The paper considers parallel (synchronous) array sorting algorithms obtained by parallelizing the corresponding sequential sorting algorithms. A classification of synchronous sorting algorithms, based on the sequential scheme, is proposed. A number of new efficient synchronous array sorting algorithms are developed.

Structured design of sequential in-memory sorting algorithms is considered in [1]. Our knowledge in this field, including a number of well-known and new sorting algorithms, is represented by the algebraic-grammatical apparatus, which provides a specification of the algorithms in terms of structured schemas.

In this paper, we consider the parallelization of the sequential sorting algorithms from [1]: they are reduced to synchronous algorithms by interpretation, reinterpretation, and transformation of the corresponding schemas. The main result is the extension of the classification proposed in [1] to synchronous in-memory sorting algorithms.

Section 1 introduces the basic concepts and notation and briefly reviews the classification of sequential sorting algorithms. Sections 2 and 3 construct a number of new synchronous in-memory sorting algorithms, including varieties of the logarithmic-time array sort. Bitonic array sorting algorithms are considered in Sec. 4. Section 5 describes a superfast abstract-register array sorting algorithms with running time bounded by a constant.

1. CLASSIFICATION OF SEQUENTIAL SORTING ALGORITHMS

Sequential sorting algorithms are described in [1] in terms of regular schemas (RS) — operator representations in Glushkov’s systems of algorithmic algebras (SAA). For the description of parallel algorithms, we will use the apparatus of modified SAA [2, 3]. The RS representation of sorting algorithms is based on the notion of a marked array.

Let \( D = D \cup R \), where \( D \) is the set of elements with the order relation \( \preceq \) and \( R = V \cup W \) is the set of separators consisting of the sets \( V \) (pointers) and \( W \) (markers). A marked array \( M \) is any sequence of the form \( PL u * \), where \( u \in D^+ \), \( D^+ \) is the set of nonempty sequences in the alphabet \( D \); \( PL, * \notin D \) are special markers marking the left (resp., the right) end of the marked array. By definition, \( PL < a_j < * \) for any element \( a_j \in D \).

Let \( M = PL z_1a_1z_2a_2...z_ia_iz_{i+1}...z_ja_jz_{j+1}...z_{n+1} * \) be a marked array, where \( z_1, z_2, ..., z_{n+1} \in R^+ \) are distinct sequences of pointers and markers (1 \( \leq i < j \leq n \)). The distance between the elements \( a_i \) and \( a_j \) of the array \( M \) is defined as \( j - i - 1 \), i.e., the number of elements of array \( M \) located between the elements \( a_i \) and \( a_j \). The distance between any two arbitrarily chosen separators in the array \( M \) is similarly defined as the number of array elements \( a_i \in D \) between the two separators.

On a marked array \( M \) we define logical conditions (predicates) and operators that constitute the basis of the sorting algorithm RS.

We introduce the pointers \( \lfloor, \rfloor, \llceil, \rrceil, \lfloor, \rfloor \) and the marker \( \# \).

Basis conditions:
\( d(v_1, v_2) = k \) is true if the separators \( v_1 \) and \( v_2 \) in \( M \) are at a distance \( k \) from each other;
\( d(z) \) is true (\( z \in R^+ \)) when \( z \) overlaps, i.e., when \( M \) contains the string \( z \);
\( d(PL) \) and \( d(*) \) are true when the pointers \( \lfloor \) and \( \rfloor \) reach the markers \( PL \) and \( * \), respectively;
let \( l \) and \( r(i, s) \) be the elements of \( M \) located immediately to the left and to the right of \( \lfloor \) (resp., of \( \rfloor \)).
$1 \leq_k s, l \geq_k s$ are true when the corresponding relationships are satisfied for the elements $l$ and $s$ of the given array located at a distance $k$, i.e., $1 \leq_k s = (d(\lceil\cdot\rceil, 1) = k) \land (l \leq s)$, $l \geq_k s = (d(\lceil\cdot\rceil, 1) = k) \land (l \geq s)$;

$l \leq r, l > r$ are true when these relationships are satisfied for the neighboring array elements $l$ and $r$ located immediately to the left and to the right of the pointer $\lceil$ (in particular, when the pointers $\lceil$ and $\rceil$ overlap, which corresponds to the case $k = 0$, we take $l = i$ and $r = s$);

$\pi(k)$ is true if for any pair of elements $a$ and $a'$ of array $M$ located at distance $k$ from each other we have $a \leq a'$ (for $k = 0$, we obtain the condition $\pi = \pi(0)$, which is true when the relation $a \leq a'$ is satisfied for any two neighboring elements $a$ and $a'$ in the array).

If the conditions $\pi_k$ and $\pi$ are true for the entire array or for some fragment of the array, then we say that the array or its fragment are sorted by the relation $\leq_k$ and $\leq$, respectively. In all other cases, these conditions take the value false.

Basis operators:
- PLC($z$) and DEL($z$) — place and delete the sequence $z$, $z \in \mathbb{R}^+$;
- $\overrightarrow{C}_k(v)$ and $\overleftarrow{C}_k(v)$ — simultaneous shift of a collection of pointers $v$ by $k$ elements, $k > 0$, to the left and resp. to the right (for $k = 1$, these operators are denoted by $\overrightarrow{C}(v)$ and $\overleftarrow{C}(v)$);
- $\overrightarrow{C}, \overleftarrow{C} (\overrightarrow{F}, \overleftarrow{F})$ — shift of the pointer $\lceil$ (resp., $\rceil$) by one array element to the left and to the right;
- $\overrightarrow{C}_k = \overrightarrow{C}_k (\lceil\cdot\rceil)$, $\overleftarrow{C}_k = \overleftarrow{C}_k (\lceil\cdot\rceil)$;
- TRANS$P(l, r)$ — transposition of the neighboring array elements $l$ and $r$ separated by the pointer $\lceil$; TRANS$P_k(l, s)$ — transposition of the elements $l$ and $s$ located at a distance $k$ from each other.

Sequential sorting algorithms are representable by RS in this basis [1].

As an example consider the RS $\overrightarrow{SHUTTLE}$ — the shuttle sort algorithm (direct insertions):

$$\overrightarrow{SHUTTLE} := \{ C \}_{d(\cdot) \in \mathbb{R}} \{ TRANSP (l, r) \overrightarrow{C} \}_{d(\cdot)} \{ \overrightarrow{C} \}).$$

The initial state of the array to be sorted is $M_0 = PL[l_1 a_1 \ldots a_n \ast]$. 

---

Fig. 1