Bull. Nov. Comp. Center, Comp. Science, 17 (2002), 75–88
© 2002 NCC Publisher

Associative parallel algorithm of checking spanning trees for optimality

A.S. Nepomniaschaya, T.V. Borets

In this paper, by means of an abstract model of the SIMD type with vertical data processing (the STAR-machine), we present a simple associative parallel algorithm for implementing the criterion of Chin and Houck to verify minimal spanning trees in undirected graphs. This algorithm is given as the corresponding STAR procedure CST, whose correctness is proved and time complexity is evaluated. We also provide an experiment of verifying two spanning trees for optimality in a given undirected graph.

1. Introduction

Associative (content-addressable) parallel systems of the SIMD type with vertical processing and simple single-bit processing elements are best suited to solve non-numerical problems. Such an architecture performs data parallelism at the basic level, provides massively parallel search by contents, and allows one using two-dimensional tables as basic data structures [1]. However, to solve tasks on these systems, it is necessary to construct new approaches and methods which take into account the advantages of this architecture.

To sum up the main results, we have constructed a natural straight forward implementation of a group of classical graph algorithms on a model of associative parallel systems (the STAR-machine) [2] using simple and natural data structures. For directed graphs, we have proposed associative versions of Warshall's algorithm for finding transitive closure [3], of Floyd's algorithm for finding the all-pairs shortest paths [3], of Dijkstra's algorithm [4] and the Bellman–Ford one [5] for finding the single-source shortest paths, and of Edmonds' algorithm for finding optimum branchings [6]. For undirected graphs, we have suggested associative versions of Kruskal's and the Prim–Dijkstra algorithms – one for finding the minimal spanning tree [7], and of Gabow's algorithm for finding the smallest spanning tree with a degree constraint [8]. The associative versions of algorithms have been given as the corresponding procedures represented on the STAR-machine, and their correctness has been proved.

Here, we suggest an associative version of the criterion of Chin and Houck [9] for verifying minimal spanning trees in undirected graphs. In [10], Tarjan proposed a special technique, path compression on balanced trees, to compute functions defined on paths in trees under various assumptions. This technique is applied to solve several graph problems. Among them there is the criterion of Chin and Houck. On sequential computers this algorithm takes $O(m\,\alpha(m,n))$ time, where n is the number of vertices, m is the number of edges in the given graph, and α is a functional inverse of Ackermann's function.

In this paper, for a given graph represented as a list of triples and for a given spanning tree, the criterion of Chin and Houck is implemented on the STAR-machine as procedure CST (checking a spanning tree) which returns true if and only if all non-tree edges of the graph satisfy the criterion. This procedure uses a new construction which defines for every vertex v_i of the given graph positions of edges belonging to the tree path from the source vertex to the vertex v_i. We prove correctness of the procedure CST and evaluate its complexity. We obtain that it takes $O(m \log n)$ time assuming that each elementary operation of the STAR-machine (its microstep) requires one unit of time.

2. Model of associative parallel machine

Let us recall our model which is based on a Staran-like associative parallel processor [11, 12]. We define it as an abstract STAR-machine of the SIMD type with bit-serial (or vertical) processing and simple single-bit processing elements (PEs). The model consists of the following components:

- a sequential control unit (CU), where programs and scalar constants are stored;
- an associative processing unit consisting of p single-bit PEs;
- a matrix memory for the associative processing unit.

The CU broadcasts an instruction to all the PEs in unit time. All active PEs execute it in parallel while inactive PEs do not perform it. Activation of a PE depends on the data. It should be noted that the time of performing any instruction does not depend on the number of processing elements [11].

Input binary data are loaded in the matrix memory in the form of two-dimensional tables in which each datum occupies an individual row and it is updated by a dedicated processing element. It is assumed that there are more PEs than data. The rows are numbered from top to bottom and the columns – from left to right. Both a row and a column can be easily accessed. Some tables may be loaded in the matrix memory.

The associative processing unit is represented as h vertical registers ($h \geq 4$), each consisting of p bits. The vertical registers can be regarded as a one-column array. The bit columns of the tabular data are stored in the registers which perform the necessary Boolean operations and record the search results.

The STAR-machine run is described by means of the language STAR [2] which is an extension of Pascal. Let us briefly consider the STAR constructions needed for the paper. To simulate data processing in the matrix memory, we use data types **word**, **slice**, and **table**. Constants for the types **slice** and **word** are represented as a sequence of symbols of $\{0, 1\}$ enclosed within single quotation marks. The types **slice** and **word** are used for the bit column access and the bit row access, respectively, and the type **table** is used for defining the tabular data. Assume that any variable of the type **slice** consists of p components which belong to $\{0, 1\}$. For simplicity, let us call *slice* any variable of the type **slice**.

Now, we present some elementary operations and predicates for slices.

Let X, Y be variables of the type **slice** and i be a variable of the type **integer**. We use the following operations:

SET(Y)	sets all the components of Y to $'1'$;
CLR(Y)	sets all the components of Y to $'0'$;
$Y(i)$	selects the i-th component of Y;
FND(Y)	returns the ordinal number i of the first (or the uppermost) component $'1'$ of Y, $i \geq 0$;
STEP(Y)	returns the same result as FND(Y) and then resets the first component $'1'$.

In the usual way, we introduce the predicates ZERO(Y) and SOME(Y) and the bitwise Boolean operations X *and* Y, X *or* Y, *not* Y, and X *xor* Y.

Let T be a variable of the type **table**. We employ the following two operations:

ROW(i, T)	returns the i-th row of the matrix T;
COL(i, T)	returns the i-th column of T.

Remark 1. All operations for the type **slice** can also be performed for the type **word**.

Remark 2. Note that the STAR statements [2] are defined in the same manner as for Pascal. We will use them later for presenting our procedures.

3. Preliminaries

At first, let us recall some notions being used in the paper.

Let $G = (V, E)$ be an *undirected weighted graph* with the set of vertices $V = \{1, 2, \dots, n\}$, the set of edges $E \subseteq V \times V$, and the function w that assigns a weight to every edge. We assume that $|V| = n$ and $|E| = m$.

In the STAR-machine matrix memory, an undirected weighted graph will be represented as association of the matrices *left*, *right*, and *weight*, where each edge $(u, v) \in E$ is matched with the triple $\langle u, v, w(u, v)\rangle$. Recall that vertices and weights are integers represented as binary strings.

A *path* from the vertex u to the vertex v in G is a sequence of vertices $u = v_1, v_2, \dots, v_k = v$, where $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \dots, k-1$ and $k > 0$.

A *spanning tree* $T = (V, E')$ of the given graph G is a connected acyclic subgraph of G, where $E' \subseteq E$.

A *minimal spanning tree* (MST) of G is a spanning tree, where the sum of weights of the corresponding edges is minimal.

Now, recall three basic procedures implemented on the STAR-machine which will be used later on. The first two procedures use a global slice X to select by ones positions of the rows which will be processed.

The procedure MATCH(T, X, v, Z) from [13] defines in parallel positions of those rows of the given matrix T which coincide with the given pattern v written in binary code. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW$(i, T) = v$ and $X(i) = '1'$.

The procedure GREAT(T, X, v, Z) from [13] defines in parallel positions of those rows of the given matrix T which are greater than the given pattern v written in binary code. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW$(i, T) > v$ and $X(i) = '1'$.

The procedure CLEAR(k, F) [4] sets zeros in all columns of the matrix F, where k is the number of columns in F.

4. Verifying minimal spanning trees on the RAM model

In [10], Tarjan suggests a special technique, path compression on balanced trees, being applied to compute functions defined on paths in trees. Here, we consider an application of this technique to verify a minimal spanning tree in undirected graphs.

Let T be a spanning tree of the given graph G. In [9], Chin and Houck presents the following criterion of verifying minimal spanning trees in undirected graphs.

A spanning tree T is *optimum* if and only if for each edge $(v_i, v_j) \in E - E'$ $w(v_i, v_j) \geq \max\{w(x, y)$: (x, y) is on the tree path joining v_i and $v_j\}$.

Let us shortly consider an implementation of this criterion on sequential computers given in [10]. It uses the following data structures:

- a graph G given as a list of m edges and their weights;
- an unrooted spanning tree T given as arrays *parent* and *children*;
- non-tree edges given as an array *pairs*.

This algorithm runs as follows.

First, it arbitrarily chooses a root r for T. Next, for each edge (v_i, v_j) from the array *pairs* by means of the procedure LCA, it computes the least common ancestor $u_i = \text{LCA}(v_i, v_j)$. Finally, it computes the maximal weight of edges on the tree paths from u_i to v_i and from u_i to v_j. Combining these results, we obtain the maximal weight of an edge along the tree path joining v_i and v_j for each non-tree edge (v_i, v_j).

The algorithm is realized as the procedure EVALUATE_PATHS which uses virtual trees. A *virtual tree* contains the same vertices as the real tree but different edges and labels [10]. Note that the root of a virtual tree saves the maximal weight of the path joining the vertices of the corresponding non-tree edge.

The procedure EVALUATE_PATHS initializes virtual trees and an array *bucket*. Initially for each vertex $v_i \neq r$, we create a virtual tree with the vertex v_i having $w(\textit{parent}(v_i), v_i)$ as its label. Then by means of the array *bucket* for each pair of vertices (v_i, v_j), we save its least common ancestor u_i, that is, $\textit{bucket}(u_i) = \{(v_i, v_j) \in \textit{pairs} : \text{LCA}(v_i, v_j) = u_i\}$. After that, the recursive procedure SEARCH(r) carries out a depth-first search to select the maximal weight on the tree path from the root of the current virtual tree to a vertex. During the search, each pair (v_i, v_j) is examined twice: once when the search is at v_i and once when the search is at v_j. When we follow parent pointers to the root r, virtual trees are merged by means of the procedure LINK.

The procedure LINK(v_i, v_j) adds the virtual tree with the root v_j to the virtual tree with the root v_i and assigns a new label for the vertex v_i as maximum of the labels for v_i and v_j.

5. Verifying minimal spanning trees on the STAR-machine

In this section, we present the implementation of the criterion of Chin and Houck for verifying minimal spanning trees on the STAR-machine. To this end, we first propose the procedure MatrixPath which defines for every vertex v_i *positions* of edges belonging to the tree path from the source vertex to v_i. Next, we present the procedure CST which returns **true** if and only if all non-tree edges satisfy the criterion.

On the STAR-machine, we represent a graph as association of the matrices *left*, *right*, and *weight*, and a spanning tree as a slice T in which positions of edges belonging to T are selected by ones.

5.1. Associative algorithm for finding tree paths. Here, we first present the main idea of the procedure MatrixPath. Assume we know positions of edges included into the tree path from the source vertex s to a vertex v_r. Then we construct a tree path for such a vertex v_k which is adjacent to v_r, the corresponding edge γ from the spanning tree T connects the vertices v_r and v_k, and the tree path from s to the vertex v_k has not yet been defined. The tree path from s to v_k is obtained by adding the position of the edge γ to the tree path from s to v_r.

Explain the meaning of the main variables being used.

The procedure MatrixPath uses a global slice Y for the matrices *left* and *right*, in which we select by ones positions of edges from the spanning tree T not included in any tree path; a global slice U for the matrix *code* in whose every i-th row there is the binary code of the vertex v_i; a variable *node*1 (respectively, *node*2) of the type **word** for saving the binary code of the vertex for which the tree path from s has been constructed (respectively, has not been constructed) and a variable k (respectively, j) of the type **integer** for storing its decimal code; a slice $N1$ (respectively, $N2$) for storing positions of the tree edges whose left (respectively, right) vertex has been included in the tree path from s.

Let us present the procedure MatrixPath.

```
proc MatrixPath(left,right,code: table; T: slice(left);
                n: integer; var R: table);
var U,U1: slice(code); X,Y,Z,N1,N2: slice(left);
    node1,node2: word; i,j,k: integer;
Begin CLR(N1); CLR(N2); SET(U);
  Y:=T; CLEAR(n,R);
  node1:=ROW(1,code);
/* The binary code of the source vertex is saved by means of node1. */
  MATCH(left,Y,node1,Z); N1:=N1 or Z;
  MATCH(right,Y,node1,Z); N2:=N2 or Z;
  X:=N1 or N2;
/* Positions of the tree edges which is incident with the source vertex
   are selected by ones in the slice X. */
  while SOME(X) do
    begin i:=STEP(X);
/* We determine the position of the tree edge which is incident
   with the vertex for which the tree path has been obtained. */
      if N1(i)='1' then
        begin node1:=ROW(i,left);
          node:=ROW(i,right); N1(i):='0';
        end
      else
```

```
14.          begin node1:=ROW(i,right);
15.            node2:=ROW(i,left); N2(i):='0';
     /* We save the binary code of the vertex for which the tree path
        has been obtained in node1, and the binary code of the vertex
        for which the tree path has not been obtained in node2. */
16.          end;
17.        Y(i):='0';
     /* The tree edge from the i-th position is indicated as updated one. */
18.        MATCH(code,U,node1,U1); k:=FND(U1);
19.        MATCH(code,U,node2,U1); j:=FND(U1);
20.        Z:=COL(k,R); Z(i):='1'; COL(j,R):=Z;
     /* The tree path to the vertex v_j is obtained from the tree path
        to the vertex v_k by adding the position of the edge (v_k, v_j). */
21.        MATCH(left,Y,node2,Z); N1:=N1 or Z;
22.        MATCH(right,Y,node2,Z); N2:=N2 or Z;
23.        X:=N1 or N2;
24.      end;
25. End;
```

Correctness of the procedure MatrixPath is established by means of the following

Theorem 1. *Let an undirected graph be given as association of matrices left and right. Let code be a matrix in whose i-th row there is the binary representation of the vertex v_i. Let a spanning tree T be given as a slice in which positions of edges belonging to it are selected by ones. Then the procedure* MatrixPath(*left*, *right*, *code*, T, n, R) *returns the matrix R in whose every j-th column positions of edges belonging to the tree path from the source vertex s to the vertex v_j are selected by ones.*

Proof. We prove this by induction on the number of edges r included in the spanning tree T.

Basis is verified for $r = 1$. One can immediately verify that after performing lines 1–3, the slice Y saves the copy of the spanning tree T, the matrix R consists of zeros, and the variable *node*1 saves the binary code of the source vertex s. As a result of performing lines 4–6, we indicate by one in the slice $N1$ (respectively, $N2$) position of the edge from T whose left (respectively, right) vertex coincides with the binary code of the vertex s. Therefore by means of the slice X, we save the position of the edge from T being incident with the vertex s. Since $X \neq \Theta$,* we perform the cycle from line 7.

*The notation $X \neq \Theta$ denotes that there is at least one component '1' in the slice X.

Here, on performing line 8, by means of the operation STEP(X), we define the position i of the edge selected by one in the slice X. On performing lines 9–16, we first define whether the position of the selected edge belongs to the slice $N1$. If it is true, the right vertex of the selected edge has not been updated. In this case, we save the binary code of the left vertex in *node*1 and the binary code of the right vertex in *node*2, and perform the statement $N1(i) := '0'$. Otherwise, we save the binary code of the right vertex in *node*1 and the binary code of the left vertex in *node*2, and fulfil the statement $N2(i) := '0'$.

On performing line 17, the position of the edge incident with s is selected by zero in the slice Y. Therefore, $Y = \Theta$. On fulfilling lines 18–19, the variable k saves the result of decoding *node*1 and the variable j saves the result of decoding *node*2. On performing line 20, there is a unique $'1'$ in the j-th column of the matrix R located in the i-th position. Hence, the position of the edge from T which connects the vertices s and v_j is selected by one in the j-th column of the matrix R.

Finally on performing lines 21–22, we obtain $N1 = N2 = \Theta$ because $Y = \Theta$ and $N1(i) = N2(i) = '0'$. Therefore in view of the statement $X := N1$ *or* $N2$ (line 23), we obtain $X = \Theta$. Hence, the cycle terminates.

Step of induction. Let the assertion be true for $1 \leq r \leq n-2$. We will prove it for spanning trees with $r+1$ edges. By inductive assumption for each l $(1 \leq l \leq r)$ in the l-th column of the matrix R, we select by ones positions of the tree edges which belong to the tree path joining the vertices s and v_l. Moreover, positions of updated edges in the slice Y are selected by zero. Since $r \leq n-2$ and $Y \neq \Theta$, on performing lines 21–23, we obtain $X \neq \Theta$. Therefore, we fulfil the current iteration starting from line 7. Using the same line of reasoning as in the basis, position of the last updated tree edge is selected by zero in the slice Y. Moreover, we obtain that the variable k saves the result of decoding *node*1 and the variable j saves the result of decoding *node*2. In addition, *node*2 saves the last vertex for which the tree path from s will be constructed. On performing line 20, we append the edge (v_k, v_j) to the tree path from s to the vertex v_k. Therefore in the j-th column of the matrix R, positions of edges belonging to the tree path from s to v_j are selected by ones.

This completes the proof. $\square$

Let us evaluate time complexity of the procedure MatrixPath. In view of basic procedures CLEAR and MATCH, execution of lines 1–6 takes $O(n) + O(\log n)$ time. The cycle **while** SOME(X) **do** is performed $n-1$ times because it updates each edge of the spanning tree. Since the basic procedure MATCH takes $O(\log n)$ time inside the cycle, we obtain that the procedure MatrixPath requires $O(n \log n)$ time.

5.2. Associative algorithm for verifying minimal spanning trees. Here, we first present the main idea of representing the criterion of Chin and Houck on the STAR-machine. For every non-tree edge (v_i, v_j) by means of the auxiliary procedure MatrixPath, we determine positions of edges included into the tree path joining the vertices v_i and v_j. Then by means of the basic procedure GREAT, we verify whether there is such an edge in this path whose weight is greater than the weight of the non-tree edge (v_i, v_j).

Let us explain the meaning of the main variables being used.

The procedure CST uses a global slice U for the matrix *code*; a slice Z – for saving positions of non-tree edges; a matrix R – for saving the result of the procedure MatrixPath; the variables *node*1, *node*2 and w of the type **word** – for selecting the left vertex, the right vertex, and the weight of the current non-tree edge.

```
proc CST(left,right,weight,code: table; T: slice(left);
         n: integer; var result: boolean);
var R: table; U,U1: slice(code); X,Y,Z: slice(left);
    node1,node2,w: word; i,j,k: integer;
Begin result:=true; SET(U); Z:= not T;
/* Positions of non-tree edges are selected by ones in the slice $Z$. */
  MatrixPath(left,right,code,T,n,R);
  while SOME(Z) do
    begin i:=STEP(Z);
/* We select the position of the uppermost unexamined non-tree
   edge $\gamma$ in the slice $Z$. */
      node1:=ROW(i,left);
      node2:=ROW(i,right);
      w:=ROW(i,weight);
/* By means of node1, node2, and $w$, we save the binary codes of the left
   vertex, the right one, and the weight of the selected non-tree edge $\gamma$. */
      MATCH(code,U,node1,U1);
      k:=FND(U1); X:=COL(k,R);
/* Positions of edges which belong to the tree path from $s$
   to the left vertex of $\gamma$ are saved in the slice $X$. */
      MATCH(code,U,node2,U1);
      j:=FND(U1); Y:=COL(j,R);
/* Positions of edges which belong to the tree path from $s$
   to the right vertex of $\gamma$ are saved in the slice $Y$. */
      X:=X xor Y;
    /* Positions of edges which belong to the tree path that join
       the vertices of $\gamma$ are selected by ones in the slice $X$. */
13.        GREAT(weight,X,w,Y);
    /* We select by ones in the slice $Y$ positions of the edges from the tree path,
       that join the vertices of $\gamma$, whose weights are larger than $w(\gamma)$. */
14.        if SOME($Y$) then
15.          begin result:=false; exit
16.          end;
17.      end;
18. End;
```

Theorem 2. *Let an undirected weighted graph be given as association of the matrices left, right, and weight. Let code be a matrix in whose i-th row there is the binary representation of the vertex v_i. Let a spanning tree T be given as a slice in which positions of edges belonging to it are selected by ones. Then the procedure* CST(*left*, *right*, *weight*, *code*, T, n, *result*) *returns the value* **true** *if and only if T is a minimal spanning tree.*

Proof. We prove this by induction on the number of edges r not included in the spanning tree T.

Basis is verified for $r = 1$. First after initializing, the variable *result* has the value **true**, the slice U consists of ones and positions of non-tree edges are selected by ones in the slice Z (line 1). After performing the auxiliary procedure MatrixPath (line 2), we construct the matrix R, in whose every j-th column positions of edges belonging to the tree path from the source vertex s to the vertex v_j are selected by ones. Since $Z \neq \Theta$, we perform the cycle from line 3.

Here, on fulfilling line 4, we determine the position i of the unique edge selected by one in Z. Then on performing lines 5–7 by means of *node*1 (respectively, *node*2), we save the binary code of the left (respectively, right) vertex of this edge, and by means of w its weight. As a result of performing lines 8–9, the variable k saves the result of decoding *node*1, and positions of edges which belong to the tree path from s to the vertex v_k are selected by ones in the slice X. Similarly, on performing lines 10–11, the variable j saves the result of decoding *node*2 and positions of the edges, which belong to the tree path from s to v_j, are selected by ones in the slice Y.

On performing the statement $X := X \; xor \; Y$ (line 12), we select by ones positions of edges from the tree path joining the vertices v_k and v_j.

Finally, on fulfilling the basic procedure GREAT(*weight*, X, w, Y) positions of the tree edges joining the vertices v_k and v_j whose weights are greater than w are selected by ones in the slice Y. If there is such an edge (that is, $Y \neq \Theta$), the procedure CST returns **false**, otherwise it returns

true (lines 1, 16). Since $Z = \Theta$, the procedure terminates.

Step of induction. Let the assertion be true for $r \leq n - 2$. We will prove it for $r + 1$. By inductive assumption after updating the first r non-tree edges selected by ones in the slice Z, the procedure CST returns **false** if and only if for a non-tree edge γ there is such an edge σ in the tree path joining the vertices of γ, for which $w(\sigma) > w(\gamma)$. Without loss of generality it is sufficient to consider the case when the criterion of Chin and Houck is fulfilled for the first r non-tree edges. Then after updating these edges position of the last non-tree edge is selected by one in the slice Z. Since $Z \neq \Theta$, we perform the current iteration starting from line 4. In the same manner as shown in the basis, we verify the criterion for the tree path which joins the vertices of the last non-tree edge. Since now $Z = \Theta$, the procedure CST terminates and returns either the result **true** if the criterion is fulfilled for the last $(r + 1)$-th non-tree edge or the result **false** otherwise.

This completes the proof. □

Let us evaluate time complexity of the procedure CST. In view of the procedure MatrixPath, execution of lines 1–2 takes $O(n \log n)$ time. The cycle **while** SOME(Z) **do** is performed for all edges not included in the spanning tree, that is, $m - n + 1$ times. Since the basic procedure MATCH takes $O(\log n)$ time inside the cycle, we obtain that the procedure CST takes $O(m \log n)$ time on the STAR-machine having no more than m processing elements.

6. Experiments

In this section, we provide two examples to illustrate the implementation of the procedure CST(*left*, *right*, *weight*, *code*, T, n, *result*) of verifying spanning trees T_1 and T_2 for optimality in an undirected graph.

The original graph G is given in Figure 1 while the spanning trees are given in Figures 2 and 3. In the procedure CST, the graph G is represented as association of matrices *left*, *right*, and *weight*, the spanning tree is given as a slice T_1 or T_2, and $n = 6$.

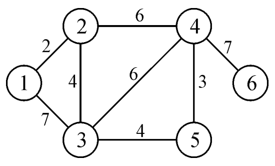

Figure 1. Graph G

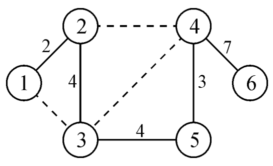

Figure 2. Spanning tree T_1

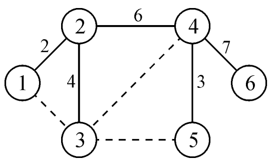

Figure 3. Spanning tree T_2

	Table			Slice	
	left	right	weight	T_1	T_2
1	001	010	010	1	1
2	001	011	111	0	0
3	010	011	100	1	1
4	010	100	110	0	1
5	011	100	110	0	0
6	100	101	011	1	1
7	011	101	100	1	0
8	100	110	111	1	1

	Code
1	001
2	010
3	011
4	100
5	101
6	110

In the *first test*, we consider the spanning tree T_1. After performing the procedure MatrixPath, we obtain the matrix R_1 as shown below. Positions of non-tree edges are selected by ones in the slice Z. Since the slice Z does not consist of zeros, the procedure CST continues its run, until all ones will be deleted from Z.

	Table R_1						Slice			
	1	2	3	4	5	6	Z	X_1	X_2	X_3
1	0	1	1	1	1	1	0	1	0	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	1	1	1	1	0	1	1	0
4	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	1	0	0	0
6	0	0	0	1	0	1	0	0	1	1
7	0	0	0	1	1	1	0	0	1	1
8	0	0	0	0	0	0	0	0	0	0

The first non-zero element in the slice Z corresponds to the non-tree edge (1,3) having weight 7 and located in the second row of the graph representation. Positions of edges which belong to the path joining vertices 1 and 3 are saved in the slice X_1. The weights of edges belonging to this path are not larger than the weight of edge (1,3). Since $Z \neq \Theta$, the procedure continues its run.

The position of the next non-zero element in Z corresponds to the fourth edge (2,4) with weight 6. Positions of tree edges belonging to the path joining vertices 2 and 4 are saved in the slice X_2. Again the weights of edges belonging to this path are not larger than the weight of the edge (2,4).

The last non-tree edge is the fifth edge (3,4). Positions of tree edges belonging to the path joining vertices 2 and 4 are saved in the slice X_3. The weight of the edge (3,4) is larger than the weights of edges belonging to this path. After updating the last non-tree edge, the slice Z consists of zeros. Therefore, the procedure CST stops with the result **true**. Hence, the spanning tree T_1 is minimal.

In the *second test*, we consider the spanning tree T_2. After performing the procedure MatrixPath, we obtain the matrix R_2 as shown below. Positions of non-tree edges are selected by ones in the slice Z.

	Table R_2						Slice			
	1	2	3	4	5	6	Z	X_1	X_2	X_3
1	0	1	1	1	1	1	0	1	0	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	1	0	0	0	0	1	1	1
4	0	0	0	1	1	1	0	0	1	1
5	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	1	0	0	0	0	1
7	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	1	0	0	0	0

The first non-tree edge (1,3) was considered in the first test.

The next non-tree edge is the fifth edge (3,4) with weight 6. Positions of tree edges belonging to the path between vertices 3 and 4 are saved in the slice X_2. Again the weights of edges belonging to this path are not larger than the weight of edge (3,4).

The last non-tree edge is edge (3,5) with weight 4. Positions of tree edges belonging to the path between vertices 3 and 5 are saved in the slice X_3. However, the weight of the fourth edge (2,4) is greater than the weight of edge (3,5). Therefore the procedure CST stops with the result **false**.

7. Conclusions

We have presented a natural matrix implementation of the criterion of Chin and Houck for verifying a spanning tree to be a minimal one by means of the STAR-machine which is a model of associative parallel systems with vertical processing. To this end, for a graph given as a list of triples and for a spanning tree T given as a slice, we have suggested a simple associative parallel algorithm which constructs the Boolean matrix in whose each i-th column positions of edges included in the tree path from the source vertex to the vertex v_i are selected by ones. We have also presented implementation of the criterion of Chin and Houck using Tarjan's technique for path compression on balanced trees. Our result illustrates that associative parallel systems with vertical processing allows one to use both a simple and natural data structure and a simple algorithm for implementing criterion of Chin and Houck.

We are planning to employ our construction for designing new associative parallel algorithms which utilize tree paths. In particular, it will be used to find a fundamental set of cycles in undirected graphs relatively to a given spanning tree.

References

[1] Potter J.L. Associative Computing: a Programming Paradigm for Massively Parallel Computers. – New York and London: Kent State University, Plenum Press, 1992.

[2] Nepomniaschaya A.S. Language STAR for associative and parallel computation with vertical data processing // Proc. of the Intern. Conf. "Parallel Computing Technologies". Singapure: World Scientific. – 1991. – P. 258–265.

[3] Nepomniaschaya A.S. Solution of path problems using associative parallel processors // Proceedings of the International Conference on Parallel and Distributed Systems, ICPADS'97, December 10–13, 1997. – Seoul, Korea: IEEE Computer Society Press, 1997. – P. 610–617.

[4] Nepomniaschaya A.S., Dvoskina M.A. A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. – Amsterdam: IOS Press, 2000. – Vol. 43. – P. 227–243.

[5] Nepomniaschaya A.S. An associative version of the Bellman–Ford algorithm for finding the shortest paths in directed graphs // Proceedings of the 6-th Intern. Conf. PaCT-2001. Lect. Notes in Comp. Sci. – Berlin: Springer-Verlag, 2001. – Vol. 2127. – P. 285–292.

[6] Nepomniaschaya A.S. Efficient implementation of Edmonds' algorithm for finding optimum branchings on associative parallel processors // Proc. of the Eighth Intern. Conf. on Parallel and Distributed Systems (ICPADS'01). – KyongJu City, Korea: IEEE Computer society Press, 2001. – P. 3–8.

[7] Nepomniaschaya A.S. Comparison of performing the Prim–Dijkstra algorithm and the Kruskal algorithm by means of associative parallel processors // Cybernetics and System Analysis. – 2000. – № 2. – P. 19–27 (in Russian); English translation by Plenum Press.

[8] Nepomniaschaya A.S. Representation of the Gabow algorithm for finding smallest spanning trees with a degree constraint on associative parallel processors // Euro-Par'96 Parallel Processing. Second Intern. Euro-Par Conf. Lyon, France. Proceedings, Lect. Notes in Comp. Sci. – Berlin: Springer-Verlag, 1996. – Vol. 1123. – P. 813–817.

[9] Chin F., Houck D. Algorithms for updating minimal spanning trees // J. of Computer and System Sciences. – 1978. – Vol. 16. – P. 333–344.

[10] Tarjan R.E. Applications of path compression on balanced trees // J. of the ACM. – 1979. – Vol. 26, № 4. – P. 690–715.

[11] Foster C.C. Content Addressable Parallel Processors. – New York: Van Nostrand Reinhold Company, 1976.

[12] Mirenkov N. The Siberian approach for an open-system high-performance computing architecture // Computing and Control Engineering J. – 1992. – Vol. 3, № 3. – P. 137–142.

[13] Nepomniaschaya A.S. Investigation of associative search algorithms in vertical processing systems // Proc. of the Intern. Conf. "Parallel Computing Technologies". – Obninsk, Russia, 1993. – Vol. 3. – P. 631–642.