数理解析研究所講究録
第556巻 1985年 37-48

37

EFFICIENT IMPLEMENTATIONS OF PARALLEL SORT ALGORITHMS ON A MESH-CONNECTED PROCESSOR ARRAY

Yoshihide Igarashi, Kazuhiro Sado and Noriaki Adachi

五十嵐 善英　　佐渡 一広　　安達 範明

Department of Computer Science,

Gunma University, Kiryu 376, Japan

1. Introduction

The parallel bubble sort is also called the odd-even transposition sort and rather slow compared with other sophisticated parallel sort algorithms. If we assume that the computing time for routing values, comparing two values and exchanging the two values if necessary is one time unit, then the parallel bubble sort takes N time to sort N values whereas both Batcher's bitonic sort and odd-even merge sort take $O(\log^2 N)$ time [2]. The time complexity of the fastest parallel sort known is $O(\log N)$ [1, 2, 7]. Thompson and Kung [10], Nassimi and Sahni [8], and Kumar and Hirschberg [6] have shown efficient implementations of the Batcher's algorithms to sort N values on an $n \times n$ mesh-connected processor array, where $N = n^2$. These implementations take $O(N^{1/2})$ time.

Although the time complexity of the parallel bubble sort is poor, it has a couple of significant advantages. The construction of the parallel bubble sort is simple, and it needs very little in the way of control hardware. The parallel bubble sort network occupies less area than any of other parallel sort networks. The parallel bubble sort, therefore, is considered to be the most realistic sort algorithm to be implemented on a VLSI chip. The time unit used in this paper is called a stage.

In this paper we investigate how we can improve the time efficiency of the parallel bubble sort without increasing much its contol hardware. We introduce a function defined on the set of pairs of values and stages of the parallel computation. It is denoted by COUNT(v, s) and proved that its function value is the exact number of stages necessary to route value v at stage s to its final position. This function is a powerful tool to design efficient parallel sort algorithms and to analyze their

time efficiency. We design some parallel sort algorithms that can be implemented on a mesh-connected parallel processor array. Our algorithms are some combinations of the parallel bubble sorts in horizontal and vertical directions, and their structure and control hardware are simple. Although the time complexities of our algorithms are $O(N^{1/2} \log N)$, even in the worst case they are as fast as the implementations of Batcher's algorithms for practical values of N, $1 \leq N \leq 128^2$. In the average case some of our algorithms are even faster than the implementations of Bathcer's algorithms for the same range of practical values of N.

2. The parallel bubble sort

Although we study sorting networks realized on VLSI chips, for clarity we describe our algorithms by sequences of PASCAL like statements. We suppose that N values are initially stored in an array A[1 . . N]. Without loss of generality we assume that N values in A[1 . . N] are a permutation of (1, 2, . . . ,N) unless we specify them. The computation of the parallel bubble sort starts at the first stage and its s-th stage consists of the following operations (s = 1, 2, . . .):

Case 1. s is an odd number: The following $\lfloor N/2 \rfloor$ operations are executed in parallel: A[2t-1] and A[2t] are compared, and if the order is not correct then the contents of A[2t-1] and A[2t] are exchanged ($t = 1, \ldots, \lfloor N/2 \rfloor$).

Case 2. s is an even number: The following $\lfloor (N-1)/2 \rfloor$ operations are executed in parallel: A[2t] and A[2t+1] are compared, and if the order is not correct then the contents of A[2t] and A[2t+1] are exchanged ($t = 1, \ldots, \lfloor (N-1)/2 \rfloor$).

The parallel bubble sort is defined as BUBBLE(A[1 . . N], N), where procedure BUBBLE is defined as follows:

```
     procedure BUBBLE(A[1 . . N], k);
     begin
1.     for s:= 1 step 1 until k do
2.       if s is odd then
3.         for all t (t = 1,. . .,⌊N/2⌋) do in parallel
4.           if A[2t-1] > A[2t] then exchange A[2t-1] and A[2t]
5.       else   {in the case where s is even}
```

6. for all t (t = 1,. . .,⌊(N-1)/2⌋) do in parallel
7. if A[2t] > A[2t+1] then exchange A[2t] and A[2t+1]

end.

In the above procedure we consider that the computation from line 2 to line 4 or from line 5 to line 7 takes one time unit (i.e., one stage).

Definition 1. Suppose that for an initial configuration of N input values in array A procedure BUBBLE(A[1 . . N], k) is executed. For value v ($1 \leqq v \leqq N$), position p ($1 \leqq p \leqq N$) in array A and stage s ($1 \leqq s \leqq k+1$) of the computation,

(1) PST(v, s) is the position of value v in A at the beginning of the s-th stage of the computation,

(2) VAL(p, s) is the value located at the p-th position of array A at the beginning of the s-th stage of the computation,

(3) LES(v, p, s) is the number of positions j such that PST(v, s) $< j \leqq p$ or PST(v, s) $> j \geqq p$, and $v >$ VAL(j, s),

(4) GRT(v, p, s) is the number of positions j such that PST(v, s) $< j \leqq p$ or PST(v, s) $> j \geqq p$, and $v <$ VAL(j, s),

(5) MAXLG(v, s) = max{LES(v, p, s) - GRT(v, p, s) | LP(v, s) $\leqq p \leqq$ PST(v, s)}, where LP(v, s) is the least position j such that $v \leqq$ VAL(j, s), and

(6) MAXGL(v, s) = max{ GRT(v, p, s) - LES(v, p, s) | PST(v, s) $\leqq p \leqq$ GP(v, s)}, where GP(v, s) is the greatest position j such that VAL(j, s) $\leqq v$.

Definition 2. Suppose that BUBBLE(A[1 . . N], k) is executed. For each value v ($1 \leqq v \leqq N$) and each stage s ($1 \leqq s \leqq k+1$) COUNT(v, s) is defined as follows:

(1) When GRT(v, 1, s) = 0, COUNT(v, s) = LES(v, N, s) + MAXGL(v, s).

(2) When LES(v, N, s) = 0, COUNT(v, s) = GRT(v, 1, s) + MAXLG(v, s).

(3) When GRT(v, 1, s) $\neq$ 0 and LES(v, N, s) $\neq$ 0, COUNT(v, s) = LES(v, N, s) + GRT(v, 1, s) + max{1, MAXLG(v, s), MAXGL(v, s)}.

Note that if LES(v, N, s) = 0 and GRT(v, 1, s) = 0, then COUNT(v, s) = 0 is from any of (1) and (2) of Definition. From Definition 2 COUNT(v, s) = 0 if and only if LES(v, N, s) = 0 and GRT(v, 1, s) = 0. Thus we have the next lemma.

Lemma 1. If COUNT(v, s) = 0, then value v does not move from the beginning of stage s to the end of the computation of BUBBLE(A[1 . . N], k).

Theorem 1. Suppose that BUBBLE(A[1 . . N], k) is executed. If $k \geq s \geq 2$ and COUNT(v, s) > 0, then COUNT(v, s+1) = COUNT(v, s) - 1.

Proof. The proof is by a case analysis. We assume that $s \geq 2$ and COUNT(v, s) > 0.

Case 1. GRT(v, 1, s) = 0, the values at PST(v, s) and at PST(v, s) +1 are compared at stage s, and v < VAL(PST(v, s) +1, s).

Since COUNT(v, s) > 0 and GRT(v, 1, s) = 0, LES(v, N, s) cannot be 0. Therefore, there exists at least one value i less than v such that PST(i, s) $\geq$ PST(v, s) + 2 and VAL(PST(i, s) - 1, s) > v. The first value less than v after PST(v, s) is one of such values. Since $s \geq 2$, for such any value i VAL(PST(i, s)-1, s) and i are compared and exchanged at stage s. Therefore, MAXGL(v, s+1) = MAXGL(v, s)-1. Since LES(v, N, s) = LES(v, N, s+1) in this case, COUNT(v, s+1) = COUNT(v, s) - 1.

Case 2. GRT(v, 1, s) = 0, the values at PST(v, s) and at PST(v, s) + 1 are compared at stage s, and v > VAL(PST(v, s) +1, s).

Since v and VAL (PST(v, s)+1, s) are exchanged at stage s, for any position j greater than PST(v, s) LES(v, j, s+1) = LES(v, j, s)-1. Since $s \geq 2$, for any value i less than v such that VAL(PST(i, s)-1, s) > v and PST(i, s) > PST(v, s), VAL(PST(i, s)-1, s) and i are also exchanged. Therefore, MAXGL(v, s) = MAXGL(v, s+1). Hence, COUNT(v, s+1) = LES(v, N, s+1)+MAXGL(v, s+1) = LES(v, N, s)-1 + MAXGL(v, s) = COUNT(v, s) - 1.

Case 3. GRT(v, 1, s) = 0 and the values at PST(v, s) and at PST(v, s) - 1 are compared at stage s.

Since GRT(v, 1, s) = 0, VAL(PST(v, s)-1, s) < v. Therefore, the proof is the same as that of Case 1. Note that this case includes the case where PST(v, s) = 1 and v is compared with the imaginary value at the imaginary position PST(v, s) - 1 (i.e., the case where v at the left end is not compared at stage s).

Case 4. LES(v, N, s) = 0, the values at PST(v, s) and at PST(v, s) -1 are compared, and VAL(PST(v, s) - 1, s) < v.

The proof is analogous to that of Case 1.

Case 5. LES(v, N, s) = 0, the values at PST(v, s) and at PST(v, s) -1 are compared at stage s, and VAL(PST(v, s)-1, s) > v.

The proof is analogous to that of Case 2.

Case 6. LES(v, N. s) = 0 and the values at PST(v, s) and at PST(v, s) + 1 are compared at stage s.

The proof is analogous to that of Case 3. Note that this case includes the case where PST(v, s) = N and v is compared with the imaginary value at the imaginary position PST(v, s) + 1.

Case 7. GRT(v, 1, s) > 0, LES(v, N, s) > 0, the values at PST(v, s) and at PST(v, s)+1 are compared at stage s, and v < VAL(PST(v, s) +1, s).

Note that in this case PST(v, s) is not 1 nor N. Since v does not move at stage s, LES(v, N, s) = LES(v, N, s+1) and GRT(v, 1, s) = GRT(v, N, s+1). Since $s \geq 2$, for any pair of values at position p and at position p+1 such that VAL(p, s) > VAL(p+1,s) these two values are compared and exchanged at stage s. Since in this case VAL(PST(v, s)-1, s) < v < VAL(PST(v, s)+1, s), both MAXLG(v, s) and MAXGL(v, s) are not 0. Therefore, MAXLG(v, s+1) = MAXLG(v, s)-1 and MAXGL(v, s+1) = MAXGL(v, s)-1. Since $s \geq 2$ and LES(v, N, s) > 0, VAL(PST(v, s)+1, s) < VAL(PST(v, s)+2, s) and there exists a position j such that PST(v, s)+2 < $j \leq$ N and VAL(j, s) < v. Therefore, MAXGL(v, s) ≥ 2. Hence, COUNT(v, s+1) = LES(v, s+1)+ GRT(v, s+1)+max{1, MAXLG(v, s+1), MAXGL(v, s+1)} = LES(v, s) + GRT(v, s) + max{1, MAXLG(v, s)-1, MAXGL(v, s)-1} = LES(v, s) + GRT(v, s) + max{ 1, MAXLG(v, s), MAXGL(v, s)} - 1 = COUNT(v, s) - 1.

Case 8. GRT(v, 1, s) > 0, LES(v, N, s) > 0, the values at PST(v, s) and at PST(v, s)+1 are compared at stage s, and v > VAL(PST(v, s) + 1, s).

GRT(v, 1, s+1) = GRT(v, 1, s) is immediate. Since v and VAL(PST(v, s), s) are exchanged at stage s, LES(v, N, s+1) = LES(v, N, s) - 1. Since $s \geq 2$ and GRT(v, 1, s) > 0, there exists a value i greater than v such that PST(i, s) < PST(v, s) - 1 and VAL(PST(i, s)+1, s) < v. Any of such values is exchanged with the value at its immediately right position at stage s. However, VAL(PST(v, s)+1, s) is moved to the left at stage s. Hence, MAXLG(v, s+1) = MAXLG(v, s). We can similarly show MAXGL(v, s+1) = MAXGL(v, s). Hence, COUNT(v, s+1) = COUNT(v, s) - 1.

Case 9. GRT(v, 1, s) > 0, LES(v, N, s) > 0 and the values at PST(v, s) and at PST(v, s)+1 are compared at stage s.

The proof is analogous to the proofs of Case 7 and Case 8. □

The next theorem is immediate from Theorem 1 and Lemma 1.

Theorem 2. Suppose that BUBBLE(A[1 . . N], N) is executed. Then for any value v ($1 \leq v \leq$ N) and any stage s ($2 \leq s \leq$ N), PST(v, s +

COUNT(v, s)) is the final position of v and v does not move after the end of stage s + COUNT(v, s) - 1.

Theorem 3. Suppose that BUBBLE(A[1 . . N], N) is executed. Then for any value v ($1 \leq v \leq N$)

(1) $0 \leq$ COUNT(v, 1) $\leq N$,

(2) COUNT(v, 2) = COUNT(v, 1) - 1 or COUNT(v, 2) = COUNT(v, 1), and

(3) if COUNT(v, 1) = N then COUNT(v, 2) = COUNT(v, 1) - 1 = N - 1.

Proof. Assertion (1) of the theorem is immediate from Definition 1 and Definition 2. For any value v ($1 \leq v \leq N$) the computation of BUBBLE at stage 1 reduces COUNT(v, 1) by at most one. Assertion (2) of the theorem is therefore true. The details of the proof are similar to the proof of Theorem 1. Suppose that COUNT(v, 1) = N. Then from Definition 1 and Definition 2 GRT(v, 1, 1) > 0 and LES(v, N, 1) > 0. Thus PST(v, 1) is not 1 nor N, and COUNT(v, 1) = LES(v, N, 1) + GRT(v, 1, 1) + max{1, MAXLG(v, 1) + MAXGL(v, 1)}. If at least one of MAXLG(v, 1) and MAXGL(v, 1) were not 0, COUNT(v, 1) should be at most N - 1. This is contrary to our assumption. Hence, MAXLG(v, 1) = MAXGL(v, 1) = 0 and LES(v, N, 1) + GRT(v, 1, 1) = N - 1. Therefore, v is compared and exchanged with the value at PST(v, 1) - 1 or at PST(v, 1) + 1 at stage 1. Hence, COUNT(v, 2) = LES(v, N, 1) + GRT(v, 1, 1) + 1 - 1 = N - 1. □

Cororally 1. Suppose that BUBBLE(A[1 . . N], N) is executed. Then for any value v ($1 \leq v \leq N$) v reaches its final position at the end of the COUNT(v, 1)-th stage or at the end of the (COUNT(v, 1) + 1)-th stage, and does not move after that stage.

The purpose behind introducing the concepts of function COUNT is now clear. As shown in the above results, COUNT(v, s) is an interesting characteristic of value v at stage s in the computation of BUBBLE(A[1 . . N], N). For $s \geq 2$, it indicates the exact number of stages necessary to move v to its final position. This characteristic is a powerful tool to design efficient parallel sort algorithms and to analyze them.

As in the case of the serial bubble sort, an obvious technique for improving the time efficiency of BUBBLE is to remember whether or not any change in A[1 . . N] has occured at each stage. If there is no exchange of values at a stage, then the sorting is completed and we may terminate the computation at the end of that stage. The improved version of BUBBLE in this way is called SBUBBLE and described as follows:

```
     procedure SBUBBLE(A[1 . . N], k, s);
     begin
1.     s:= 0
2.     repeat
3.       s:= s+1;  CHANGE:= false
4.       if s is odd then
5.         for all t (t = 1,. . .,⌊N/2⌋) do in parallel
6.           if A[2t-1] > A[2t] then
               begin
7.               exchange A[2t - 1] and A[2t];  CHANGE:= true
               end
8.       else       {in the case where s is even}
9.         for all t (t = 1,. . .,⌊(N-1)/2⌋) do in parallel
10.          if A[2t] > A[2t+1] then
               begin
11.              exchange A[2t] and A[2t+1]; CHANGE:= true
               end
12.    until (s ≥ 2 and CHANGE = false) or (s = k)
     end.
```

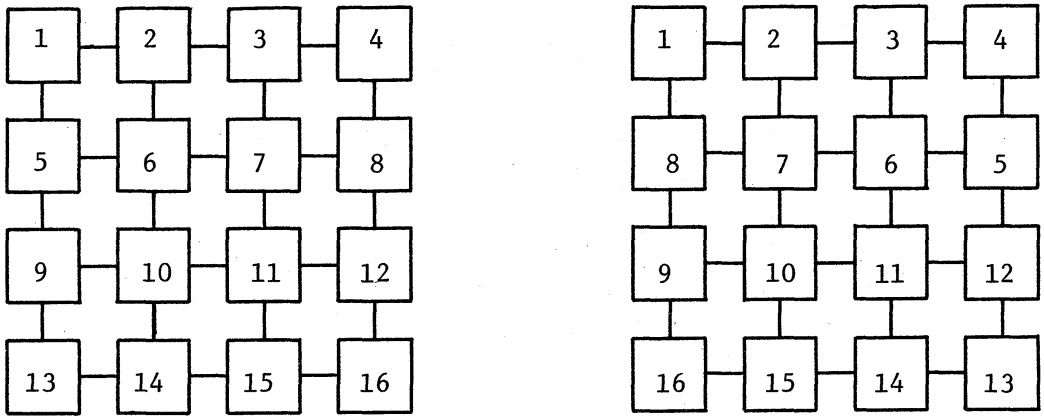
On the average the improvement of the time efficiency by SBUBBLE is marginal. The control hardware for implementing SBUBBLE is somewhat more complicated than that for implementing BUBBLE.

Let $\overline{\text{BUBBLE}}$ is a procedure obtained from BUBBLE by reversing the direction of the inequalities at line 4 and line 7. Thus $\overline{\text{BUBBLE}}$(A[1 . . N], N] is the parallel bubble sort in nonincreasing order for N input values. Similarly we define $\overline{\text{SBUBBLE}}$ by reversing the direction of the inequalities at line 6 and line 10 of SBUBBLE.

3. Sorting on a mesh-connected processor array

We assume a parallel computer with $N = n^2$ identical processors. The processors of this model are arranged in a two-dimensional square array A[1 . . n, 1 . . n]. Each processor contains a comparison-exchange element, registers and some circuit. If there is no confusion, the processor at location A[i, j] is denoted by its location. A[i, j] is directly connected to its neighbors A[i, j-1], A[i-1, j], A[i+1, j] and A[i, j+1], provided they exist. This model is the same as that used in [8,

9, 10]. The processors are indexed by an appropriate way. The row-major (or column-major) indexing and the snake-like row-major indexing are commonly accepted ways to order a two-dimensional array (see Fig. 1). In the worst case the computing time of any sort algorithm implemented on the mesh connected processor array cannot be smaller than $2n - 2$. The i-th row and the j-th column of A[1 . . n, 1 . . n] are denoted by



(a) Row-major indexing

(b) Snake-like row-major indexing

Fig. 1 Processor array indexing schemes

A[i, 1 . . n] and A[1 . . n, j], respectively. To simplify the proofs of the correctness of our algorithms we use the following well known result called the zero-one principle [5].

Theorem 4 (zero-one principle [5]). If a network with N input lines sorts all 2^N sequences of 0's and 1's, it will sort any arbitrary sequence of N numbers.

```
     procedure FHVBUBBLE(A[1 . . n, 1 . . n]);
     begin          {T(n^2) = (n+1) log n + 5 n/2 - 2}
1.     if n = 1 then return
2.     k:= n;
3.     for i:= 1 step 1 until ⌈log n⌉ do
         begin
4.         for all t (t = 1,. . .,n) do in parallel
5.           if t is odd then BUBBLE(A[t, 1 . . n], n)
6.           else BUBBLE(A[t, 1 . . n], n)
7.         for all t (t = 1,. . .,n) do in parallel
8.           BUBBLE(A[1 . . n, t], k);
```

```
9.          k:= ⌊(n+1)/2^(i+1)⌋ + 1
          end;
10.     for all t (t = 1,. . .,n) do in parallel
11.       if t is odd then BUBBLE(A[t, 1 . . n], n)
12.       else BUBBLE(A[t, 1 . . n], n)
        end.
```

Theorem 5. FHVBUBBLE(A[1 . . n, 1 . . n]) sorts n^2 input values in A into snake-like row-major nondecreasing order. The computing time of the procedure is $(n+1)\lceil \log n \rceil + 2n - 1 + \sum_{i=2}^{\lceil \log n \rceil} \lfloor (n+1)/2^i \rfloor$.

Proof. We assume that 0 or 1 is initially allocated to each entry of A[1 . . n, 1 . . n]. We first consider the configuration of A at the end of the computation from line 4 to line 6 in the first loop of the outermost "for statement". Let p and q be arbitrary column numbers such that $1 \leq p < q \leq n$. Let r_1 and r_2 be the number of 1's in the odd numbered entries of A[1 . . n, p] and the number of 1's in the even numbered entries of A[1 . . n, p], respectively. Let r_3 and r_4 be the number of 1's in the odd numbered entries of A[1 . . n, q] and the number of 1's in the even numbered entries of A[1 . . n, q], respectively. Then $0 \leq r_1, r_2, r_3, r_4 \leq \lceil n/2 \rceil$. Since each odd numbered row is sorted in nondecreasing order and each even numbered row is sorted in nonincreasing order from line 4 to line 6, $r_1 \leq r_3$ and $r_2 \geq r_4$. Hence, $|(r_1 + r_2) - (r_3 + r_4)| = |(r_2 - r_4) - (r_3 - r_1)| \leq \lceil n/2 \rceil$. At line 7 and line 8 each column is sorted in nondecreasing order. Therefore, all rows except at most $\lceil n/2 \rceil$ consecutive rows are sorted at the end of the first loop of the outermost "for statement". In the other words, all rows above the unsorted rows are all 0's, all rows below the unsorted rows are all 1's and the number of the unsorted rows is at most $\lceil n/2 \rceil$ at the end of the first loop of the outermost "for statement". Hence, just after the end of the computation from line 4 to line 6 in the second loop of the outermost "for statement", for any t $(1 \leq t \leq n)$ the length of the longest subword of A[1 . . n, t] in the form $(10 \ldots 10)^T$ is at most $2\lfloor (n+1)/4 \rfloor$, where α^T is the transposition of a sequence α. By Definition 2 the domain of COUNT is the set of ordered pairs of values and stages. In the case of 0-1 sequences the same values may be at different positions. To distinguish a value at a position from the same value at a different position, we use a notation such as VAL.A[i, t] which

means the value in A[i, t]. We may modify the domain of COUNT to be the set of pairs of values at positions and stages. For example, we may write COUNT(VAL.A[i, t], s) instead of writing COUNT(v, s), where v is the value in A[i, t] st stage s. This modification of the domain does not affect the results on COUNT.

Since for any t ($1 \leq t \leq n$) the unsorted subword of column A[1 . . n, t] just after the computation from line 4 to line 6 in the second loop of the outermost "for statement"is in the form $(10 \ . \ . \ 10)^T$ and since its length is at most $2\lfloor(n+1)/4\rfloor$, from Definition 2 for any i ($1 \leq i \leq n$) COUNT(VAL.A[i, t], 1) is at most $\lfloor(n+1)/4\rfloor$, where stage 1 of the second argument of COUNT means the first stage of the computation of BUBBLE(A[1 . . n, t], k) at line 8 in the second loop of the outermost "for statement". Therefore, from Theorem 2 and Theorem 3 the number of stages necessary to sort each column A[1 . . n, t] ($1 \leq t \leq n$) at line 8 in the second loop is at most $\lfloor(n+1)/4\rfloor + 1$. We can repeat this argument to each loop of the outermost "for statement". That is, for the i-th loop of the outermost "for statement" the number of stages necessary to sort each column A[1 . . n, t] ($1 \leq t \leq n$) by BUBBLE at line 8 is at most $\lfloor(n+1)/2^i\rfloor + 1$. Hence, the value of the second argument of BUBBLE at line 8 is enough to sort each column. When the number of unsorted rows becomes 1, the computation escapes from the loop and the values in A are completely sorted in snake-like row-major ordering by the computation from line 10 to line 12.

The computing time of FHVBUBBLE is evaluated as follows: In each loop of the outermost "for statement" at line 3 the computation from line 4 to line 6 takes n stages. In the first loop of the outermost "for statement" at line 3 the computation from line 7 to line 8 takes n stages. In the i-th loop ($2 \leq i \leq \lceil \log n \rceil$) of the outermost "for statement" the computation from line 7 to line 8 takes $\lfloor(n+1)/2^i\rfloor + 1$ stages. The computing time from line 10 to line 12 is n stages. Therefore, the total number of stages of the computation of FHVBUBBLE for n^2 input values is

$$n\lceil \log n \rceil + 2n + \sum_{i=2}^{\lceil \log n \rceil} (\lfloor(n+1)/2^i\rfloor + 1)$$

$$= (n+1)\lceil \log n \rceil + 2n + \sum_{i=2}^{\lceil \log n \rceil} \lfloor(n+1)/2^i\rfloor. \qquad \square$$

We may replace BUBBLE's at line 5, line 8 and line 11 of FHVBUBBLE by SBUBBLE's and may replace $\overline{\text{BUBBLE}}$'s at line 6 and line 12 of FHVBUBBLE by $\overline{\text{SBUBBLE}}$'s. Then the average computing time of FHVBUBBLE improves

somewhat by this modification, but its control hardware becomes somewhat more complicated.

Corollary 2. When n is a power of 2, the computing time of FHVBUBBLE for n^2 input values is $(n+1)\log n + 5n/2 - 2$.

Cororally 3. If we replace the statements at line 11 and line 12 of FHVBUBBLE by BUBBLE(A[t, 1 . . n], n), then it sorts n^2 input values in A into row-major nondecreasing order and its computing time is the same as that of FHVBUBBLE.

We can obtain an algorithm from FHVBUBBLE by changing the order of the alternation of the horizontal and vertical parallel bubble sorts in FHVBUBBLE. This modified algorithm is somewhat less efficient than FHVBUBBLE. As shown in Theorem 5 the asymptotic computing time of FHVBUBBLE is worse than those of the implementations given in [6, 8, 10]. However, the control hardware of our algorithm is much simpler than that of the implementations of Batcher's algorithms and its computing time for practical values of n is not much different from that of Batcher's algorithms. We finally present a variation of our algorithm which is faster than any implementation reported in [6, 8, 10] for practical values of n, $1 \leq n \leq 128$.

```
     procedure SVHBUBBLE(A[1 . . n, 1 . . n]);
     begin        {good performance for random data}
1.     if n = 1 then return;
2.     for all t (t = 1,. . .,n) do in parallel
3.       SBUBBLE(A[1 . . n, t], n sc(t));
4.     COMPLETE:= false;  k:= ⌊n/2⌋ + 1;
5.     while not COMPLETE do
         begin
6.         if k ≤ 2 then COMPLETE:= true
7.         for all t (t = 1,. . .,n) do in parallel
8.           if t is odd then
9.             SBUBBLE(A[t, 1 . . n], n, sr(t))
10.          else SBUBBLE(A[t, 1 . . n], n, sr(t));
11.        for all t (t = 1,. . .,n) do in parallel
12.          SBUBBLE(A[1 . . n, t], k, sc(t));
13.        k:= ⌊max{sc(t) | 1 ≤ t ≤ n}/2⌋ + 1
         end;
```

```
14.    for all t (t = 1,. . .,n) do in parallel
15.      if t is odd then
16.        SBUBBLE(A[t, 1 . . n], n, sr(t))
17.      else SBUBBLE(A[t, 1 . . n], n, sr(t))
       end.
```

The correctness of SVHBUBBLE can be shown in the same way as the proof of Theorem 5, but it seems to be difficult to evaluate analytically its computing time.

References

1. Ajtai, M., Komlos, J., and Szemeredi, E., An O(N log N) sorting network, Proc. of 15th Annual ACM Symp. on Theory of Computing, 1983, pp. 1-9.
2. Batcher, K. E., Sorting networks and their applications, Proc. of AFIPS Spring Joint Computer Conf., 32, 1968, pp. 307-314.
3. Bilardi, G., and Preparata, F., A minimum area VLSI architecture for O(log N) time sorting, Proc. of 16th Annual ACM Symp. on Theory of Computing, 1984, pp. 64-70.
4. Bitton, D., Dewitt, D. J., Hsiao, D. K., and Mrnon, J., A texonomy of parallel sorting, Tech. Rep. TR 84-601, Cornell Univ., Ithaca, New York, 1984.
5. Knuth, D. E., The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.
6. Kumar, M., and Hirschberg, D. S., An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes, IEEE Trans. Comput., C-32 (1983), pp. 254-264.
7. Leighton, T., Tight bounds on the complexity of parallel sorting, Proc. of 16th Annual ACM Symp. on Theory of Computing, 1984, pp. 71-80.
8. Nassimi, D., and Sahni, S., Bitonic sort on a mesh-connected parallel computer, IEEE Trans. Comput., C-27 (1979), pp. 2-7.
9. Thompson, C. D., The VLSI complexity of sorting, IEEE Trans. Comput., C-32 (1983), pp. 1171-1184.
10. Thompson, C. D., and Kung, H. T., Sorting on a mesh-connected parallel computer, Commun. ACM, 20 (1977), pp. 263-271.
11. Ullman, J. D., Computational Aspects of VLSI, Computer Science Press, Rockville, Maryland, 1984.