Odd-even, compare-exchange parallel sorting

Stavros D. Nikolopoulos*a,*, Stylianos D. Danielopoulosb,†

*Department of Computer Science, University of Cyprus, 75 Kallipoleos str, P.O. Box 537, Nicosia, Cyprus
bDivision of Applied Mathematics and Informatics, University of Ioannina, GR-451 10 Ioannina, Greece

Received 12 May 1993; Revised 16 November 1993; Accepted 6 April 1994

Abstract

We present a parallel sorting algorithm and its proof which sorts a sequence of $n$ elements in time $O(\log^2 n)$ with $n/2$ processors on an EREW-PRAM computational model. A sorting network directly implements the algorithm using $O(n \cdot \log n)$ PEs. The algorithm is based on the elementary Compare-Exchange operation and has the advantage that it does not require a powerful computational model, uses the least amount of space for the sorting problem, has small constants and can be implemented directly on a sorting network. Furthermore, the architecture of the network is simple and makes no unrealistic technological assumptions.

Key words: Parallel sorting; Compare-exchange schemes; EREW-PRAM; Sorting networks; Complexity

1. Introduction

For both practical and theoretical reasons, sorting is probably the most well-studied problem in computer science. Recently parallel sorting has received more and more attention because of the advances in computer technology. Many parallel algorithms have been developed to sort a sequence of $n$ numbers on a variety of parallel computational models, such as PRAMs [2, 4, 6, 8, 10, 16, 18, 23], Linear Array [11, 22], Perfect Shuffle [19], Mesh Connected [11, 14, 20], Cube connected [15, 17], and Special-purpose sorting networks [1, 4, 9, 12, 13, 21].

Since reduction of execution time is the principal reason for considering parallelism, algorithms are often classified according to time complexity. Most published parallel sorting algorithms fall into four categories: (I) $O(\log n)$, (II) $O(\log^2 n)$, (III) $O(n^{1/2})$ and (IV) $O(n)$. The $O(\log n)$ time algorithms have
the best theoretical performance, but most suffer from practical disadvantages such as reliance on powerful parallel computational models [10, 18], requirement of large amount of space [8], unsuitability for sorting networks [5], existence of large constants [1], and limitations on the kind of data handled [6]. Algorithms on other categories trade theoretical performance against practical considerations.

In this paper we present a parallel sorting algorithm which sorts a sequence of \( n \) elements in time \( G(\log_2 n) \) using \( n/2 \) processors on an EREW-PRAM computational model. The algorithm is quite simple and based on the elementary operation Compare-Exchange. The algorithm has the advantage that it uses the weakest PRAM model, i.e. EREW-PRAM, rather than the CREW-PRAM or CRCW-PRAM. It requires the least space for the sorting problem, has small constants and translates directly into a sorting network. Furthermore, the architecture of the network is quite simple and makes no unrealistic technological assumptions.

The remainder of the paper is organised as follows. In Section 2, we introduce a type of undirected graphs \( G = (V, E) \) and construct a scheme as a sequence of such graphs. In Section 3, we present the sorting algorithm and we prove its correctness. An EREW-PRAM implementation of the algorithm is given in Section 4. In the last section, we design a special-purpose sorting network which directly implements the algorithm. Throughout the paper all logarithms are base two.

2. The compare-exchange scheme \( \text{OECE} [n] \)

An undirected graph \( G = (V, E) \) is a structure which consists of a finite, nonempty set \( V \) of \( n = |V| \) elements called vertices and a set \( E = \{(i, j) | i, j \in V, i \neq j\} \) of \( e = |E| \) two element subsets of \( V \) called edges [7]. For simplicity, we will usually denote edges by pairs of vertices. For example, the edge \( \{x, y\} \) would be denoted \( (x, y) \) or \( (y, x) \), for \( x, y \in V \). Thus, in an undirected graph the pair \( (x, y) \) and \( (y, x) \) represent the same edge \( \{x, y\} \in E \).

We define a compare-exchange graph to be a graph \( G = (V, E) \) with the following properties:

(i) \( V = \{1, 2, \ldots, n\} \),
(ii) \( |E| = n/2 \),
(iii) for every pair of edges \( (i, j) \in E \) and \( (k, m) \in E \), 
\[ i \neq k, i \neq m, j \neq k \text{ and } j \neq m. \]

We shall denote a compare-exchange graph \( G = (V, E) \) with \( n \) vertices by \( G(n) = (V_G, E_G) \) (see Fig. 1).

Given two compare-exchange graphs \( X(p) = (V_X, E_X) \) and \( Y(q) = (V_Y, E_Y) \), \( p = |V_X| \) and \( q = |V_Y| \), we define the graph union of \( X(p) \) and \( Y(q) \), denoted by \( X(p) \cup Y(q) \), to be a compare-exchange graph \( Z(n) = (V_Z, E_Z) \), \( n = |V_Z| \), with the following properties:

(a) \( V_Z = \{1, 2, \ldots, p, p+1, \ldots, p+q\} \),
(b) if \( (i, j) \in E_X \) then \( (i, j) \in E_Z \), and
if \( (k, m) \in E_Y \) then \( (p+k, p+m) \in E_Z \), \( 1 \leq i, j < p \), \( 1 \leq k, m \leq q \).

By definition \( n = p + q \). Two compare-exchange graphs \( X(p) = (V_X, E_X) \) and \( Y(q) = (V_Y, E_Y) \) are said to be equal, if (i) \( |V_X| = |V_Y| \), i.e. \( p = q \), and (ii) for every \( (i, j) \in E_X \), \( (i, j) \in E_Y \). By the notion \( X(p) = Y(q) \), we mean that \( X(p) \) and \( Y(q) \) are equal graphs; otherwise \( X(p) \neq Y(q) \) (see Fig. 2).

A compare-exchange scheme is defined to be a sequence of \( L \) compare-exchange graphs \( S_1(n), S_2(n), \ldots, S_L(n) \); we denote this scheme by \( S[n] = (S_1(n), S_2(n), \ldots, S_L(n)) \), \( L \geq 1 \). The magnitude \( L \) is called length of the compare-exchange scheme \( S[n] \) and denoted by \( \text{length}(S[n]) \).

Given two compare-exchange schemes \( A[p] = (A_1(p), A_2(p), A_3(p), \ldots, A_L(p)) \) and \( B[q] = (B_1(q), B_2(q), \ldots, B_L(q)) \) of length \( L \), we define the union of \( A[p] \) and \( B[q] \), denoted by \( A[p] \cup B[q] \), to be a compare-exchange scheme \( C[n] = (C_1(n), C_2(n), \ldots, C_L(n)) \), with the following properties:

(a) \( \text{length}(C[n]) = L \),
(b) \( C_i(n) = A_i(p) \cup B_i(q), i = 1, 2, \ldots, L \).

We consider two compare-exchange graphs \( \text{ODD}(n) = (V_{\text{ODD}}, E_{\text{ODD}}) \) and \( \text{EVEN}(n) = (V_{\text{EVEN}}, \ldots) \).
Fig. 2. Two compare-exchange graphs $X(4)$ and $Y(4)$, and the unions $Z(8) = X(4) \uplus Y(4)$ and $Q(8) = Y(4) \uplus X(4)$. Obviously, $Z(8) \neq Q(8)$.

Next, we define the compare-exchange scheme $OECE[n]$ (Odd-Even Compare-Exchange scheme) recursively as follows:

$OECE[2] = (ODD(2)) = (EVEN(2))$

$OECE[n] = (ODD(n), EVEN(n), OECE[n/2] \uplus OECE[n/2])$

where $n = 2^k$, $k \geq 2$ (see Fig. 3).

Let us compute the length $L$ of the compare-exchange scheme $OECE[n]$. The definition of $OECE[n]$ implies that

$length(OECE[2]) = 1,$

$length(OECE[n]) = 2 + length(OECE[n/2]).$

Solving the above recurrence, we obtain

$length(OECE[n]) = 2(\log n - 1) + length(OECE[2]).$
Since the length of the scheme \( OECE[2] = (ODD(2)) = (EVEN(2)) \) is 1, the length length \( (OECE[n]) \) of \( OECE[n] \) is given by

\[
length(OECE[n]) = 2(\log n - 1) + 1 = 2\log n - 1.
\]

Let \( S[n] = (S_1(n), S_2(n), \ldots, S_L(n)) \) be a compare-exchange scheme of length \( L > 1 \). For simplicity, in the rest of the paper, we shall denote the compare-exchange graph \( S_i(n) \) of the scheme \( S[n] \) by \( S(n, i), 1 \leq i \leq L \).

Thus, the compare-exchange scheme

\[
OECE[n] = (OECE_1(n), OECE_2(n), \ldots, OECE_L(n))
\]

will be denoted by

\[
OECE[n] = (OECE(n, 1), OECE(n, 2), \ldots, OECE(n, L)),
\]

where \( L = 2\log n - 1 \).

It is easy to see that the compare-exchange scheme \( OECE[n] \), defined above, has the following properties:

(i) \( length(OECE[n]) = 2\log n - 1 \),

(ii) \( OECE(n, 1) = OECE(n, 3) = \ldots = OECE(n, L) = ODD(n) \),

(iii) \( OECE(n, 2) = EVEN(n) \), and

\[ OECE(n, 2) \neq OECE(n, 4) \neq \ldots \neq OECE(n, L - 1), \]

where \( L = 2\log n - 1 \).

3. The Odd-Even Compare-Exchange Algorithm

In this section, we present a parallel algorithm, which we propose to call Odd-Even Compare-Exchange Sort (hereafter referred to as OECE-Sort), for sorting a sequence \( S = (x_1, x_2, \ldots, x_n) \) of \( n \) elements, where \( n = 2^k \) for some integer \( k \geq 1 \). We assume that each element \( x_i \) assumes a value from a set on which a total order is defined. Furthermore, we assume that all elements of \( S \) are pairwise distinct, i.e., for any elements \( x_i, x_j \) we have the property that, if \( i \neq j \), either \( x_i < x_j \) or \( x_j < x_i \). The assumption that \( x_i \neq x_j \) for \( i \neq j \) simplifies the presentation of the algorithm without any sacrifice, since the algorithm works correctly in the presence of equal elements.

Given the compare-exchange scheme \( OECE[n] \) and a sequence \( S \) of length \( n \), the high-level idea behind the algorithm proposed here is quite simple. The algorithm sorts the sequence \( S \) into increasing (or decreasing) order in \( R \cdot L \) steps. The work done in each step is as follows: First, a compare-exchange graph \( OECE(n, l) \) of the scheme \( OECE[n] \) is associated to the step, \( 1 \leq l \leq L \), where \( L \) is the length of the scheme \( OECE[n] \), and then \( n/2 \) pairs of elements of the sequence \( S \) are selected (the pair \( (x_i, x_j) \) is selected if \( (i, j) \) is an edge of the associated graph \( OECE(n, l) \), \( i < j \)); the elements of each pair are compared under relation ' < '; and they exchange positions where necessary. Thus, the sorting algorithm OECE-Sort can be described as follows:

\[
\text{for } r \leftarrow 1 \text{ to } R
\]

\[
\text{for } l \leftarrow 1 \text{ to } L
\]

\[
\text{Step: Compare-Exchange}
\]

for every edge \((i, j), i < j, \) of the compare-exchange graph \( OECE(n, l) \),

if \( x_i > x_j \) then interchange \( x_i \) with \( x_j \).

end;

end;

where the term \( R \) is the number of times the compare-exchange scheme \( OECE[n] \) must be repeated to ensure complete sorting, and \( L \) is the length of the scheme.

As stated earlier, each compare-exchange graph of the scheme \( OECE[n] \) has vertex set \( \{1, 2, \ldots, n\} \) and every vertex belongs to exactly one edge, i.e., if
and \((p, q)\) are two edges, then \(i \neq p, i \neq q, j \neq p\)
and \(j \neq q\). Therefore, in each step of the algorithm,
the \(n/2\) selected pairs of elements of the sequence
\(S\) are disjoint.

Next, we shall establish the notation and terminology employed in our paper. Let \(S = \{x_1, x_2, \ldots, x_n\}\) be a sequence of \(n\) elements, \(n = 2^k, k \geq 1\).
Throughout the text, when we say 'apply the compare-exchange operation \(OECE(n, l)\) to \(S\)',
we mean that a step of the algorithm is executed with corresponding compare-exchange graph \(OECE(n, l)\), \(1 \leq l \leq L\).
Moreover, when we say 'apply the compare-exchange operation \(OECE[n]\) to \(S\)',
we mean that we apply the compare-exchange operations \(OECE(n, 1), OECE(n, 2), \ldots, OECE(n, L)\)
to \(S\), i.e. \(L\) steps of the algorithm are executed; the 1st step with corresponding graph \(OECE(n, 1)\), the 2nd step with corresponding graph \(OECE(n, 2)\), \ldots, the \(L\)th step with corresponding graph \(OECE(n, L)\),
where \(L = 2 \log n - 1\). Suppose that the sequence \(S\) is sorted in increasing order.
We call small the \(n/2\) elements of \(S\) located in low-order positions 1, 2, \ldots, \(n/2\)
and large the \(n/2\) elements of \(S\) located in high-order positions \(n/2 + 1, n/2 + 2, \ldots, n\).
Given a sequence \(S = \{x_1, x_2, \ldots, x_n\}\), we refer to positions 1, 2, \ldots, \(n/2\) as lower half of \(S\), and to positions \(n/2 + 1, n/2 + 2, \ldots, n\) as upper half of \(A\).

**Lemma 3.1.** Let \(S\) be a sequence of \(n\) elements, from a totally ordered set, sorted in decreasing order,
where \(n = 2^k\) for some integer \(k \geq 2\). If we apply the compare-exchange operations \(OECE(n, 1), OECE(n, 2)\) and \(OECE(n, 3)\) to \(S\), then the sequence \(S\) is sorted in increasing order.

**Proof.** It is easy to verify that, applying the operations \(OECE(n, 1), OECE(n, 2)\) and \(OECE(n, 3)\) to a sequence \(S\) sorted in decreasing order, the results will be sorted in increasing order. (We remind the reader that \(OECE(n, 1) = OECE(n, 3) = ODD(n)\) and \(OECE(n, 2) = EVEN(n);\) see Section 2.)

**Lemma 3.2.** Let \(S\) be a sequence of \(n\) elements, where \(n = 2^k\) for some integer \(k > 2\). We apply the compare-exchange operations \(OECE(n, 1)\) and \(OECE(n, 2)\) to \(S\). Then, at most \(n/4\) large elements remain in the lower half of \(S\), and consequently at most \(n/4\) small elements remain in the upper half of \(S\).

**Proof.** We apply the operations \(OECE(n, 1)\) and \(OECE(n, 2)\) to \(S\) and we assume that \(n/4 + 1\) large elements remain in the lower half of \(S\), i.e. at positions 1, 2, \ldots, \(n/2\). That means, both positions \(i\) and \(n + 1 - i\) hold large elements, \(1 \leq i \leq n/2\). Therefore, sequence \(S\) contains \(2(n/4 + 1) = n/2 + 2\) large elements, which is absurd.

**Lemma 3.3.** Let \(S\) be a sequence of \(n\) elements, where \(n = 2^k\) for \(k = 2, 3\). If we apply \(k - 1\) times the compare-exchange operation \(OECE[n]\) to \(S\), then the sequence is sorted in increasing order.

**Proof.** (i) \(k = 2\). Then, the sequence \(S\) is of length \(n = 4\) and the compare-exchange scheme \(OECE[4]\) of length \(2 \log(4) - 1 = 3\). Applying the operations \(OECE(4, 1)\) and \(OECE(4, 2)\) to \(S\), we obtain that 2 small elements move into the lower half of \(S\) and 2 large elements into the upper half. Obviously, applying the operation \(OECE(4, 3)\), sequence \(S\) is sorted in increasing order.

(ii) \(k = 3\). In this case, \(n = 8\) and the compare-exchange scheme \(OECE[8]\) is of length \(2 \log(8) - 1 = 5\) (see Fig. 3). We consider the worst-case in which, after the operations \(OECE(8, 1)\) and \(OECE(8, 2)\), 2 large elements have been remained in the lower half of \(S\) and therefore, 2 small elements in the upper half of \(S\) (Lemma 3.2). That means, each small (large) element of the lower
half is smaller than every small (large) element of the upper half of S. Applying now, the operation OECE(8, 3), OECE(8, 4) and OECE(8, 5), the 4 elements of the lower half and the 4 elements of the upper half are sorted in increasing order. Thus, after the application of the operation OECE[8], the sequence has the following form

\[
\{\{2 \text{ small}\} \{2 \text{ large}\} \{2 \text{ small}\} \{2 \text{ large}\}\}
\]

where \( S_1 < S_3 \) and \( S_2 < S_4 \).

We now apply again the operation OECE[8] to sequence S. Obviously, applying the operation OECE(8, 1), no element changes location since \( x_i < x_{i+1} \), \( i = 1, 3, 5 \) and 7. But, applying the operation OECE(8, 2), the elements of \( S_1 \) and \( S_2 \) are compared with those of \( S_4 \) and \( S_3 \), respectively, and since \( S_2 > S_3 \), all the elements of \( S_2 \) (\( S_3 \)) move to \( S_3 \) (\( S_2 \)). Finally, applying the operations OECE(8, 3), OECE(8, 4) and OECE(8, 5), it is easy to see that, the sequence S is sorted in increasing order.

Theorem 3.1. Let S be a sequence of n elements, where \( n = 2^k \) for some integer \( k \geq 2 \). If we apply \( k - 1 \) times the compare-exchange operation OECE[n] to S, then the sequence is sorted in increasing order.

Proof. We shall prove the theorem by induction on the length n of the sequence. Lemma 3.3 implies that, in the base cases \( k = 2 \) or \( k = 3 \), the theorem is true. Assume that a sequence S of length \( n = 2^k \), \( k \geq 3 \), is sorted in increasing order by applying \( k - 1 \) times the operation OECE[2^k]. We shall prove that a sequence S of length \( n = 2^{k+1} \), \( k \geq 3 \), is sorted in increasing order by applying \( k \) times the operation OECE[2^{k+1}].

Let us consider the case where we apply \( k - 1 \) times the operation OECE[2^{k+1}] to a sequence S of length \( n = 2^{k+1} \), \( k \geq 3 \), and \( n/4 \) large elements remain in the lower half of S and \( n/4 \) small in the upper half of S. This obviously is the worst-case (see Lemma 3.2). Analyzing the worst-case, we are led to the following two conclusions:

(a) Since \( n/4 \) large and \( n/4 \) small elements are placed in the lower and upper half of S respectively, there follows that in each of the \( k - 1 \) applications of the operations OECE(2^{k+1}, 1) and OECE(2^{k+1}, 2), large (small) elements were compared only with large (small) elements. Moreover, since

\[
\text{OECE}[2^{k+1}] = (\text{OECE}(2^{k+1}, 1), \text{OECE}(2^{k+1}, 2), \text{OECE}[2^k] \oplus \text{OECE}[2^k])
\]

it follows that, \( k - 1 \) times the operation OECE[2^k] was applied to the lower half of S and \( k - 1 \) times it was applied to the upper half of S. Then, from the induction hypothesis, we have that the \( n/2 = 2^k \) elements of the lower half of S and the \( n/2 = 2^k \) elements of the upper half of S are sorted in increasing order.

Thus, after the \( (k - 1) \)th application of the operation OECE[2^{k+1}], sequence S has the following form

\[
\{\{n/4 \text{ small}\} \{n/4 \text{ large}\} \{n/4 \text{ small}\} \{n/4 \text{ large}\}\}
\]

where \( S_1 < S_2 \) and \( S_3 < S_4 \).

(b) As we mentioned in (a), the \( n/4 \) small elements of \( S_1 \) were compared with the \( n/4 \) small elements of \( S_3 \) \( k - 1 \) times, i.e. the operations OECE(2^{k+1}, 1) and OECE(2^{k+1}, 2) were applied in \( n/2 = 2^k \) elements \( k - 1 \) times. From the way in which the compare-exchange scheme OECE[2^{k+1}] is constructed, the operation OECE[2^{k-1}] was applied \( k - 2 \) times to the \( n/4 = 2^{k-1} \) small elements of the lower half of S and \( k - 2 \) times to the \( n/4 = 2^{k-1} \) small elements of the upper half of S. Then, by the induction hypothesis, we have that the \( n/2 = 2^k \) small
elements of $S$ are sorted in increasing order, and therefore all the elements of $S_1$ are smaller than the elements of $S_3$, i.e. $S_1 < S_3$. Following a similar reasoning, we conclude that $S_2 < S_4$.

We now apply once the operation $\text{OECE}[^{2k+1}]$ to sequence $S$. Obviously, the application of the operation $\text{OECE}[^{2k+1}, 1]$ has no effect in $S$, i.e. no exchanges of elements take place. But, since $S_2 > S_3$, the application of the operation $\text{OECE}[^{2k+1}, 2]$ cause all elements of $S_2$ to be exchanged with these of $S_3$. Therefore, applying the operations $\text{OECE}[^{2k+1}, 1]$ and $\text{OECE}[^{2k+1}, 2]$, we have:

(i) The $n/2$ small and the $n/2$ large elements are located in the lower and the upper half of $S$, respectively.

(ii) The elements in $S_1$ and $S_4$ are sorted in increasing order, while the elements in $S_2$ and $S_3$ are sorted in decreasing order.

Applying the operations $\text{OECE}[^{2k+1}, 5]$, $\text{OECE}[^{2k+1}, 6]$ and $\text{OECE}[^{2k+1}, 7]$, we obtain that the elements of $S_2$ and $S_3$ are in increasing order (Lemma 3.1).

Therefore, the elements of a sequence $S$ of length $n = 2^k$, $k \geq 2$, can be sorted in increasing order by applying $k = \log n - 1$ time the operation $\text{OECE}[^n]$ to sequence $S$. □

Theorem 3.1 proves the correctness of algorithm OECE-Sort. Moreover, it provides us with the number of times $R = \log n - 1$ we have to apply the compare-exchange operation $\text{OECE}[^n]$ to a sequence $S$ in order to be sorted.

4. An EREW-PRAM implementation

We now present the implementation of the sorting algorithm OECE-Sort on an Exclusive-Read Exclusive-Write Parallel RAM computational model (EREW-PRAM).

An EREW-PRAM consists of a set of $n$ processors operating in parallel. Each of them is a sequen-
tial RAM with own local memory. The processors are identified with distinct integers $m$, $1 \leq m \leq n$, called labels. All processors have access to a common memory for both reading and writing operations. Simultaneous reading by several processors from the same memory location as well as simultaneous writing to the same memory location is not allowed (Exclusive-Read Exclusive-Write).

Let the computational model used consist of $n/2$ processors $P_m$, $m = 1, 2, \ldots, n/2$. We assume here, for convenience, that the elements of sequence $S$ are stored in an array $A[1..n]$. We assume moreover that the array $A[1..n]$ occupies contiguous locations in the common memory. The sorting problem is considered solved when the array elements have been sorted in increasing order, i.e. $A[i]$ is less than $A[i+1]$ for $i = 1, 2, \ldots, n-1$.

At the $k$th step of the algorithm, $1 \leq k \leq R \cdot L$, each processor executes the following operations:

(i) compare a pair $(i, j)$, $1 \leq i < j \leq n$,

(ii) compare the elements at locations $i$ and $j$, i.e. $A[i]$ and $A[j]$,


The pair $(i, j)$ is computed to be an edge of the compare-exchange graph $\text{OECE}(n, 1)$ which corresponds to the $k$th step of the algorithm, $1 \leq l \leq L$. Consequently, all the $n/2$ processors compute simultaneously $n/2$ pairs $(i, j)$, $1 \leq i, j \leq n$. In the $k$th step of the algorithm, $1 \leq k \leq R \cdot L$, each processor $P_m$, $1 \leq m \leq n/2$, knows its own label $m$, the length $n$ of the array $A$ and the step $k$. It is easy to see that, the step $k$ of the algorithm and the order $l$ of the compare-exchange graph $\text{OECE}(n, l)$, $1 \leq l \leq L$, are related by $l = k \mod L$, where $L = 2 \log n - 1$.

Thus, our intention is to state formulae which relate the pair $(i, j)$ with the label $m$, the number $n$ and the order $l$.

By the definition of the compare-exchange scheme $\text{OECE}[^{n}]$ (see Section 2), we have

$$\text{OECE}(n, 1) = \text{OECE}(n, 3) = \cdots = \text{OECE}(n, l) = \text{ODD}(n).$$
Therefore, the \( n/2 \) edges \((i,j)\) of the above compare-exchange graphs are given by
\[
(i,j) = (2m - 1, 2m)
\]
where \( m = 1, 2, \ldots, n/2 \) and \( l \) is an odd number.

Let us now analyze the case where \( l \) is an even number. We assume that in the \( k \)th step of the algorithm the compare-exchange operation \( \text{OECE}(n, l) \) is applied to array \( A \), where \( k = c \cdot L + l \), \( c \geq 0 \), \( 1 \leq l \leq L \) and \( L = \text{length}(\text{OECE}[n]) \).

We partition the array \( A[1..n] \) into \( p \) subarrays \( A_1, A_2, \ldots, A_p \), each of length \( n/p \), where
\[
p = 2^{\lceil \log n \rceil - 1}
\]
and
\[
A_q = A\left[(q-1)(n/p) + 1 \ldots k(n/p)\right],
\]
\( 1 \leq q \leq p \).

We also partition the set \( Q \) of the \( n/2 \) processors into \( p \) subsets \( Q_1, Q_2, \ldots, Q_p \), each of \( n/2p \) processors, where
\[
Q_q = \left\{ P_{(q-1)(n/2p)+1}, P_{(q-1)(n/2p)+2}, \ldots, P_{q(n/2p)} \right\},
\]
\( 1 \leq q \leq p \).

Each processor \( P_m \) of the set \( Q_q \), \( 1 \leq q \leq p \), computes a pair \((i,j)\) in such a way that
\[
(q-1)(n/p) + 1 \leq i < j \leq q(n/p)
\]
and
\[
(i,j) \text{ is an edge of } \text{OECE}(n, l),
\]
\( 1 \leq l \leq L \). This computation is carried out in the following way:
(a) An index \( q \), \( 1 \leq q \leq p \), indicating the set \( Q_q \) which contains processor \( P_m \), \( 1 \leq m \leq n/2 \), is computed using the following formula
\[
q = \text{trunc}\left(\frac{m-1}{n/2p}\right) + 1.
\]
(b) An integer \( i \) of the interval \([(q-1)(n/p) + 1, (q-1)(n/p) + (n/2p)]\) is computed using the following formula
\[
i = m + (q - 1)\left(\frac{n}{2p}\right).
\]
(c) An integer \( j \) of the interval \([(q-1)(n/p) + (n/2p) + 1, q(n/p)]\) is computed using the following formula
\[
j = q - \frac{n}{p} - \left(i - \left(q - 1\right)\frac{n}{2p} + 1\right).
\]

The pair \((i,j)\) computed above is an edge of the compare-exchange graph \( \text{OECE}(n, l) \) of the scheme \( \text{OECE}[n] \), where \( l \) is an even number, \( 1 \leq l \leq L \).

Thus, the \( n/2 \) pairs \((i,j)\) computed in the \((c \cdot L + l)\)th step of the algorithm, \( c \geq 0 \), \( 1 \leq l \leq L \), are given as a function of \( m, n \) and \( l \), by
\[
(i,j) = (2m - 1, 2m)
\]
when \( l \) is odd, and by
\[
(i,j) = \left(m + (q - 1)\frac{n}{2p}, q - \frac{n}{p} - \left(m + (q - 1)\frac{n}{2p} + 1\right)\right),
\]
when \( l \) is even, where \( q = \text{trunc}((m - 1)/(n/2p)) + 1 \), \( m = 1, 2, \ldots, n/2 \) and \( L = 2\log n - 1 \).

The sorting algorithm \( \text{OECE-Sort} \) (Odd-Even Compare-Exchange Sort) for an EREW-PRAM computational model is given in Fig. 4.

4.1. Time and processor analysis

We will now determine the time and processor complexity of the proposed parallel sorting algorithm \( \text{OECE-Sort} \). Theorem 3.1 implies that, in order to sort a sequence \( S \) of length \( n \) using algorithm \( \text{OECE-Sort} \), the Compare-Exchange operation \( \text{OECE}[n] \) must be applied to sequence \( R \) times, i.e.
\[
R = \log n - 1.
\]
Algorithm OECE-Sort;
begin
for \( r \leftarrow 1 \) to \( \log n - 1 \) do begin
  for \( l \leftarrow 1 \) to \( 2\log n - 1 \) do begin
    for all processors \( P_m, m = 1, 2, \ldots, n/2 \) in parallel
      if \( l \) is odd then
        \((i, j) \leftarrow (2m - 1, 2m)\);
      else
        \( p \leftarrow 2^{(l/2) - 1} \);
        \( q \leftarrow \text{trunc}((m - 1)/(n/2p)) + 1 \);
        \( a \leftarrow m + (q - 1)(n/2p) \);
        \( b \leftarrow q(n/p) - (a - ((q - 1)(n/p) + 1)) \);
        \((i, j) \leftarrow (a, b)\);
    end;
    if \( A[i] > A[j] \) then swap\( (A[i], A[j]) \);
  end;
end;
end-OECE-Sort.

Fig. 4. The sorting algorithm OECE-Sort.

In Section 2, we have computed the length \( L \) of the Compare-Exchange scheme OECE\([n]\), which is

\[ L = 2 \log n - 1. \]

Thus, the number \( T(n) \) to Compare-Exchange steps required by the algorithm OECE-Sort to sort a sequence of length \( n \), is given by

\[ T(n) = L \cdot R = (2 \log n - 1) \cdot (\log n - 1) = 2 \log^2 n - 3 \log n + 1. \]

Moreover, it is easy to see that each step of the algorithm consists of \( n/2 \) comparison and \( n/2 \) exchange operations and, therefore, can be executed in constant time \( O(1) \) with \( n/2 \) processors on an EREW-PRAM. Consequently, we can formulate the following theorem:

**Theorem 4.1.** The algorithm OECE-Sort correctly sorts a sequence \( S = (x_1, x_2, \ldots, x_n) \) of \( n \) elements in time \( O(\log^2 n) \) using \( n/2 \) processors on a EREW-PRAM computational model, where \( n = 2^k \) for some integer \( k \geq 1 \).

5. The OECE sorting network

In this section we present the sorting network which directly implements the parallel algorithm OECE-Sort.

A processing element (PE) or comparator is a processor with the following properties:

(a) It consists of two inputs and two outputs, denoted \((A, B)\) and \((L, H)\) respectively.

(b) It compares the input data \( a \) and \( b \), and produces \( \min(a, b) \) in the output \( L \) and \( \max(a, b) \) in the output \( H \) (see Fig. 5).

Let \( C = (PE_1, PE_2, \ldots, PE_{n/2}) \) be a sequence of \( n/2 \) PEs. We assume that the inputs and outputs of the PEs of \( C \) are numbered with distinct
integers \( i, 1 \leq i \leq n \), called labels. Input \( A \) and output \( L \) of the processing element \( \text{PE}_m \) of the sequence \( C \), have the same label \( 2m - 1 \), and input \( B \) and output \( H \) of \( \text{PE}_m \) have the same label \( 2m, m = 1, 2, \ldots, n/2 \).

The comparisons performed by the parallel algorithm OECE-Sort are data-independent and hence can be hardwired in a sorting network. For this purpose, we invoke the structure of the Compare-Exchange scheme \( \text{OECE}[n] \) to say that the sorting network can be constructed by using the following components and topology.

(i) Components: \( L \) sequences \( C_i = (\text{PE}_1, \text{PE}_2, \ldots, \text{PE}_{n/2}) \), each of \( n/2 \) PEs, \( 1 \leq i \leq L, L = 2 \log n - 1 \).

(ii) Topology: Inputs \((2m - 1, 2m)\) of \( C_i \) are connected with outputs \((i, j)\) of \( C_{i-1} \) and outputs \((2m - 1, 2m)\) of \( C_i \) are connected with inputs \((i, j)\) of \( C_{i+1} \), iff \((i, j)\) is an edge of the compare-exchange graph \( \text{OECE}(n, l) \), where \( l = 2, 4, \ldots, L - 1 \) and \( m = 1, 2, \ldots, n/2 \).

We call the resulting architecture an OECE sorting network. The network OECE for sorting an arbitrary sequence of length 4 is illustrated in Fig. 5.

Let us now determine the time and processor complexity of the proposed network. We begin by assuming that a PE can read its input, perform a comparison, and produce its output all in one time unit. Now, let \( T(n) \) denote the time required by a sorting network to sort an arbitrary sequence of length \( n \) using \( P(n) \) processing elements. Then, it should be obvious that the time and processor complexity of the OECE sorting network, \( T(n) \) and \( P(n) \) respectively, are given by

\[
T(n) = 2 \log^2 n - 3 \log n + 1 = O(\log^2 n)
\]

and

\[
P(n) = L \cdot n/2 = (2 \log n - 1) \cdot n/2 = n \log n - n/2 = O(n \log n)
\]

where \( n = 2^k \) for some integer \( k \geq 1 \). The results of this section can be summarized by the following theorem:

**Theorem 5.1.** The OECE network for sorting an arbitrary sequence of length \( n = 2^k \), \( k \geq 1 \), requires \( O(\log^2 n) \) time and \( O(n \log n) \) PEs.

**References**


Stavros D. Nikolopoulos received the B.Sc. degree (with Honours) in Mathematics from the University of Ioannina, Greece, in 1982, the M.Sc degree in Computer Science from University of Dundee, Scotland, in 1985, and the Ph.D. degree in Computer Science from the University of Ioannina, in 1991. He was a research assistant in the Signal Processing Group, Saclant Undersea Research Centre, Italy, in 1990. The academic year 1991–92 he was an Associate Professor of Computer Science at the Hellenic Airforce Academy, Athens. In 1992 he joined the Department of Computer Science, University of Cyprus, Nicosia, where he is currently a Lecturer. He has published work in the areas of Parallel processing, Signal processing and Discrete event simulation. His current research interests focus on Parallel and Distributed Computation, Parallel Algorithms and Complexity, Graph Theory, Computational Geometry, and Combinatorial Mathematics.

Stylianos D. Danielopoulos (Nov. 28, 1932–Nov. 7, 1992) was born in Bucharest in 1932. He received the B.Sc. degree in Physics from the University of Athens, Greece, in 1954, and the Ph.D. degree in Theoretical Physics from North Carolina State University, USA, in 1970. From 1971 to 1978 he was with the Department of Computer Science at North Carolina State University as an Assistant Professor. From 1979 to 1992 he was a Faculty member at the University of Ioannina, Greece, where he last served as a Professor of Computer Science. Professor Danielopoulos passed away in 1992. His research interests were centred around Symbolic Mathematical Computation (Symbolic and Algebraic Manipulation), Algorithms and Complexity, Parallel Algorithms, Theory of Computation, Computational Linguistics, Contextual Grammars, Marcus, and Automatic Programming. He was a member of Sigma Xi, Sigma Pi Sigma and Pi Mu Epsilon honor societies.