It follows directly that lower estimates of the complexity of parallel algorithms for these problems are of the same order.

An analogous reduction technique may be used to establish the equivalence in time complexity of a number of other problems.

Starting from the means chosen to express parallel algorithms and the accepted computational model, the time spent on processor interaction during data transfer was not taken into account. However, it can be shown that for a number of models that allow us to take account of time needed to transfer data (for example PMP [1] or MSS [7] having a multidimensional lattice type of connection structure), the formulated equivalence assertions will also be valid.

LITERATURE CITED


EXTERNAL PARALLEL SORTING WITH MULTIPROCESSOR COMPUTERS

S. I. Comanescu

For external sorting the data structure must be such that relatively slow peripheral storage devices (tapes, disks, drums, etc.), are capable of satisfying the needs of the sorting algorithm. Thus, none of the traditional internal sorting methods can be directly applied to such files (they must be sorted stage by stage).

The process of external sorting can be divided into two essentially different phases: At the first phase the entire main computer memory is used for internal sorting of entries, forming out of them sorted segments of the greatest possible size, and outputting them to external memories (two or more, the choice of distribution depending on the specific method used for subsequent merging of sorted segments); at the second phase the obtained segments are merged (into still greater segments) until all entries form one ordered segment. It should be noted that this sorting strategy is not the only one possible. There are external sorting methods in which the process is not sharply divided into two distinct phases. Internal sorting and merging take place continuously during the entire process.

The methods discussed here employ the first strategy: internal parallel sorting with subsequent external parallel merging. They differ from each other in the last sorting phase (merging). The proposed methods provide considerably better estimates than the known similar methods [2].

All these methods are suitable for sequential files stored on magnetic tape. The use of other external memory devices (disks, drums, etc.) practically does not change the logic of the algorithm but only the method used to access to the external device for reading or writing a batch of data. It is important to know, in this connection, if the tapes can be
read in the reverse direction. If data can be read in one direction only, it will be necessary to rewind the tapes after each merging cycle. Such rewinding should be avoided with the aid of reverse reading, and if this is impossible internal computations should be combined with rewinding operations.

As an estimate of the measure of the efficiency of external sorting methods we will use the total sorting time as a function of \( N \) (the number of entries to be sorted) and \( P \) (the number of parallel processors participating in the sorting process). Since the efficiency of the external sorting process is completely determined by the input/output time (provided the sorting keys are not so long and complex as to require more internal computing time than input/output time), the function should express the total number of entries that passed through the main memory in the course of sorting. An entry will be assumed to have passed the main memory if it has been read from input tape, processed in the main memory, and written onto output tape.

By a "time unit" of algorithm operation is understood the set of all entries simultaneously passing through the main memory in one time quantum on \( P \) parallel processors.

Assume that a file of \( N \) entries is to be sorted on \( P \) parallel processors \( \Pi_1, \Pi_2, \ldots, \Pi_P \), and that the internal computer memory can hold only \( n \) entries (considering memory space needed for their internal sorting). Assume that the entire file is initially located on a single tape.

Algorithm A. This algorithm uses \( 4P \) tape transport mechanisms (including the tape holding the original unsorted file).

1. Set initially \( i = 1 \).
2. Enter next batch of \( n \) entries into main memory, \( \{a_{(i-1)n+1}, a_{(i-1)n+2}, \ldots, a_{in}\} \rightarrow \text{MM}. \)
3. Internal sort. Using \( P \) parallel processors sort out the contents of the MM.
4. Parallel output of the MM contents onto \( P \) parallel tapes. (Assume that the MM consists of \( P \) memory modules \( M_1, M_2, \ldots, M_P \) with a capacity of \( n/P \) entries each, the contents of \( M_i \) not exceeding the contents of \( M_{i+1} \) after executing step 3.) If \( i \) is odd, \( \Pi_j \) writes the contents of module \( M_j \) on tape \( T_j \); otherwise, the contents of \( M \) is written onto tape \( T_{P+j} \) for all \( j = 1, 2, \ldots, P \).
5. Increment \( i, i = i + 1 \). If \( i > N/n \), go to step 6, otherwise go to step 2.
6. Form \( P \) ordered subfiles of a size \( N/P \) each. Four tapes, two loaded with \( N/2n \) ordered segments of a length \( n/P \) and two empty, are assigned to processor \( \Pi_j, j = 1, 2, \ldots, P \).

Each processor executes the well-known [1] tape sorting algorithm using four tapes and a suitable main memory model which it controls. As a result we get \( P \) tapes each containing \( N/P \) ordered entries and a number of empty tapes.

7. Merge \( P \) ordered subfiles obtained at step 6. Execute \( \log_2 P \) sorting stages in accordance with the following scheme: At the first stage every two segments are sorted by merging into a single segment of \( 2N/P \) entries, at the \( i \)-th stage every two segments of \( 2^{i-1}N/P \) entries into a single segment of \( 2^iN/P \) entries, etc. At the last stage (\( i = \log_2 P \)) we have two tapes each having \( N/2 \log_2 P/2P \) ordered entries which by merging are sorted into a single file of \( N \) entries.

8. End algorithm.

If tapes can be read only in one direction, all tapes must be rewound to the start before executing step 7 and after each of its stages; otherwise, merging can take place in the course of each rewind step with the provision that if the segments were ordered by increasing key values at the \( i \)-th stage they will be ordered by decreasing keys at the \( (i + 1) \)-th stage. Thus, an obvious modification of step 7 can eliminate all rewind steps altogether. Only one output-file rewind step may prove necessary at the end of algorithms.

Let \( T_P(N) \) be the number of time units necessary for executing algorithm A.

**Assertion 1.** For all \( P \geq 2 \), we have \( T_P(N) < (N/P) \log_2 (N/n) + 2.5N \).

**Proof.** Step 2 of the algorithm is executed by recursion for \( i = 1, 2, \ldots, N/n \), \( n/2 \) entries passing through the main memory at each recursion (assuming that reading and writing times from and onto magnetic tape are the same and that entry processing in the main memory