-1], sign);
begin
1.  ZSHUFFLE(A[p..p+2t-1, q..q+t-1, r..r+2t-1]);
2.  XBUBBLE(A[p..p+2t-1, q..q+t-1, r..r+2t-1], t);
3.  ATBUBBLE(A[p..p+2t-1, q..q+t-1, r..r+2t-1], 1);
4.  PLANEUBUBBLE(A[p..p+2t-1, q..q+t-1, r..r+2t-1], t, 2t, sign)
end

procedure FRMERGE(A[p..p+2t-1, q..q+2t-1, r..r+2t-1], sign);
begin
1.  YSHUFFLE(A[p..p+2t-1, q..q+2t-1, r..r+2t-1]);
2.  XBUBBLE(A[p..p+2t-1, q..q+2t-1, r..r+2t-1], t);
3.  PLANEUBUBBLE(A[p..p+2t-1, q..q+2t-1, r..r+2t-1], 2t, 2t, sign)
end

procedure CMERGE(A[p..p+2h-1, q..q+2h-1, r..r+2h-1], sign)
begin
1.  for all j := 0..1 and k := 0..1 do in parallel
2.    AUMERGE(A[p..p+2h-1, q+j·2h-1..q+(j+1)·2h-1-1,
              r+k·2h-1..r+(k+1)·2h-1-1], k);
3.  for all j := 0..1 do in parallel
4.    LRMERGE(A[p..p+2h-1, q+j·2h-1..q+(j+1)·2h-1-1, r..r+2h-1], j);
5.  FRMERGE(A[p..p+2h-1, q..q+2h-1, r..r+2h-1], sign)
end

procedure CUBESORT(A[1..2h, 1..2h, 1..2h], 0);
begin
1.  if h = 0 then (* skip *)
2.  else if h = 1 then EIGHTSORT(A[p..p+1, q..q+1, r..r+1], sign);
      else begin
3.    for all i, j, k = 0..1 do in parallel
4.      CUBESORT(A[p+i·2h-1..p+(i+1)·2h-1-1, q+j·2h-1..q+(j+1)·2h-1-1,
                  r+k·2h-1..r+(k+1)·2h-1-1], i);
5.      CMERGE(A[p..p+2h-1, q..q+2h-1, r..r+2h-1], sign);
      end;
end

```

3. アルゴリズムの正当性と計算量

正当性の証明は、0-1 原理 (zero-one principle)^[3] を用いて行い、要素は 0 と 1 のみから構成されているとする。

定理 3.1 すでにソートされている 2 つの系列 $a_1, a_3, \dots, a_{n-1}$ と $a_2, a_4, \dots, a_n$ に対し、並列バブルを用いて系列 $a_1, a_2, a_3, a_4, \dots, a_{n-1}, a_n$ をソートするのに必要な比較交換時間は、高々 $n/2$ である^[2]。

定理 3.2 2^d 個の要素の $2 \times 2 \times \dots \times 2$ 網状結合プロセッサ配列上でのソートは、並列双単調ソート法 (bitonic sort) により、 $d(d+1)/2$ 回の比較置換時間でできる^[13]。

補題 3.1 要素が面優先-行優先順と面優先-逆行優先順にソートされている大きさの等しい上下に向かい合う 2 つの配列 $A[p..p+k-1, q..q+k-1, r..r+k-1]$ と $A[p+k..p+2k-1, q..q+k-1, r..r+k-1]$ の要素は、手続き AUMERGE により面優先-行優先順 (面優先-逆行優先順) にソートできる ($k \geq 2$ の偶数)。

証明 上部および下部配列は、各々第 a 面および第 b 面の要素のみ 0 と 1 が混在し、それより上の面の要素はすべて 0 であり、下の面の要素は 1 である。1 行目の手続き XBUBBLE を実行すると、配列 A の第 $a+b$ 面の要素のみ 0 と 1 が混在するので、以下この面についてソートすることを考える (図 4(a) 参照)。2 行目の PLANEUBBLE により各列を非減少順にソートした後 (図 4(b) 参照)、各行を非減少順にソートすると (図 4(c) 参照)、第 $a+b$ 面の要素は非減少順にソートされる。よって、配列 A のすべての要素は面優先-行優先順にソートされる。また、配列 A の要素を面優先-逆行優先順にソートするには、2 行目の文で各列を非増加順にソートし、各行を非増加順にソートすればよい。 $\square$

補題 3.2 面優先-行優先順と面優先-逆行優先順にソートされている大きさの等しい左右に向かい合う 2 つの配列 $A[p..p+2k-1, q..q+k-1, r..r+k-1]$ と $A[p..p+2k-1, q..q+k-1, r+k..r+2k-1]$ の要素は、手続き LRMERGE により、面優先-行優先順 (面優先-逆行優先順) にソートできる ($k \geq 2$ の偶数)。

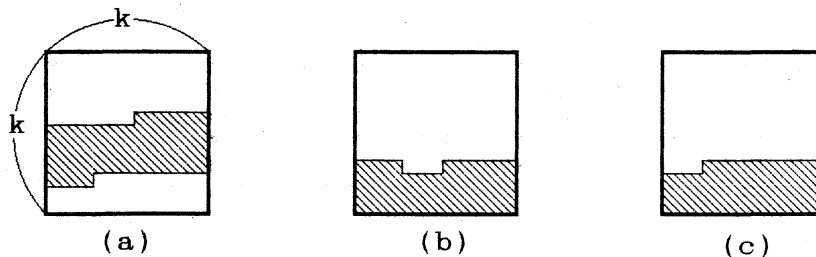


Fig.4 A process by AUMERGE for merging $2k \times k \times k$ subarrays (from above)

証明 左部および右部配列は、各々第 a 面および第 b 面の要素のみ 0 と 1 が混在している。このとき、配列 A を上から見た所を図 5(a) に示す。1 行目の ZSHUFFLE 実行後、x 方向の系列を非減少順にソートするには、定理 3.1 より k 回の並列バブルを行えばよい (図 5(b) 参照)。よって、XBUBBLE 実行後、補題 3.1 の証明と同様に第 a+b 面について考える。3 行目の ATBUBBLE 実行後、y 方向の任意の 2 列の 0 の数の差は高々 1 となる (図 5(c) 参照)。以後補題 3.1 と同様に証明できる。□

補題 3.3 面優先-行優先順と面優先-逆行優先順にソートされている大きさの等しい前後に向かい合う 2 つの配列 $A[p..p+2k-1, q..q+k-1, r..r+2k-1]$ と $A[p..p+2k-1, q+k..q+2k-1, r..r+2k-1]$ の要素は、手続き FRMERGE により、面優先-行優先順 (面優先-逆行優先順) にソートできる ($k \geq 2$ の偶数)。

証明 補題 3.2 と同様に証明できる。□

補題 3.1, 補題 3.2, 補題 3.3 より次の定理が導ける。

定理 3.3 手続き CUBESORT($A[1..n, 1..n, 1..n], 0$) は、 n^3 個の要素を面優先-行優先順にソートする (n は 2 の巾乗)。

定理 3.4 手続き CUBESORT($A[1..n, 1..n, 1..n], 0$) の計算時間は、 $(2n-4)t_e + (13n + \log_2 n - 21)t_c$ である。

証明 $T(2k)$ を、配列の要素が面優先-行優先順と面優先-逆行優先順にソートされた大きさ $k \times k \times k$ の各々 4 つの配列をマージし、大きさ $2k \times 2k \times 2k$ の要素がソートされた配列を得るための計算時間とする。定理 3.2 より $T(2) = 6$ である。 $k > 2$ のときは、CMERGE の計算時間から、 $T(2k) = 2kt_e + (13k+1)t_c$ となる。従って、 $n \geq 2$ に対する CUBESORT の計算時間は、

$$\begin{aligned} T(n) &= \sum_{i=2}^{\log_2 n} (2 \cdot 2^{i-1} t_e + (13 \cdot 2^{i-1} + 1) t_c) + T(2) \\ &= (2n-4)t_e + (13n + \log_2 n - 21)t_c \end{aligned}$$

□

上記の定理よりただちに次の系が導ける。

系 3.1 $t_e \approx t_c$ としてこれらの単位計算時間をステップとすると、手続き CUBESORT($A[1..n, 1..n, 1..n]$) の計算時間は、 $15n + \log_2 n - 25$ ステップである。

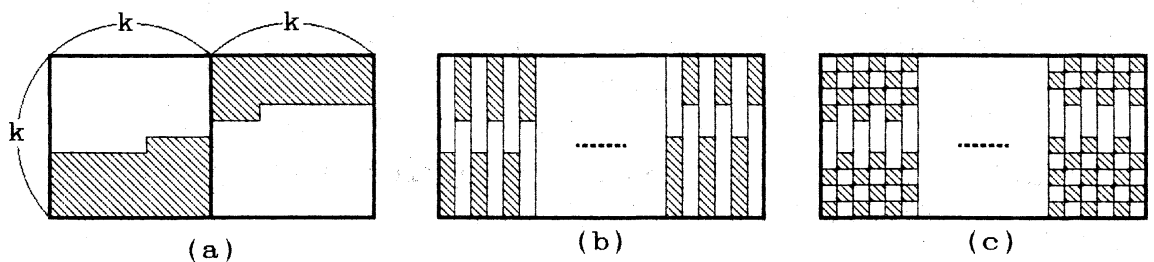


Fig.5 A process by LRMERGE for $2k \times k \times 2k$ (from above)

4. 3次元以上の網状配列における時間計算量の下限

d 次元網状結合プロセッサ配列モデルにおいて、並列マージ法および並列擬似マージ法を用いて n^d 個の要素をソートする計算時間の下限値を導く。面優先-行優先順序を d 次元に拡張した順序を辞書式順という。以下、この順序にソートすることを単にソートするという。

定義 4.1 すべての $p \leq i_1 < i_2 \leq q$, $r \leq j_1, j_2 \leq s$, $t \leq k_1, k_2 \leq u$ に対し, $A[i_1, j_1, k_1] \leq A[i_2, j_2, k_2]$ であるとき, 配列 $A[p..q, r..s, t..u]$ は荒くソートされているという。

繰返しマージソートと繰返し擬似マージソートの概要は (1) と (2) である。

(1) **for** $i := 1 \dots r$ **do**

ソートした 8 つの $2^{i-1} \times 2^{i-1} \times 2^{i-1}$ 配列をマージしソートした $2^i \times 2^i \times 2^i$ 配列を作る。

(2) **for** $i := 1 \dots r$ **do**

荒くソートした 8 つの配列をマージして荒くソートした配列を作る;

for $i := 1 \dots 2^r$ **do in parallel**

第 i 平面にある要素を行優先にソートする。

(2) のアルゴリズムの最初の **for** 文の部分を繰返し擬似マージと呼ぶ。本アルゴリズムや Schimmler のアルゴリズムは、繰返しマージソートに属する。

次の補題は容易に導ける。

補題 4.1 $2 \times 2 \times 2$ 網状プロセッサ配列において、単純置換と比較置換のみを基本命令として 8 個の要素をソートするには、少なくとも 4 ステップ必要である。

補題 4.2 $4 \times 4 \times 4$ 網状プロセッサ配列において、繰返しマージソートによりソートするには、少なくとも 13 ステップかかる場合がある。

証明 以下に示す配列 $A[1..4, 1..4, 1..4]$ の初期状態を考える:

配列 $A[3..4, 3..4, 3..4]$ の p 個の要素は、他のどの 7 つの $2 \times 2 \times 2$ 配列の要素よりも小さく、配列 $A[3..4, 3..4, 3..4]$ の $8-p$ 個の要素は、他のどの 7 つの $2 \times 2 \times 2$ 配列よりも大きい ($0 \leq p \leq 8$)。

補題 4.1 から、配列 $2 \times 2 \times 2$ の要素をソートするには 4 ステップ以上かかる。8 つの $2 \times 2 \times 2$ 配列をマージする過程で、初めに配列 $A[3..4, 3..4, 3..4]$ にあったどの要素も、5 ステップ以下では $A[1,1,1]$ に到達できない。また、5 ステップ後に $A[1,1,1]$ にある要素の最終位置は、 p の値により第 1 面 1 行目以外の最右端となる場合がある。従って、このマージ過程で少なくとも 9 ステップかかる。よって、少なくとも 13 ステップ必要である。 □

定理 4.1 $n \times n \times n$ 網状プロセッサ配列において、いかなる繰返し並列マージ法でも、 $n^3 (n \geq 4)$ 個の要素をソートするには、少なくとも $7.25n - 4\log_2 n - 8$ ステップ必要である。

証明 $n=4$ のとき、補題 4.1 より明らか。 $T(m)$ を大きさ $m \times m \times m$ 網状プロセッサ配列において、いかなる繰返しマージソートでも m^3 個の要素をソートするのに必要な計算時間とする。大きさ $2k \times 2k \times 2k$ 配列 $A[1..2k, 1..2k, 1..2k]$ ($k \geq 4$) を考える。すべての配列 $k \times k \times k$ のすべての要素は、すでに独立に繰返しマージソートによりソートされているとする。ここで、この 8 つの配列 $k \times k \times k$ を配列 $2k \times 2k \times 2k$ にマージする過程を考える。各々の配列は独立にソートされているから、次のような場合を考える：

配列 $A[k+1..2k, k+1..2k, k+1..2k]$ の p ($0 \leq p < 4k^2$) 個の要素は、他の 7 つの配列

のどの要素よりも大きく、それ以外の要素は他の配列の要素よりも小さい。

このマージ過程で、初めに配列 $A[k+1..2k, k+1..2k, k+1..2k]$ にあったどの要素も、 $3k-1$ ステップ以下では $A[1,1,1]$ に到達できない。また、 $3k-1$ ステップ後に $A[1,1,1]$ にある要素は、 p の値により $A[q,2k,2k]$ ($k/4 \leq q \leq 2k$) に移動する場合がある。よって、その要素は、さらに少なくとも $4.25k-3$ ステップ移動しなければならない。したがって、次の再帰式が導ける。

$$T(2k) \geq T(k) + 7.25k - 4 \quad (k \geq 4)$$

$$T(4) \geq 13 \quad (\text{補題 4.1 より})$$

これを解くと、

$$T(n) \geq 7.25n - 4\log_2 n - 8.$$

□

次の 2 つの補題 4.1 と補題 4.2 は同様の方法で導ける。

補題 4.3 $2 \times 2 \times 2$ 網状プロセッサ配列において、単純置換と比較置換のみを基本命令として 8 個の要素を荒くソートするには、少なくとも 3 ステップ必要である。

補題 4.4 $4 \times 4 \times 4$ 網状プロセッサ配列において、いかなる繰返し擬似マージでも要素を荒くソートするには、少なくとも 8 ステップかかる場合がある。

よって、次の 2 つの定理が導ける。

定理 4.2 $4 \times 4 \times 4$ 網状プロセッサ配列において、いかなる擬似マージソートでも要素をソートするには、少なくとも 12 ステップ必要である。

定理 4.3 $n \times n \times n$ 網状プロセッサ配列 ($n \geq 8$) において、いかなる繰返し並列擬似マージ法でも n^3 個の要素をソートするには、少なくとも $5.25n - \log_2 n - 6$ ステップ必要である。

定理 4.1 と定理 4.3 を多次元配列に拡張すると、次の 2 つの定理が得られる。

定理 4.4 d 次元網状プロセッサ配列で、 n^d 個 ($n \geq 2^d$) の要素をソートするいかなる繰返し並列マージソートでも、少なくとも $(3d+2^{-d+1}-2)n - (d+1)\log_2 n - 2^{d-1}(3d-2) + d^2 - 2 + T(2^{d-1})$ ステップ必要である。ここで $T(2^{d-1})$ は、 $(2^{d-1})^d$ 個の要素を繰返しマージによりソートするのに必要な計算時間である。

定理 4.5 d 次元網状プロセッサ配列で, n^d 個 ($n \geq 2^d$) の要素をソートするいかなる繰返し擬似マージソートでも, 少なくとも $(2d+2^{-d+1})n - \log_2 n - 2^{d-1}d - 2 + T(2^{d-1})$ ステップ必要である. ここで $T(2^{d-1})$ は, $(2^{d-1})^d$ 個の要素を繰返し擬似マージにより荒くソートするのに必要な計算時間である.

5. むすび

3次元網状結合プロセッサ配列モデルで実現する並列ソーティングアルゴリズムの設計を行った. 本アルゴリズムで用いる命令は, 隣合う要素の単純置換と比較置換のみであるため大変局所的で, 命令系列が規則的であるため VLSI 向きのアルゴリズムであると考えられる. また, 3次元以上の網状結合プロセッサ配列モデル上で, 並列マージソート法や並列擬似マージソート法の計算時間の下限値を導いた. これは, 将来, 多次元に接続されたプロセッサモデルの実現性を考える上で大変興味深い問題であるが, 残念ながら我々が導いた下限値と設計したアルゴリズムの計算量との間には差がある.

References

1. Ajtai, M., et. al.: "An $O(N \log N)$ sorting network", Proc. 15 ACM Symp. on Theory of Computing, pp. 1-9 (1983).
2. Batcher, K.E.: "Sorting networks and their applications", Proceedings of AFIPS Spring Joint Computer Conf., 32, pp. 307-314 (1968).
3. Knuth, D.E.: "The art of computer programming: Vol. 3 Sorting and searching", Addison-Wesley, Reading, Mass. (1973).
4. Kunde, M.: "Lower bounds for sorting on mesh-connected architectures", VLSI Algorithms and Architectures, Lect. Notes in Comp. Sci., 227, pp. 84-95 (1986).
5. Lang, H.-W., et. al.: "Systolic sorting on a mesh-connected network", IEEE Trans. Comput., C-34, pp. 652-658 (1985).
6. Sado, K. and Igarashi, Y.: "A divide-and-conquer method of the parallel sort", IECEJ Technical Report, AL84-68 (Feb. 1985).
7. Sado, K. and Igarashi, Y.: "A fast parallel pseudo-merge sort algorithm", IECEJ Technical Report, AL85-16 (June 1985).
8. Sado, K. and Igarashi, Y.: "Some parallel sorts on a mesh-connected processor array", J. Parallel and Distributed Computing, Vol. 3, No. 3, pp. 398-410 (1986).
9. Saga, K., et. al.: "An Iterative parallel pseudo-merge sort and its computing time", IECEJ Technical Report COMP86-26 (July 1986).
10. Saga, K., et. al.: "Parallel Pseudo-merge sorting on a mesh-connected processor array", Trans. of IECEJ, E69, pp. 1104-1113 (1986).
11. Schimmler, M.: "Sorting on a three dimensional cube grid", Technical Report 8604, Christian Albrechts Universität, Kiel (1986).
12. Schnorr, C.P. and Shamir, A.: "An optimal sorting algorithm for mesh-connected computers", Proc. 18 ACM Symp. on Theory of Computing, pp. 255-263 (1985).
13. Siegel, H.J.: "The universality of various types of SIMD machine interconnection networks", Proc. 4 ACM Symp. on Computer Architecture, pp. 23-25 (1977).
14. Thompson, C.D. and Kung, H.T.: "Sorting on a mesh-connected parallel computer", Commun., ACM, 20, pp. 263-271 (1977).