Parallel construction of binary tree based on sorting*

Ya. E. Romm¹, D. A. Chabanyuk²*
¹,² Taganrog Chekhov Institute, Rostov State University of Economics (RINH) branch, Rostov-on-Don, Russian Federation

**E-mail: romm@list.ru, denchabanyuk@gmail.com

Introduction. Algorithms for the parallel binary tree construction are developed. The algorithms are based on sorting and described in a constructive form. For the \( N \) element set, the time complexity has \( T(R) = O(1) \) and \( T(R) = O(\log_2 N) \) estimates, where \( R = (N^2 - N) / 2 \) is the number of processors. The tree is built with the uniqueness property. The algorithms are invariant with respect to the input sequence type. The work objective is to develop and study ways of accelerating the process of organizing and transforming the tree-like data structures on the basis of the stable maximum parallel sorting algorithms for their application to the basic operations of information retrieval on databases.

Materials and Methods. A one-to-one relation between the input element set and the binary tree built for it is established using a stable address sorting. The sorting provides maximum concurrency, and, in an operator form, establishes a one-to-one mapping of input and output indices. On this basis, methods for the mutual transformation of the binary data structures are being developed.

Research Results. An efficient parallel algorithm for constructing a binary tree based on the address sorting with time complexity of \( T(N^2) = O(\log_2 N) \) is obtained. From the well-known analogues, the algorithm differs in structure and logarithmic estimation of time complexity, which makes it possible to achieve the acceleration of \( O(N^a) \), \( a \geq 1 \) order analogues. As an advanced version, an algorithm modification, which provides the maximum parallel construction of the binary tree based on a stable address sorting and a priori calculation of the stored subtree root indices is suggested. The algorithm differs in structure and estimation of \( T(1) = O(1) \) time complexity. A similar estimate is achieved in a sequential version of the mod-

* Work performed within the framework of the initiative R&D.

** E-mail: romm@list.ru, denchabanyuk@gmail.com

*** The research is done within the frame of independent R&D.

A unified algorithm, which allows obtaining the acceleration of known analogs of $O(N^a)$, $a > 1$ order.

**Discussion and Conclusions.** The results obtained are focused on the creation of effective methods for the dynamic database processing. The proposed methods and algorithms can form an algorithmic basis for an advanced deterministic search on the relational databases and information systems.

**Keywords:** data structures, data processing algorithms, binary tree, algorithms for parallel sorting.


**Introduction.** There is a tendency to the convergence of parallel information processing technologies and various processor architectures in the field of modern high-performance computing. Despite the variety of processor architectures and ways of presenting information, the idea of parallel processing is one of the most important tasks of computer science to increase the data-rate. To accelerate processing speed, the authors propose to use a stable address sorting algorithm with maximum parallelism.

**Method of parallel construction of a binary tree.** For $A = (a_0, a_1, ..., a_n)$ array, the comparison matrix is developed according to [1, 2]. $a_{ij}$ element of this matrix is defined as

$$a_{ij} = \begin{cases} +, & a_j > a_i \\ 0, & a_j = a_i \\ -, & a_j < a_i \end{cases},$$

where $i, j = 1, 2, ..., n$.

$a_i$ element in $C = (c_0, c_1, ..., c_{n-1})$ sorted array gets the number $k = \sum_{i=0}^{j} a_{ij}$, where $a_{ij} \geq 0$ at $i \leq j$, $a_{ij} > 0$ at $i > j$. All comparisons are mutually independent; the sorting is stable and as parallel as possible with the estimate of $T\left(\frac{N^2-N}{2}\right) = O(1)$ time complexity. On this basis, you can perform a parallel construction of a binary tree [3, 4].

Suppose we are given a set of $N$ elements, all elements of which are represented as a single-dimension array. On the set, $\leq$ ordering relation is assumed. It is required to convert the array into a binary tree. For this, the described array sorting is performed. $C$ medial array cell has $j_k = \left\lfloor \frac{N}{2} \right\rfloor$ index and is taken as the root of the tree [3]. All $C$ array components to the left of $C_{j_k}$ form a left subtree (left subarray). The components to the right of $C_{j_k}$ form a right subtree (right subarray). The left subarray is interpreted as a new array. It similarly contains $j_{sp_{\text{left}}} = \left\lfloor \frac{1}{2}\left(\left\lfloor \frac{N}{2} \right\rfloor - 1\right)\right\rfloor = \frac{j_k-1}{2}$ root index. Here, $C_{j_{sp_{\text{left}}}}$ is the left-nearest descendant of the root of $C_{j_k}$ tree. All components of the subarray to the left of $C_{j_{sp_{\text{left}}}}$ do not exceed $C_{j_{sp_{\text{left}}}}$; all components of the subarray on the right are not less than $C_{j_{sp_{\text{left}}}}$. Simultaneously, the root index of $j_{sp_{\text{right}}} = \left\lfloor \frac{N}{2} \right\rfloor + \left\lfloor \frac{1}{2}\left(\left\lfloor \frac{N}{2} \right\rfloor - 1\right)\right\rfloor = j_k + \frac{j_k-1}{2}$ right subarray is determined. At this, $C_{j_{sp_{\text{right}}}}$ is the nearest descendant of the root of $C_{j_k}$ tree. The process recursively resumes in each pair of the adjacent subarrays.
j_{cp. \text{mean}, 1/2^2} = \left\lfloor \frac{j_{cp. \text{mean}, 1/2^3} - 1}{2} \right\rfloor,

j_{cp. \text{mean}, 1/2^2} = j_{cp. \text{mean}, 1/2^2} + \left\lfloor \frac{j_{cp. \text{mean}, 1/2^3} - 1}{2} \right\rfloor,

j_{cp. \text{mean}, 1/2^2} = j_{cp. \text{mean}, 1/2^2} + \left\lfloor \frac{j_{cp. \text{mean}, 1/2^3} - j_{cp. \text{mean}, 1/2^3}}{2} \right\rfloor, \quad i = 1, 2, \ldots, \log_2 N.

As a result, all components of the lower level of the binary tree are formed in \(O(1)\) time. The process can continue until \(\log_2 N\) exhaustion of the levels of the binary tree.

The number of algorithm steps for constructing the binary tree in a parallel form is the sum of the sorting step and the step sequence when calculating the indices of the roots of subtrees. From here, \(T(R) = \log_2 N \cdot \tau + \tau = O(\log_2 N)\), where \(R\) is the number of processor elements, \(\tau\) is the time of binary comparison, and \(\tau\) is the time for calculating one root index. \(R\) number of processors is determined by the maximum \(N\) parallel sorting of input elements, and \(\tau\) – through the calculation of indices with doubling by the number of tree levels. When calculating the indices, this number will not exceed \(2^{\log_2 N - 1} = N/2\), therefore the number of processors involved in sorting is sufficient. As a result, \(R\) will be less than \(\frac{N^2 - N}{2}\) [3]. Finally, the time complexity of the parallel algorithm for constructing the binary tree will be

\[ T\left(\frac{N^2 - N}{2}\right) = O(\log_2 N). \]

Example [3]. The binary tree for an array of 15 elements \(X = (14, 9, 24, 7, 11, 20, 28, 3, 8, 10, 13, 17, 21, 25, 30)\) is constructed as follows.

The result of the sort is the array

\(C = (3, 7, 8, 9, 10, 11, 13, 15, 17, 20, 21, 24, 25, 28, 30)\).

The root of the binary tree is the medial element of \(C\) array: \(j_{cp} = \left\lfloor \frac{15}{2} \right\rfloor = 8, \ C_8 = 14\). The left subarray has

\(j_{cp. \text{mean}, 1/2} = \left\lfloor \frac{8 - 1}{2} \right\rfloor = 4\) root, \(C_4 = 9\) element is the root of the left subtree, which is the left-nearest descendant of \(C_{15}\) medial component. The right subarray has \(j_{cp. \text{mean}, 1/2} = 8 + \left\lfloor \frac{8 - 1}{2} \right\rfloor = 12\) root, \(C_{12} = 24\) element is the root of the right subtree and the right-nearest descendant of \(C_{15}\) root. Further, \(j_{cp. \text{mean}, 1/4} = \left\lfloor \frac{4 - 1}{2} \right\rfloor = 2, \ C_2 = 7\) element is the root of the subtree on the left and the left-nearest descendant of the root of \(C_{15}\) subtree. In the right subtree, the root has

\(j_{cp. \text{mean}, 1/4} = 4 + \left\lfloor \frac{4 - 1}{2} \right\rfloor = 6\) number, \(C_6 = 11\) element is the root of the right subtree and right-nearest child of \(C_{12}\) subtree.

Similarly, to the left of \(C_{15}\) root, \(j_{cp. \text{mean}, 1/4} = 12 - \left\lfloor \frac{12 - 8 - 1}{2} \right\rfloor = 10\) root is determined, \(C_{10} = 20\) element is the root of
the left subtree from it and the left-nearest descendant of \( C_{\text{sp. lев.1/2}} \) subtree root. For the subarray, adjacent to the right one discussed above, the root has \( j_{\text{sp. lев.1/4,2}} = 12 + \left\lceil \frac{12 - 8 - 1}{2} \right\rceil = 14 \) number, \( C_{14} = 28 \) element is \( C_{\text{sp. прав.1/2}} \) right-nearest descendant and the right subtree root. The lower level of the tree will be formed by the descendants remaining to the left and to the right of each of four identified roots (Fig. 1):

\[
\begin{array}{c}
0 \text{ level} \\
\begin{array}{c}
\overset{j_{\text{cp}} = 8}{C_8 = 14}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
1 \text{ level} \\
\begin{array}{c}
\overset{j_{\text{sp. lев.1/2}} = 4}{C_4 = 9} \\
\overset{j_{\text{sp. прав.1/2}} = 12}{C_6 = 11}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
2 \text{ level} \\
\begin{array}{c}
\overset{j_{\text{sp. lев.1/4,1}} = 2}{C_2 = 7} \\
\overset{j_{\text{sp. lев.1/4,2}} = 6}{C_3 = 8} \\
\overset{j_{\text{sp. прав.1/4,1}} = 10}{C_9 = 17} \\
\overset{j_{\text{sp. прав.1/4,2}} = 14}{C_{11} = 21}
\end{array}
\end{array}
\]

Fig. 1. Example of constructing binary tree based on sorting

There is Theorem 1 [3]. For a single-dimensional array of \( N \) components, a binary tree can be built in parallel using sorting with \( T\left(\frac{N^2}{2}\right) = O(\log N) \) time complexity.

The used sorting is stable; the binary tree is implied to be constructed with uniqueness. The indices of all medial components (all roots of subtrees) can be identified [3]. Considering this modification, all the indices from the above example for \( N \) subtree values can be calculated synchronously and mutually independently. This leads to a single estimate of the build time of the binary tree. For each specific \( N \), all the values of the tree node indices can be calculated a priori and stored in the computer memory. With their help, the sorted components can be synchronously and mutually independently addressed to all the tree nodes. Formulas for calculating the node indices depend only on the total number of \( N \) input elements and are in no way dependent on their mutual arrangement after the stable sorting. To simplify memory addressing, the computed indices can be ordered at each level and arranged in ascending levels. Then, the entire population of the ordered node indices is read from \( N \) key. It only remains to arrange the sorted tree elements by the read-in addresses. Based on the above, there is

Theorem 2. For a single-dimensional array of \( N \) components, a binary tree can be built in parallel using sorting and prior calculation of indices with \( T\left(\frac{N^2}{2}\right) = O(1) \) time complexity.

The following unified table contains the formal estimates of time complexity of sequential and parallel algorithms for constructing a binary tree versus the proposed algorithms.
Comparative estimates of time complexity of sequential and parallel algorithms for constructing binary tree versus proposed algorithms

<table>
<thead>
<tr>
<th>Binary tree algorithm</th>
<th>Algorithm time complexity</th>
<th>Acceleration when using unit time-complexity algorithm</th>
<th>Acceleration when using logarithmic time-complexity algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm of A. Lagana and V. Kumar (2004) [5]</td>
<td>$\tilde{T} = O\left(\frac{N \cdot \log N}{T}\right)$</td>
<td>$\frac{\tilde{T}}{T} = O\left(\frac{N \cdot \log N}{T}\right)$</td>
<td>$\frac{\tilde{T}}{T} = O\left(\frac{N \cdot \log N}{T}\right) = O(N \log N)$</td>
</tr>
<tr>
<td>Algorithm of P. Chalermsook (2015) [6]</td>
<td>$\tilde{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O\left(\frac{N^3}{\log N}\right) = O(N^2 \log N)$</td>
</tr>
<tr>
<td>Polynomial algorithm (2016) [7]</td>
<td>$\tilde{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O\left(\frac{N^3}{\log N}\right) = O(N^2 \log N)$</td>
</tr>
<tr>
<td>“Left child – right sibling” algorithm (2014) [8]</td>
<td>$\tilde{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O(N^3)$</td>
<td>$\frac{\tilde{T}}{T} = O\left(\frac{N^3}{\log N}\right) = O(N^2 \log N)$</td>
</tr>
<tr>
<td>Pattern-based algorithm (1991) [9]</td>
<td>$\tilde{T} = O\left(\left</td>
<td>D\right</td>
<td>\log D\right)$</td>
</tr>
</tbody>
</table>

In Table 1: $D$ is capacity of the template dictionary, $N$ is the number of input elements of the binary tree, $k$ is the dimension of the space in which sorting is performed.

The table shows that the proposed algorithm with a logarithmic estimate of time complexity abstractly improves estimates of the known algorithms. Minimum acceleration is achieved with respect to the algorithm from [5]: $\frac{\tilde{T}}{T} = O\left(\frac{N \cdot \log N}{T}\right)$, or $\frac{\tilde{T}}{T} = O(N)$; and maximum acceleration is achieved relative to the polynomial algorithm from [7]: $\frac{\tilde{T}}{T} = O\left(\frac{N^3}{\log N}\right)$ or $\frac{\tilde{T}}{T} = O(N^3)$. Regarding the proposed algorithm with a single estimate of time-complexity, the evaluation of the known algorithms also improves. In this case, minimum acceleration is achieved relative to the algorithm from [5]: $\frac{\tilde{T}}{T} = O(N)$, and maximum acceleration is achieved with respect to the polynomial algorithm from [7]: $\frac{\tilde{T}}{T} = O(N^3)$.

**Conclusion.** The developed algorithms differ from the known techniques [5–7, 10, 11] of constructing a binary tree in that they use maximum parallel sorting to calculate the indices of the nodes. In this case, either a logarithmic number of steps is consumed by building a tree, or additional time is not spent at all, if the values of the indices are a priori calculated for all $N$ values in some real boundaries and stored in the computer memory. The proposed parallel algorithm for constructing a binary tree can be used to organize efficient methods for dynamic processing of databases.
References


Received 21.01.2018
Submitted 24.01.2018
Scheduled in the issue 20.07.2018

Authors:

Romm, Yakov Ye.,
Head of the Information Technology Department,
Taganrog Chekhov Institute, Rostov State University of Economics (RINH) branch (48, Initsiativnaya St., Taganrog, RF), Dr.Sci. (Eng.), professor,
ORCID: http://orcid.org/0000-0002-7251-2844
romm@list.ru

Chabanyuk, Denis A.,
associate professor of the Theoretical, General Physics and Technologies Department, Taganrog Chekhov Institute,
Rostov State University of Economics (RINH) branch (48, Initsiativnaya St., Taganrog, RF),
ORCID http://orcid.org/0000-0003-2972-0944
denchabanyuk@gmail.com