Bull. Nov. Comp. Center, Comp. Science, 36 (2014), 65–78
© 2014 NCC Publisher

Constructions used in associative parallel algorithms for undirected graphs. Part 2

A.S. Nepomniaschaya

Abstract. This paper selects constructions used in a group of algorithms for undirected graphs represented as a list of edges and their weights on a model of associative (content addressable) parallel systems with vertical processing (the STAR-machine). To this end, the paper first analyzes the implementation of the Prim–Dijkstra algorithm on the STAR-machine for finding an MST along with the method of finding the tree paths with respect to a given spanning tree. Then the paper considers the implementation on the STAR-machine of the dynamic graph algorithms for updating an MST. This group includes associative parallel algorithms for the dynamic edge update of an MST and for the dynamic reconstruction of an MST after deleting or after inserting a vertex along with its incident edges.

1. Introduction

Associative processing is a completely different way of storing, manipulating, and retreiving data as compared to conventional computation techniques. Of special interest is a class of associative (content addressable) parallel processors of the SIMD type with bit-serial (vertical) processing and simple single-bit processing elements [2]. The vertical processing systems are best suited to solving non-numerical problems. In [3], we propose an abstract model of the SIMD type (the STAR-machine) that simulates the run of such systems on micro level. Associative parallel algorithms are represented as corresponding procedures for the STAR-machine. In [4], we present the basic associative parallel algorithms that are used to design different associative algorithms for different applications. Let us enumerate some results of solving graph problems. In [3, 5], we construct a natural straightforward implementation of classical graph algorithms by Dijkstra and Bellman–Ford on the STAR-machine for finding the single-source shortest paths. In [7], we propose two associative parallel algorithms for the dynamic edge update of minimum spanning trees of undirected graphs. In [8–9], we propose associative parallel algorithms for the dynamic reconstruction of an MST after deleting and after inserting a new vertex along with its incident edges. In [10], we present the efficient implementation on the STAR-machine of the Italiano algorithms for the dynamic update of the transitive closure of directed graphs. In [11–12], we propose the efficient implementation on the STAR-machine of the Ramalingam algorithms for the dynamic update of the shortest paths subgraph of a directed graph with a sink. In [13], we select a group of constructions used to represent on the STAR-machine the classical graph algorithms by Prim–Dijkstra, Kruskal, and Gabow, and the method of finding tree paths with respect to a given spanning tree.

In this paper, we select a group of constructions from the dynamic algorithms on undirected graphs that use a fast method for finding a tree path between any pair of vertices. This group includes an associative parallel algorithm for finding an MST along with the matrix of tree paths, associative parallel algorithms for the dynamic edge update of an MST, and the dynamic reconstruction of an MST after inserting and after deleting a vertex along with its incident edges. To represent these algorithms on the STAR-machine, an undirected graph is given as a list of edges and their weights. The selected constructions can be used to design new associative parallel algorithms in graphs.

2. An associative parallel machine model

In this section, we first recall the main operations of the STAR-machine. The description of the model is given, for example, in [4].

Let us present some elementary operations and a predicate for variables of the type **slice**.[1]

We use the following operations:

SET(Y)	simultaneously sets all components of Y to $'1'$;
CLR(Y)	simultaneously sets all components of Y to $'0'$;
$Y(i)$	selects a value of the ith component of Y;
FND(Y)	returns the ordinal number i of the first (the uppermost) bit $'1'$ of Y, $i \geq 0$;
STEP(Y)	returns the same result as FND(Y) and then resets the first found $'1'$ to $'0'$;
CONVERT(Y)	returns a row, whose every ith bit coincides with $Y(i)$. It is applied when a row of one matrix is used as a slice for another matrix.

To implement the data parallelism, in the usual way we introduce the bitwise Boolean operations: X *and* Y, X *or* Y, *not* Y, X *xor* Y. We also use the predicate SOME(Y) that results in **true** if there is at least a single bit $'1'$ in the slice Y. For simplicity, the notation $Y \neq \emptyset$ means that the predicate SOME(Y) results in **true**.

Note that the predicate SOME(Y) and all operations for the type **slice** are also performed for the type **word**.

[1]For simplicity let us call *slice* any variable of the type **slice**.

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

ROW(i, T) returns the ith row of the matrix T;

COL(i, T) returns its ith column.

Note that the STAR statements are defined in the same manner as for Pascal. They are used for presenting the procedures.

Now, let us recall a few basic procedures [4] implemented on the STAR-machine. They use a given slice X to indicate with $'1'$ the row positions employed in the corresponding procedure. In [4], we have shown that the basic procedures take $O(r)$ time each, where r is the number of bit columns in the corresponding matrix.

The procedure MATCH(T, X, w, Z) determines the positions of the rows of the matrix T that coincide with the given pattern w. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW$(i, T) = w$ and $X(i) = '1'$.

The procedure MIN(T, X, Z) finds the positions of rows in the given matrix T, where a minimum entry is located. These positions are marked with $'1'$ in the result slice Z.

The procedure MAX(T, X, Z) is defined by analogy with MIN(T, X, Z).

3. Preliminaries

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

In the STAR-machine matrix memory, a graph will be represented as association of the matrices `Left`, `Right`, and `Weight`, where each edge (u, v) is matched with the triple $\langle u, v, wt(u, v)\rangle$.

A *path* from v_1 to v_k in G is a sequence of the vertices $v_1, \ldots, v_k$, where $(v_i, v_{i+1}) \in E$ for $1 \le i < k$. If $v_1 = v_k$, then the path is said to be a *cycle* or a *circuit.*

A *spanning tree* $T = (V, E')$ of the given graph G is a connected acyclic subgraph of G, where $E' \subseteq E$. Each edge $e \in E \setminus E'$ is called a *chord* of G with respect to the spanning tree. Adding a chord to a spanning tree creates precisely one circuit.

Let $\delta_T(v)$ be the number of edges of T incident on v.

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

A connected component is a maximal connected subgraph.

4. Updating tree paths

In this section, we first enumerate constructions that are used to obtain an MST along with the matrix of tree paths. Then we select constructions that are used to update the matrix of tree paths.

In [6], we propose an associative version of the Prim–Dijkstra algorithm for finding an MST starting with a given vertex v. The corresponding procedure `MSTPD` returns a slice `Tree`, where *positions* of edges belonging to the MST are marked with bits $'1'$. In [13], we select constructions used to represent the procedure `MSTPD` on the STAR-machine. Moreover, we extract constructions that are used to describe the method of finding tree paths with respect to a given spanning tree.

The dynamic graph algorithms for updating an MST require a fast method for finding a tree path between any pair of vertices. In [7], by means of *minor* changes in the procedure `MSTPD`, we build an MST along with a matrix `TPaths`, whose every ith column saves *positions* of edges belonging to the tree path from the root v_1 to the vertex v_i. As input parameters, the corresponding procedure `MSTPaths` uses the matrices `Left`, `Right`, `Weight`, and `Code`, and a global slice S. It returns the slice `Tree` and the matrix of tree paths `TPaths`. The procedure `MSTPaths` runs as follows. Initially, it sets zeros in the first column of the matrix `TPaths` and saves the root v_1 being the first vertex of the fragment T_S. By analogy with `MSTPD`, at every iteration, it defines both the *position* of the current edge γ and the corresponding *new* vertex v_k being included in T_S. Moreover, it defines the endpoint v_l of γ included in T_S *before* this iteration. The tree path from v_1 to v_k is obtained by adding the *position* of γ to the tree path from v_1 to v_l defined before. This path is written in the kth column of the matrix `TPaths`.

By analogy with the procedure `MSTPD`, it uses Construction 1 to determine *positions* of edges forming a cycle. By analogy with the procedure `MatrixPaths`, it uses Construction 3 to determine binary codes of the endpoints of the current edge included into the building path at the current iteration. Then by means of Construction 5, it determines decimal numbers of the endpoints of the current edge included into the fragment of T_S. Finally, by means of Construction 4, it builds a path from v_1 to the new vertex included into T_S.

In [7], we have shown that the procedure `MSTPaths` takes the same time $O(n \log n)$ as the procedure `MSTPD` for finding an MST in undirected graphs. Without loss of generality, we will assume that initially an MST is always given along with the matrix `TPaths`.

Let a new MST be obtained from the underlying one by deleting an edge, say, γ, and inserting an edge, say, δ. Let $Y1$ be a connected component of G obtained after deleting γ.

Construction 11 (*Finding the row where the edge* (v_i, v_j) *is located in the graph representation*). We first define binary codes `node1` and `node2` of the vertices v_i and v_j, respectively. Then with a slice, we determine positions of edges in the graph representation, whose left vertex coincides with `node1`. Further among these edges, we select the position l of the edge whose right vertex coincides with `node2`.

To perform this construction, we use the basic procedure MATCH and the operation FND.

Construction 12 (*Finding a connected component Y1 obtained after deleting an edge from the lth row*). We first save the lth row of the matrix `TPaths`. Then we write the convertation of this row into the slice $Y1$.

In [7], we propose an associative parallel algorithm for updating tree paths. It determines *new* tree paths for all vertices from $Y1$. Let v_{del} and v_{ins} be the endpoints of the corresponding edges γ and δ that belong to $Y1$. Let P be a slice that saves *positions* of tree edges joining v_{ins} and v_{del}.

Construction 13 (*Finding vertices* v_{del} *and* v_{ins} *that belong to* $Y1$). Let a connected component $Y1$ be given. Let an edge γ from the lth row of the graph representation be deleted from the MST, and the edge δ from the kth row be inserted into the MST. Then we determine decimal numbers of the endpoints of the edge γ (respectively, δ). The vertex v_{del} (respectively, v_{ins}) is the endpoint of γ (respectively, δ) that belongs to $Y1$.

To perform this construction, we first define decimal numbers of the endpoints of the edges γ and δ using Construction 5. Further we check which endpoint of γ (respectively, δ) corresponds to bit $'1'$ in the slice $Y1$.

To select the next two constructions, we recall an example from [7].

Let a new MST be obtained from the underlying one after deleting the edge (v_4, v_8) (Figure 1) and inserting a new edge (v_7, v_{14}) (Figure 2). Here, the connected component $Y1$ consists of the vertices $v_8, \ldots, v_{18}$; $\mathrm{del} = 8$ and $\mathrm{ins} = 14$.

The algorithm starts with the vertex v_{14}. Then the new tree paths are recomputed for the vertices v_{15}, v_{16}, v_{17}, and v_{18} from the subtree rooted at v_{14}. Further, a new tree path is first defined for v_{13} and then for v_8. Finally, new tree paths are recomputed for the vertices v_9, v_{10}, v_{11}, and v_{12} from the subtree rooted at v_8.

Let us agree, for convenience, that a tree path from v_1 to any vertex v_r is denoted by p_r *before* updating the MST and by p'_r *after* updating the MST.

The associative parallel algorithm starts with the vertex v_{ins}. Note that p'_{ins} is known.

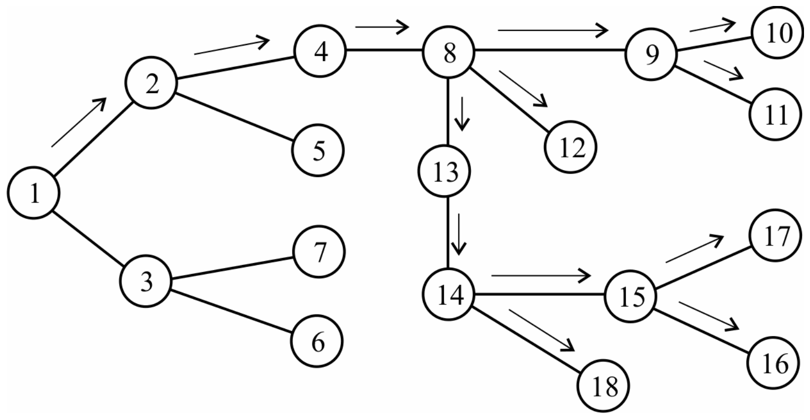

Figure 1. The MST before deleting the edge (v_4, v_8)

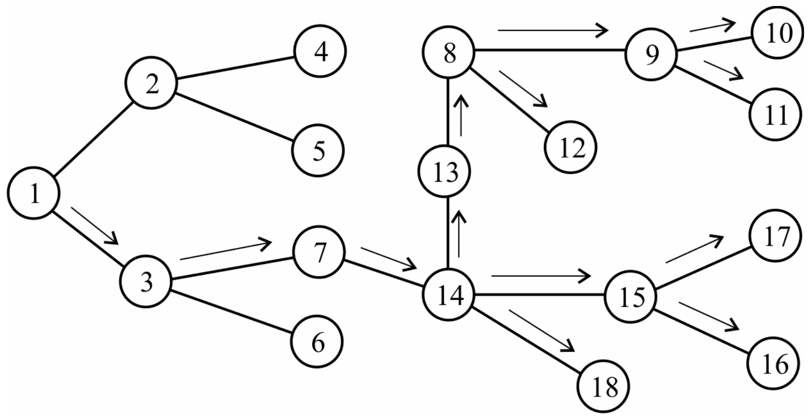

Figure 2. The MST after inserting the edge (v_7, v_{14})

Construction 14 (*Finding positions of edges belonging to the tree path from v_{ins} to v_{del}*). Knowing the matrix `TPaths`, we first save the paths p_{ins} and p_{del}. The tree path P from v_{ins} to v_{del} is obtained by means of the operation *xor* between the corresponding slices.

Construction 15 (*Finding a new tree path to a vertex from the subtree rooted at the path P*). We determine vertices not belonging to P that form a subtree of the MST with the root v_r if any. For every $v_j \neq v_r$ from this subtree, we compute p'_j as follows: $p'_j := (p_j \textit{ and } (\textit{not}\, p_r))\textit{ or } p'_r$.

To perform this construction, we first determine the *position* i of an edge from P incident on v_r. By means of Construction 12, we find all vertices from the subtree rooted at v_r. Among them we exclude the vertices being updated before.

On the STAR-machine, the associative parallel algorithm for updating tree paths is implemented as procedure `TreePaths` which uses the following input parameters: matrices `Left`, `Right`, and `Code`, vertices v_{ins} and v_{del}, the number of graph vertices n and the position l of the deleted edge. This returns the matrix `TPaths` for the new MST and the slices W and P. In [7], we have shown that the procedure `TreePaths` takes $O(h \log n)$ time, where h is the number of vertices in the connected component $Y1$.

5. Dynamic edge update of a minimum spanning tree

In [7], we present associative parallel algorithms for the dynamic edge deletion from the MST and for the dynamic edge insertion into the MST. In this section, we will select constructions that are used to implement these algorithms on the STAR-machine.

Let v_i and v_j be endpoints of the edge that is deleted from a minimum spanning tree given as a slice `Tree`. The associative parallel algorithm first determines the *position* of the edge (v_i, v_j) in the graph representation and deletes it from further consideration. Then, it defines the connected component $Y1$ that consists of vertices not reachable from the source vertex v_1 after deleting this edge. Further, it determines *positions* of edges joining two connected components. Among these edges, a minimum weight edge is selected and its position is saved in the slice `Tree`. Finally, tree paths for vertices from $Y1$ are recomputed as shown in the previous section.

Construction 16 (*Finding positions of edges joining two connected components*). First, by means of slice $N1$ (respectively, $N2$), we accumulate *positions* of edges not included into the MST, whose left (respectively, right) endpoint belongs to $Y1$. Then using operation $N1$ *or* $N2$, we determine *positions* of edges having at least one endpoint from $Y1$. After that, by means of operation $N1$ *and* $N2$, we define *positions* of edges whose both endpoints belong to $Y1$. Knowing the disjunction of slices $N1$ and $N2$ and their conjunction, we easily determine a slice that saves *positions* of edges having a *single* endpoint from the connected component $Y1$. Obviously, this slice saves positions of edges joining two connected components.

To perform this construction, we use the basic procedure MATCH.

So, the associative parallel algorithm for dynamic edge deletion from the MST uses Construction 11 to determine the position of the edge (v_i, v_j) in the graph representation, Construction 12 to find the connected component $Y1$, Construction 16 to determine positions of edges joining two connected components, Constructions 13–15 to recompute tree paths for vertices from the connected component $Y1$.

On the STAR-machine, this algorithm is implemented as procedure `DelEdge` that uses the following input parameters: matrices `Left`, `Right`, `Weight`, and `Code`, a slice Y for the matrix `Code`, and endpoints v_i and v_j of the deleted edge. This returns the current slice S for the graph G, the current MST `Tree`, and the current matrix of tree paths `TPaths`.

Now we will select constructions that are used in the associative parallel algorithm for the dynamic update of the current MST after insertion of an edge in the underlying graph G [7]. As is shown in [1] if a new edge is added to G, then the new MST is obtained by adding the new edge to the current MST and deleting the largest edge in the cycle created.

Let v_i and v_j be endpoints of an edge being inserted in G. The associative parallel algorithm runs as follows. It first determines the *position* k of an edge being added to G. Then, it defines *positions* of tree edges joining endpoints of this edge. Further, it determines *position* l of a maximum weight edge in the cycle created. If $k \neq l$, the algorithm first deletes the position of a maximum weight edge from the slice `Tree` and adds to it the position k of the new edge. Further, it defines the connected component $Y1$ whose vertices are not reachable from v_1 after deleting an edge from `Tree`. Finally, it recomputes tree paths for vertices from $Y1$.

The associative parallel algorithm for the dynamic update of the current MST after insertion of an edge to G uses Construction 11 to determine the position of the edge (v_i, v_j) in the graph representation, Construction 6 — to find a tree path joining the vertices v_i and v_j, Construction 12 to find the connected component $Y1$, Constructions 13–15 — to recompute tree paths for vertices from $Y1$.

On the STAR-machine, this algorithm is implemented as procedure `InsertEdge` that uses the following input parameters: matrices `Left`, `Right`, `Weight`, and `Code`, endpoints v_i and v_j of the inserted edge, and the number of graph vertices n. It returns the current MST `Tree` and the current matrix of tree paths `TPaths`.

In [7], we have shown that the procedures `DelEdge` and `InsertEdge` take $O(h \log n)$ time each, where h is the number of vertices in the connected component $Y1$. The factor $\log n$ arises due to the use of the basic procedure MATCH.

6. Dynamic reconstruction of an MST after deleting a vertex

In [8], we proposed an associative parallel algorithm for the dynamic update of an MST after deletion of a vertex along with its incident edges. In this section, we select costructions that are used to implement this algorithm on the STAR-machine.

Let $G \setminus v$ denote a graph after deleting the vertex v and its incident edges. We assume that $G \setminus v$ is a connected graph and has an MST. The associative parallel algorithm for the dynamic update of the MST from [8] runs as follows.

At first, it determines connected components obtained after deleting the vertex v and its incident edges from a given MST. Then it determines positions of chords and deletes positions of edges incident on v from the current MST. After that the algorithm determines positions of edges being included into the new MST and saves their endpoints. Finally, it recomputes the matrix of tree paths `TPaths`.

Construction 17 (*Finding positions of edges from the MST incident on the vertex v*). We first determine the binary code `node1` of vertex v. Then we define positions of edges from the MST having an endpoint that coincides with `node1`.

To perform this construction, we apply the basic procedure MATCH.

In [8], there is an example of finding two connected components obtained after deleting a vertex and its incident edges. The next construction will consider a general case when the number of connected components $l \geq 2$. Knowing the given MST, the vertex v, and the matrix of tree paths `TPaths`, it builds a matrix of connected components `Comp`, consisting of n bit columns and $\delta_T(v)$ rows. Its every row saves a separate connected component.

Construction 18 (*Finding connected components after deleting the vertex v*). Let a slice (say, X) save positions of edges from the MST incident on the vertex v. We update these edges as follows. We delete the position i of a current updated edge from the slice X. By means of a variable (say, $w1$), we save the ith row of the matrix `TPaths` and verify whether the vertex v belongs to $w1$. If $v \notin w1$, in the current row of the matrix `Comp`, we write $w1$. Otherwise, we write *not* $w1$.

Really, if $v \in w1$, then all vertices from the subtrees with the root v also belong to $w1$. Therefore the corresponding connected component will consist of the vertices not included into $w1$.

Construction 19 (*Finding the endpoint of an edge incident on v*). Let a row (say, `node1`) save the binary code of a given vertex v and an integer i save the position of an edge (say, γ) incident on v in the graph representation. Then by means of a row (say, $w2$), we save the binary code of the left endpoint of γ. If `node1` = `node2`, then `node2` is assigned to the right endpoint of γ. Finally, by means of the matrix `Code`, the decimal number of `node2` is determined.

To perform this construction, we apply the basic procedure MATCH and the operation FND.

On the STAR-machine, the associative parallel algorithm for finding connected components [8] is implemented as procedure `Subtrees`. It returns the matrix `Comp`, a variable `vdel` to save vertices adjacent to the vertex v, a slice S to save positions of edges remaining after deleting vertex v and all its incident edges, a slice Y to save positions of edges from the MST incident on the vertex v.

From constructing the matrix `Comp`, it is seen to consist of $\delta_T(v)$ different rows, and there is a unique bit $'1'$ in its every column.

Let us enumerate a few constructions used to build a new MST after deleting vertex v and all its incident edges.

Construction 20 (*Finding the connected component including vertex* i). Let the matrix `Comp` and a vertex having the decimal number i be given. To determine the connected component that includes vertex i, we find the ith column of the matrix `Comp` and select the row, whose ith bit is equal to $'1'$.

Construction 21 (*Checking whether endpoints of a chord belong to the same connected component*). Let the matrix `Comp` and a chord (say (i, j)) be given. To determine whether vertices i and j belong to the same connected component of the matrix `Comp`, we find the row that includes vertex i, and then check whether the jth bit of this row is equal to $'1'$.

Construction 22 (*Merging two connected components*). Let the matrix `Comp` and a chord (say $\gamma = (i, j)$) be given. Let endpoints of γ belong to different connected components of the matrix `Comp`. Then a new connected component is obtained by including the connected component containing its right endpoint (vertex j) into the connected component containing its left endpoint (vertex i). Then we write zeros in the matrix `Comp` row, where the connected component including the vertex j has been written.

Knowing connected components obtained after deleting the vertex v from the MST and positions of edges incident on v, the associative parallel algorithm determines a new MST, positions of chords joining different connected components and their endpoints. This algorithm simulates the run of the Kruskal algorithm on the STAR-machine. Initially, it acts on $\delta_T(v)$ connected components each being an MST on the corresponding graph induced by its vertices. At every iteration when the current selected chord of a minimum weight connects two *different* connected components, it unites them together with this chord forming a single connected component. The process continues until all connected components are united in a single connected component.

The associative parallel algorithm for recomputing the matrix of tree paths `TPaths` uses the following main idea.

First, this algorithm selects the connected component w_1 including the root v_1. Tree paths for vertices from w_1 do not change. Then it selects the connected component w_2 which is connected with w_1 by means of a chord. After that, the algorithm determines new tree paths for all vertices from w_2 using the procedure `TreePaths`. Further, it unites connected components w_1 and w_2 and builds a new connected component w_1. The process continues until all connected components are included in w_1.

The associative parallel algorithm for recomputing tree paths is implemented as procedure `ChangePaths`. It uses the following input parameters: matrices `Comp` and `Endpoints`, and the above-described slice Y. The procedure returns a slice `Rep`, a variable `vdel`, and the recomputed matrix `TPaths`.

On the STAR-machine, the associative parallel algorithm for updating an MST after deleting a single vertex is represented as procedure `DeleteVert`. As is shown in [8], it takes $O(h \log n)$ time, where h is the number of vertices whose tree paths change after deleting a vertex.

7. Dynamic reconstruction of an MST after inserting a vertex

In [9], we proposed an associative parallel algorithm for the dynamic update of an MST after insertion of the vertex v_r along with its incident edges. In this section, we select costructions that are used to implement this algorithm on the STAR-machine.

In [9], a given graph is written in the first m rows of the association of the matrices `Left`, `Right`, and `Weight`, and the new edges are written in the next k rows. It is assumed that every new edge is given in the form $(v_r, v_i, wt(v_r, v_i))$, where v_i is adjacent to v_r. Initially, the last k rows of the given matrix `TPaths` consist of zeros.

As is shown in [14], if a new vertex with k incident edges are added to a given MST, then C_k^2 cycles are formed. However, only $k-1$ different edges will be deleted from these cycles. It is assumed that in every selected cycle a *new* edge of a maximal weight is deleted first.

The associative parallel algorithm from [9] is performed in two stages. At the first stage, it builds a new MST and gathers some information for the next stage. At the second stage, it recomputes the matrix of tree paths.

To perform the first stage, the following two constructions are used.

Construction 23 (*Finding candidates for the new MST*). Let a slice (say, `Tree`) save positions of edges from the MST, and a slice (say, X) save positions of edges incident on the vertex v_r. We first determine a maximal weight of edges (say, w) belonging to the MST. Then we save positions of those edges from the slice X, whose weight is less than w. If there are not any new edges, then the position of the new edge of a minimal weight is saved in the slice X.

On the STAR-machine, this construction is implemented as procedure `Candidates`. Knowing the matrix `Weight` and the MST `Tree`, the above construction defines the new content of the slice X. Initially, this slice saves positions of new edges incident on the vertex v_r.

The next construction is applied after performing the previous procedure.

Construction 24 (*Finding different cycles*). Let a slice (say, `Tree`) save positions of edges from the MST, and a slice (say, X) save positions of new edges that are candidates for including in the new MST. We select the uppermost new edge (say, γ_1) and determine the tree path P_1 from the root v_1 to v_r including the edge γ_1. While $X \neq \emptyset$, we select the current uppermost new edge (say γ_2) and determine the tree path P_2 from v_1 to v_r that includes the edge γ_2. The cycle C_i is obtained by means of the operation P_1 *xor* P_2. In this cycle, we determine positions of edges of a maximal weight. If among them there is a new edge, it is deleted from the cycle C_i and its position is marked with $'0'$ in the slice X. Otherwise, the uppermost edge of a maximal weight is deleted from C_i and its position is marked with $'0'$ in the slice `Tree`.

To perform the next construction, we use a matrix Q that consists of two columns of the length $m + k$.

Construction 25 (*Saving the new edge from the tree path used for the next cycle*). Let in the current cycle C_i the tree path P_1 include the new edge γ and the tree path P_2 include the new edge δ. Let in the cycle C_i an edge be deleted from the path P_2. Then the first column of the matrix Q saves the position of the new edge γ from the path P_1 and the second column of Q saves the decimal number of its right endpoint.

On the STAR-machine, the associative parallel algorithm for updating an MST after inserting a new vertex is represented as procedure `InsertVert`. As is shown in [9], it takes $O(h \log n)$ time, where h is the number of vertices whose tree paths change after inserting a new vertex.

Conclusion

In this paper, we select constructions used to represent on the STAR-machine the Prim–Dijkstra algorithm for finding an MST along with the matrix of tree paths and a group of dynamic graph algorithms that require a fast method for finding a tree path between any pair of vertices. It includes the associative parallel algorithms for updating an MST such as, the dynamic edge update of an MST and the dynamic reconstruction of an MST after deleting and after inserting a vertex along with its incident edges. The associative parallel algorithm for finding an MST takes $O(n \log n)$ time. Since associative parallel algorithms for updating an MST take $O(h \log n)$ time each, the best gain is received when $h \ll n$.

The selected constructions can be used to design new associative parallel algorithms and to better understand the run of the vertical processing systems.

References

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

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

[3] 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.

[4] Nepomniaschaya A.S. Basic associative parallel algorithms for vertical processing systems // Bul. Novosibirsk Comp. Center. — Novosibirsk, 2009. — IIS Special Iss. 29. — P. 63–77.

[5] Nepomniaschaya A.S. An associative version of the Bellman–Ford algorithm for finding the shortest paths in directed graphs // Proc. 6th Intern. Conf. PaCT-2001. — Springer, 2001. — P. 285–292. — (LNCS; 2127).

[6] Nepomniaschaya A.S. Comparison of performing the Prim–Dijkstra algorithm and the Kruskal algorithm by means of associative parallel processors // Cybernetics and System Analysis. — Kiev: Naukova Dumka, 2000. — No. 2. — P. 19–27 (In Russian). (English translation by Plenum Press).

[7] Nepomniaschaya A.S. Associative parallel algorithms for dynamic edge update of minimum spanning trees // Proc. 7th Int. Conf. PaCT-2003. — Springer, 2003. — P. 141–150. — (LNCS; 2763).

[8] Nepomniaschaya A.S. Associative parallel algorithm for dynamic reconstructing a minimum spanning tree after deletion of a vertex // Proc. 8th Int. Conf. PaCT-2005. — Springer, 2005. — P. 151–173. — (LNCS; 3606).

[9] Nepomniaschaya A.S. Associative parallel algorithm for the dynamic update of a minimum spanning tree after insertion of a new vertex // Cybernetics and System Analysis. — Kiev: Naukova Dumka, 2006. — No. 1. — P. 19–31 (In Russian). (English translation by Plenum Press).

[10] Nepomniaschaya A.S. Efficient implementation of the Italiano algorithms for updating the transitive closure on associative parallel processors // Fundamenta Informaticae. — IOS Press, 2008. — Vol. 89, No. 2, 3. — P. 313–329.

[11] Nepomniaschaya A.S. Associative version of the Ramalingam decremental algorithm for dynamic updating the single-sink shortest paths subgraph // Proc. 10th Int. Conf. PaCT-2009, Novosibirsk, Russia. — Springer, 2009. — P. 257–268. — (LNCS; 5698).

[12] Nepomniaschaya A.S. Associative version of the Ramalingam algorithm for the dynamic update of the shortest paths subgraph after inserting a new edge // Cybernetics and System Analysis. — Kiev: Naukova Dumka, 2012. — No. 3. — P. 45–57 (In Russian). (English translation by Springer).

[13] Nepomniaschaya A.S. Constructions used in associative parallel algorithms for undirected graphs. Part 1 // Bul. Novosibirsk Comp. Center. — Novosibirsk, 2013. — IIS Special Iss. 35. — P. 69–83.

[14] Pawagi S., Ramakrishnan I.V. An $O(\log n)$ algorithm for parallel update of minimum spanning trees // Information Processing Letters. — 1986. — No. 22. — P. 223–229.