Bull. Nov. Comp. Center, Comp. Science, 10 (1999), 57–72

Finding single-source shortest paths using associative parallel processors*

A.S. Nepomniaschaya and M.A. Dvoskina

In this paper, we employ Dijkstra's algorithm for finding single-source shortest paths in directed graphs. We propose an efficient implementation of this algorithm on a model of associative parallel processors of the SIMD type with bit-serial (or vertical) processing (the STAR-machine). Moreover, we show how to extend this implementation for restoring the shortest path from the source vertex to a given vertex. These algorithms are represented as the corresponding STAR procedures whose correctness is verified and time complexity is evaluated. We also provide an experiment of finding the shortest path between two given vertices in a directed graph.

1. Introduction

Problems of finding the shortest paths are among fundamental tasks of combinatorial optimization because a lot of them can be reduced to finding the shortest path in a network. A number of algorithms are known for a variety of shortest path problems both for directed and undirected graphs. Some of these algorithms are for finding the single source shortest paths (SSSP) and others for the all-pairs shortest paths. The most familiar algorithm for the SSSP problem is Dijkstra's algorithm [1] which sorts the vertices according to their distances from the source vertex. On sequential computers it runs in $O(m + n \log n)$ time if the priority queue is realized with the use of Fibonacci heap [4], where n is the number of graph vertices and m is the number of its edges. In [12], the undirected version of the SSSP problem is solved in $O(m+n)$ time. This algorithm is based on a hierarchical bucketing structure allowing one to avoid the sorting on sequential computers.

We will utilize associative parallel processors for representing Dijkstra's algorithm because such an architecture is primarily oriented to solving non-numerical problems.

Associative (or content addressable) parallel processors belong to fine-grained SIMD systems with bit-serial (or vertical) data processing and simple single-bit processing elements (PEs). This class of supercomputers includes the well-known systems Staran, DAP, MPP and Connection Machine

*Partially supported by the Russian Foundation for Basic Research under Grant 99-01-00548.

[3, 10]. In such systems input data are physically loaded in a matrix memory such that each data item occupies an individual row and is processed with its own processing element [11].

Let us enumerate the problems of finding the shortest paths being represented on associative parallel processors. In [2], Floyd's shortest path algorithm has been represented on the associative array processor LUCAS. In [8], Warshall's transitive closure algorithm, Floyd's shortest path algorithm and Maggs-Plotkin's minimal spanning tree algorithm have been represented on the STAR-machine [5] being an abstract model of associative parallel processors with vertical data processing. We have shown that every of these algorithms takes $O(n^2)$ time assuming that every elementary operation of the STAR-machine (its microstep) takes one unit of time. A special case of Dijkstra's algorithm for finding the shortest path between two vertices in unweighted undirected graphs has been represented both on the associative array processor LUCAS and on the STAR-machine. In [7], we have shown that on the STAR-machine with n PEs it takes $O(n)$ time, while on the associative array processor LUCAS it requires $O(ln)$ time [2], where l is the shortest path length. In [9], there is a specification of Dijkstra's algorithm for finding single source shortest paths on the orthogonal machine.

Here, for directed weighted graphs we propose a natural straight-forward implementation of Dijkstra's algorithm on the STAR-machine. We also show how to extend this implementation in a natural and robust way for restoring the shortest path from the source vertex to any given vertex. These algorithms are represented as the corresponding STAR procedures whose correctness is proved. We have shown that on the STAR-machine with n PEs every of these procedures takes $O(rn)$ time, where r is the number of bits for coding maximum of the shortest distances from the source vertex.

2. Model of associative parallel machine

The model is defined as an abstract STAR-machine of the SIMD type with vertical data processing. It consists of the following components:

- a sequential control unit 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 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 has a dedicated processing element. The rows are numbered from top to bottom and the columns from left to right. A row (word) or a column

(slice) may be accessed equally easy. 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. A vertical register 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 [5] being an extension of Pascal. Recall briefly the STAR constructions needed for the paper. To simulate data processing in the matrix memory, we use three new data types **word**, **slice** and **table**. Constants for the types **slice** and **word** are represented as a sequence of symbols from $\{0, 1\}$ enclosed within single apostrophes. We use the types **slice** and **word** for bit column access and bit row access, respectively, and the type **table** for defining the tabular data. We assume that any variable of the type **slice** consists of p components which belong to $\{0, 1\}$.*

Recall some operations and predicates for slices.

Let X, Y be variables of the type **slice** and i be a variable of the type **integer**. We define 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'.

We utilize the bitwise Boolean operations X *and* Y, X *or* Y, *not* Y, X *xor* Y and predicates ZERO(Y) and SOME(Y) which are introduced in the obvious way.

For a variable T of the type **table** we use the following two operations:

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

COL(i, T) returns the i-th column.

3. Preliminaries

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

Let $G = (V, E, w)$ be a *directed weighted graph* in which $V = \{1, 2, \ldots, n\}$ is a finite set of vertices, $E \subseteq V \times V$ is a finite set of directed edges (arcs)

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

and w is a function that assigns a weight to every edge. We assume that $|V| = n$ and $|E| = m$.

A weight matrix of the graph G is an $n \times n$ matrix which contains as elements the arc weights. It is assumed that if $e = (u, v) \notin E$ then $w(u, v) = \infty$.

In the STAR-machine matrix memory, a directed weighted graph will be represented as a weight matrix. Recall that the weights are integers represented as binary strings.

Let $v_1, v_2, \ldots, v_n$ be a sequence of vertices in G. *A path* from v_1 to v_n is such a sequence of arcs $e_1, e_2, \ldots, e_{n-1}$ that for every $i = 1, 2, \ldots, n-1$ $e_i = (v_i, v_{i+1})$. *The shortest path between two vertices* in a weighted graph is the path with the minimal sum of weights of its arcs.

A tree is a connected acyclic graph. *A tree covers the graph* if its arcs include all the graph vertices. We will deal with rooted trees. *A tree of the shortest paths* in the graph G is a tree with the root vertex s which covers G where for every vertex v there is a unique path from s to v and it is the shortest path between them.

For every vertex v not included into the tree of the shortest paths we define *a conditional shortest path* from the root s as the shortest path between these vertices which goes only through the vertices from the tree of the shortest paths.

Now, we recall a group of basic procedures [6, 8] written in the language STAR which will be used later on. These procedures use the given global slice X for indicating with '1' the row positions being used in the corresponding procedure.

The procedure MATCH(T, X, v, Z) defines positions of those rows of the given matrix T which coincide with the given binary word v. It returns the slice Z in which $Z(i) =' 1'$ if ROW$(i, T) = v$ and $X(i) =' 1'$.

The procedure MIN(T, X, Z) defines the positions of those rows of the given matrix T where the minimal element is located. It returns the slice Z in which $Z(i) =' 1'$ if ROW(i, T) is the minimal matrix element and $X(i) =' 1'$.

The procedure ADDC(T, X, v, F) adds the binary word v to those rows of the matrix T which correspond to the positions '1' in the slice X and writes the result into the corresponding rows of the matrix F. The rows of F, which correspond to positions '0' in the slice X, will consist of components '0'.

The procedure TMERGE(T, X, F) writes into the matrix F those rows of the given matrix T which correspond to positions '1' in the slice X. The rows of the matrix F, which correspond to positions '0' in the slice X, are not changed.

The procedure SETMIN(T, F, X, Y) defines positions of the matrix T rows being less than the corresponding rows of the matrix F. It returns the

slice Y in which $Y(i) =' 1'$ if $\mathrm{ROW}(i, T) < \mathrm{ROW}(i, F)$ and $X(i) =' 1'$.

The procedure $\mathrm{WCOPY}(v, X, F)$ writes the binary word v into those rows of the matrix F which correspond to the positions '1' in the slice X. The rows of the matrix F, which correspond to the positions '0' in the slice X, will consist of components '0'.

The procedure $\mathrm{TCOPY1}(T, j, h, F)$ writes h columns from the given matrix T, beginning with its $(1 + (j-1)h)$-th column, into the result matrix F, where $j \geq 1$.

The procedure $\mathrm{CLEAR}(j, F)$ sets components '0' in j columns of the matrix F.

In [6, 8], we have shown that each of these procedures takes $O(k)$ time, where k is the number of bit columns in the corresponding matrix.

4. Implementation of Dijkstra's algorithm on the STAR-machine

In this section, we propose an efficient implementation on the STAR-machine of Dijkstra's algorithm for finding the single source shortest paths in directed graphs. Recall the main idea of this algorithm.

For every vertex $v \in V$ we have a super distance $D[v] \geq \mathrm{dist}(s, v)$, where $\mathrm{dist}(s, v)$ is the shortest distance from the source vertex s to the vertex v. Besides, we have a set of vertices $S \subseteq V$ belonging to the tree of the shortest paths, that is, $\forall u \in S\ D[u] = \mathrm{dist}(s, u)$. Initially, $S = \{s\}$, $D[s] = 0$ and $\forall v \notin S\ D[v] = \infty$. Let S consist of k vertices $(1 \leq k < n)$ and u be the last vertex added to the set S. Then the $(k+1)$-th vertex for the set S is defined in the following way.

At first, for every vertex $v \notin S$ we define the length of the conditional shortest path from the source vertex s which consists of the shortest path from s to u and the arc (u, v). After that, we select such a vertex v whose conditional shortest path has the minimal length. Then, as proved by Dijkstra, $D[v] = \mathrm{dist}(s, v)$. Therefore, we can append the vertex v to S. Dijkstra's algorithm finishes when $S = V$.

For performing these steps we minimize $D[v]$ as follows. For all arcs $(u, v) \in E$, if $D[u] + w(u, v) < D[v]$ then $D[v] := D[u] + w(u, v)$. We will assume that $x + \infty = \infty$ and $\min(x, \infty) = x$ for all x.

We choose infinity as $\sum_{i=1}^{n} w_i$, where w_i is the maximal weight of arcs incident to vertex i. Let r be the number of bits necessary for coding infinity. Then the weight matrix W consists of rn bit columns and every i-th vertex of the graph G is associated with the i-th *field* having r bit columns.

Remark 1. In view of vertical data processing we assume that every graph will be represented in the STAR-machine memory as a transpose weight

matrix T. Note that some real associative parallel processors allow one easily transpose every matrix.

Before implementing Dijkstra's algorithm on the STAR-machine let us informally explain the meaning of the main variables U, Z of the type **slice** and the variable $R1$ of the type **table**. We use the variable U as a global slice in which positions of vertices belonging to the tree of the shortest paths S are indicated with '0'. The variable Z is used as a result slice for the basic procedure SETMIN. We utilize the variable $R1$ for selecting the i-th field in the matrix T.

```
proc DIJKSTRA(T: table; s,r: integer; inf: word;
              var D: table);
```

/* Here T is the transpose weight matrix, s is the source vertex, r is the number of bits required for representing infinity, *inf* is the binary representation of infinity. */

```
var R1,R2: table; U,X,Z: slice; v: word; k: integer;
```

/* The first stage. */

```
1. Begin
2.   SET(U); U(s):='0';
3.   k:=s;
```

/* Here k saves the last vertex included in the set S. */

```
4.   WCOPY(inf,U,D);
```

/* The second stage. */

```
5.   while SOME(U) do
6.     begin
7.       TCOPY1(T,k,r,R1);
```

/* The k-th field of the matrix T is stored in the matrix $R1$. */

```
8.       MATCH(R1,U,inf,X);
9.       X:=X xor U;
```

/* In the slice X we indicate with '1' positions of those vertices which do not belong to S, but they are incident to the k-th vertex. */

```
10.      if SOME(X) then
11.        begin
12.          v:=ROW(k,D); ADDC(R1,X,v,R2);
```

/* In every i-th row of $R2$, corresponding to $X(i) = '1'$, there is the result of adding the super distance $D[k]$ and the weight of the arc directed from k to i. */

```
13.            SETMIN(R2,D,X,Z);
14.            TMERGE(R2, Z, D)
```

/* In every i-th row of the matrix D, corresponding to $Z(i) = '1'$, we decrease the super distance $D[i]$ to $D[k] + w(k, i)$. */

```
15.          end;
16.        MIN(D,U,X); k:=FND(X);
17.        U(k):='0'
```

/* A new vertex is included in S. */

```
18.      end;
19. End.
```

Correctness of this procedure is established by means of the following theorem.

Theorem 1. *Let a directed weighted graph G be given as the transpose weight matrix T. Let s be the source vertex, every arc weight use r bits and let inf be the binary representation of infinity. Then the procedure* DIJKSTRA(T, s, r, inf, D) *returns the distance matrix D in whose every i-th row there is the shortest distance from s to i and it takes $O(rn)$ time on the STAR-machine with n PEs.*

Proof. At first, by induction on the number of vertices j included in the tree of the shortest paths S, we prove that the procedure DIJKSTRA returns the matrix D.

Basis is verified for $j = 1$. It is obvious that after performing the first stage the tree of the shortest paths S consists of the single vertex s whose position is indicated with $'0'$ in the global slice U. The variable k saves s being the last vertex included in S. Finally, after executing the basic procedure WCOPY(inf, U, D) the s-th row of the distance matrix D consists of components $'0'$, that is, the shortest distance from s to s is equal to zero, while in every other row of D there is the binary representation of the infinity. It means that at this step these shortest distances are unknown.

Step of induction. Assume the theorem is true for $1 \leq j \leq l \leq n-1$. We will prove it for $j = l + 1$. By inductive assumption after including l vertices in S their positions are indicated with $'0'$ in the slice U, the variable k saves the last vertex being appended to S and the shortest distance from s to every i-th vertex from S is written in the i-th row of the matrix D. Since $l \neq n$, we have $U \neq \Theta$.* Therefore we fulfil the second stage of our procedure.

Here, at first, by means of the slice X we will determine positions of those vertices v' $(v' \notin S)$ which are incident to the vertex being included in S at the

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

l-th iteration. To this end after performing line 7 the field of the matrix T, which corresponds to the k-th (or the last) vertex appended to S, is stored in the matrix $R1$. After executing the basic procedure MATCH$(R1, U, inf, X)$ (line 8) for every $1 \leq i \leq n$ we have $X(i) = '1'$ if and only if $U(i) = '1'$. Therefore, as a result of performing line 9 we indicate with $'1'$ in the slice X positions of the vertices which do not belong to S, but they are incident to the vertex being included in S at the l-th iteration. There are two cases.

Case 1: $X \neq \Theta$. Then, for every vertex v' ($v' \notin S$) whose position is indicated with $'1'$ in the slice X we define in parallel the length of the conditional shortest path from s to v' and after that we minimize $D[v']$. To this purpose as a result of performing line 12 in every i-th row of $R2$, which corresponds to the position $'1'$ in the slice X, there is the result of adding the shortest distance from s to the last vertex included in S, that is, $D[k]$, and the weight of the arc directed from this vertex to the vertex i. Now, by means of the basic procedure SETMIN$(R2, D, X, Z)$ (line 13) in the slice Z we indicate with $'1'$ positions of those vertices whose super distances have been decreased due to appending the last vertex to S, that is, $Z(i) = '1'$ if $D[k] + w(k, i) < D[i]$. Then, by means of the basic procedure TMERGE$(R2, Z, D)$ in every i-th row of the matrix D, corresponding to the position $'1'$ in the slice Z, we decrease the super distance $D[i]$ to $D[k] + w(k, i)$.

Finally, we define the position of the current vertex being appended to S at the $(l+1)$-th iteration and indicate it with $'0'$ in the global slice U. With this aim in view we perform the basic procedure MIN(D, U, X) and the statement $k :=$ FND(X) (line 16). As a result the variable k will save the current vertex whose super distance has the minimal weight in the matrix D at the $(l+1)$-th iteration. Then, by means of the statement $U(k) = '0'$ this vertex is appended to the tree of the shortest paths S.

Case 2: $X = \Theta$. Since the k-th vertex appended to S has no vertices incident to it, after line 9 we will perform lines 16–17 as shown above.

Now, let us evaluate time complexity of the procedure DIJKSTRA. Clearly, the cycle from line 5 is performed $n - 1$ times. Since every basic procedure takes $O(r)$ time, we obtain that the procedure DIJKSTRA takes $O(rn)$ time. □

5. Finding the shortest path between two vertices

In this section we will show how to extend the implementation of Dijkstra's algorithm for restoring the shortest path from the source vertex s to every vertex of G. To this end we will construct both the distance matrix D and a special matrix Q of the following form. In its every i-th column we will

indicate with '1' both the position of the vertex which is appended to the tree of the shortest paths S in the i-th iteration and positions of those its descendants whose conditional shortest paths are decreased in this iteration. Then, by means of the matrix Q we will restore the shortest path from the source vertex s to a given vertex of G.

Before presenting the corresponding procedure RECOV let us agree to comment only the statements being used for constructing the matrix Q and some statements for finding the shortest path.

Consider the following procedure.

```
proc RECOV(T: table; s,f,n,r: integer; inf: word;
           var RES: table);
```

/* Here T is the transpose weight matrix, s is the source vertex,
f is the final vertex, n is the number of graph vertices,
r is the number of bits required for representing infinity,
inf is the binary representation of infinity. */

```
var D,Q,R1,R2: table; U,X,Z: slice; v: word; k,l: integer;
```

/* The first stage. */

```
1.  Begin
2.    l:=0; SET(U); U(s):='0';
3.    k:=s;
4.    WCOPY(inf,U,D);
```

/* The second stage. */

```
5.    while SOME(U) do
6.      begin
7.        TCOPY1(T,k,r,R1);
8.        MATCH(R1,U,inf,X);
9.        X:=X xor U;
10.       if SOME(X) then
11.         begin
12.           v:=ROW(k,D); ADDC(R1,X,v,R2);
13.           SETMIN(R2,D,X,Z);
```

/* In the slice Z we indicate with '1' positions of those vertices
whose super distances are decreased at the current iteration. */

```
14.           TMERGE(R2,Z,D)
15.         end;
16.       if ZERO(X) then CLR(Z);
17.       Z(k):='1';
```

/* The position of the k-th vertex is indicated with '1' in the slice Z. */

```
18.       l:=l+1; COL(l,Q):=Z;
```

```
    /* We obtain the current column for the result matrix Q. */
19.       MIN(D,U,X); k:=FND(X);
20.       U(k):='0'
21.     end;
22.   COL(n,Q):=X;
    /* In the slice X we indicate with '1' the position of the vertex
       appended to S at the n-th iteration. */
    /* The third stage. */
23.   CLEAR(n,RES);
24.   SET(U);
    /* In the slice U we indicate with '1' positions of vertices
       included in the tree of the shortest paths. */
25.   k:=f;
    /* Here k is the current vertex belonging to the shortest path. */
26.   for l:=n downto 1 do
27.     begin
28.       X:=COL(l,Q);
29.       if X(k)='1' then
30.         begin
31.           X:=X and U;
32.           COL(k,RES):=X;
    /* The slice X is written in the k-th column of the matrix RES. */
33.           k:=FND(X)
    /* The new value is defined for the variable k. */
34.         end;
35.       U:=U and (not X)
36.     end;
37. End.
```

Correctness of this procedure is established by means of the following theorem.

Theorem 2. *Let G be a directed weighted graph with the source vertex s, the final vertex f and $|V| = n$. Let T be its transpose weight matrix in which every arc weight uses r bits and let inf be the binary representation of infinity. Then the procedure* RECOV$(T, s, f, n, r, inf, RES)$ *takes $O(rn)$ time on the STAR-machine with n PEs. It returns the $n \times n$ Boolean matrix RES in which every arc (i, j) belonging to the shortest path from the vertex*

s to the vertex f is indicated with position '1' at the intersection of the i-th row and the j-th column.

At first, we prove that the procedure RECOV returns the matrix RES. To this end we will employ the following two propositions.

Proposition 1. *Let all assumptions of Theorem 2 be true. Then after performing lines 1–22 we obtain the matrix Q, in whose every j-th column we indicate with '1' the position of the vertex appended to S in the j-th iteration and positions of its descendants, whose conditional shortest paths are decreased at this iteration.*

The proposition is proved by analogy with Theorem 1.

Remark 2. From constructing the matrix Q we conclude that in the first column of Q there are bits '1' in the s-th position and in the positions of all descendants of the vertex s.

Proposition 2. *Let all assumptions of Theorem 2 be true. Then after performing lines 23–37 we obtain the $n \times n$ Boolean matrix RES in which every arc (i, j) belonging to the shortest path from the vertex s to the vertex f is indicated with position '1' at the intersection of the i-th row and the j-th column.*

Proof. We prove by induction on the number of arcs t included in the shortest path from s to f.

Basis is verified for $t = 1$. In view of Proposition 1 before performing the third stage the procedure RECOV constructs the matrix Q. It is obvious that after fulfilling lines 23–25 every column of the matrix RES consists of components '0', the slice U consists of components '1' and the variable k saves the vertex f. Then, as a result of performing line 26 we select the current l-th column of Q using the statement $X := \mathrm{COL}(l, Q)$ (line 28). After that in line 29 we verify whether the f-th bit of X is equal to '1'. There are two cases.

Case 1: $X(f) =' 0'$. Then, after performing line 35 positions of the vertices indicated with '1' in the slice X are set to '0' in the slice U, because these vertices are appended to the tree of the shortest paths S *after* the vertex f.

Case 2: $X(f) = '1'$. Then, in the slice X after performing the statement $X := X$ *and* U there is the unique bit '1'. Really, if $l = n$, then in the slice X there is the unique bit '1' due to lines 22 and 24. Otherwise, in view of Case 1 positions of vertices which are incident to f will be indicated with '0' in the slice U. Now, as a result of fulfilling line 32 in the f-th column of the matrix RES we set the bit '1' in its f-th position. After that by means

of the statement $k := \mathrm{FND}(X)$ we define the new value for the variable k. It is obvious that $k = f$ again. Notice that after performing line 35 we have $U(f) =' 0'$.

Now, by analogy with Case 1 while in the l-th current column ($l \neq 1$) of the matrix Q $X(f) =' 0'$ positions of the vertices appended to S *before* the vertex f are set to '0' in the slice U. Since the shortest path consists of the single arc (s, f) in view of Remark 2 in *the first column* of Q there is the bit '1' both in its s-th position and in the f-th position. Therefore, in the slice X after performing the statement $X := X$ *and* U (line 31) there is the unique bit '1' located in the s-th position, because positions of the vertices incident to the vertex s are indicated with '0' in the slice U. Hence, after fulfilling line 32 in the f-th column of the matrix RES there is the unique bit '1' located in its s-th position.

Step of induction. Assume the proposition is true for $t \leq g < n$. We will prove it for $t = g+1$. Let (s, s_1) be the first arc belonging to the shortest path γ from s to f. Then, we represent γ as $\gamma = \gamma_1\gamma_2$, where $\gamma_1 = (s, s_1)$ and γ_2 is the shortest path from s_1 to f. By inductive assumption, every arc (i, j), belonging to the shortest path γ_2, is indicated with position '1' at the intersection of the i-th row and the j-th column of the matrix RES. After defining the position of the first arc belonging to γ_2 one can easily check that the variable k saves s_1. As a result of performing line 35 we have $U(s_1) =' 0'$. In view of Case 1 it is sufficient to consider the occasion when $X = \mathrm{COL}(1, Q)$. Because of Remark 2 $X(s_1) = '1'$ and we will perform line 31. Since positions of the vertices incident to s are indicated with '0' in the slice U after fulfilling the statement $X := X$ *and* U (line 31) there is the unique bit '1' in the s-th position of the slice X. As a result of performing line 32 in the s_1-th column of the matrix RES the bit '1' is set in the s-th position. Hence, positions of arcs belonging to the shortest path γ are given by means of the matrix RES.

Now, we evaluate time complexity of the procedure RECOV. Obviously, it takes $O(rn)$ time since the first two stages take $O(rn)$ time and the third stage requires $O(n)$ time. □

6. Experiments

In this section, we provide an example to illustrate the implementation of the procedure RECOV for restoring the shortest paths from the source vertex s in the directed weighted graph G. To this end we use our system which simulates the statements of the language STAR by means of Pascal.

The original graph is given in Figure 1. We will execute the procedure RECOV for $s = 1$, $f = 7$, $inf = 11001$ and $r = 5$.

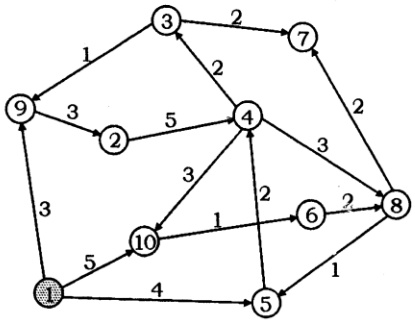

Figure 1. The first iteration

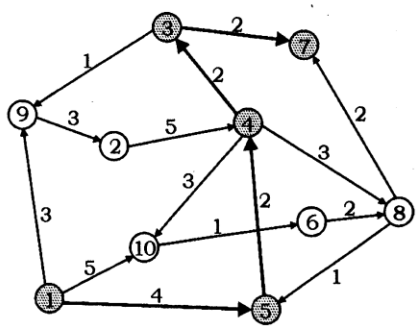

Figure 2. The shortest path from $s = 1$ to $f = 7$

We give values of the following parameters:

- T is a transpose weight matrix;
- U is a slice in which positions of vertices belonging to the tree of the shortest paths indicated with '0';
- D is a distance matrix;
- Z is a slice in which we indicate with '1' positions of the vertices, whose super distances have been decreased in this iteration.

The matrix T consists of ten fields each having five bits:

	1	2	3	4	5	6	7	8	9	10
1	00000	11001	11001	11001	11001	11001	11001	11001	11001	11001
2	11001	00000	11001	11001	11001	11001	11001	11001	00011	11001
3	11001	11001	00000	00010	11001	11001	11001	11001	11001	11001
4	11001	00101	11001	00000	00010	11001	11001	11001	11001	11001
5	00100	11001	11001	11001	00000	11001	11001	00001	11001	11001
6	11001	11001	11001	11001	11001	00000	11001	11001	11001	00001
7	11001	11001	00010	11001	11001	11001	00000	00010	11001	11001
8	11001	11001	11001	00011	11001	00010	11001	00000	11001	11001
9	00011	11001	00001	11001	11001	11001	11001	11001	00000	11001
10	00101	11001	11001	00011	11001	11001	11001	11001	11001	00000

After performing the nineth iteration of the second phase we obtain the matrix Q. Note that the vertices are added to the tree of the shortest paths in the following order: 9, 5, 10, 2, 4, 6, 3, 8, 7. The shortest path from the source vertex 1 to the final vertex 7 is obtained after performing the tenth iteration of the third stage. This path is given in Figure 2.

$N = 1$

Slice		Table D
U	Z	
0	1	00000
1	0	11001
1	0	11001
1	0	11001
1	1	00100
1	0	11001
1	0	11001
1	0	11001
1	1	00011
1	1	00101

$N = 2$

Slice		Table D
U	Z	
0	0	00000
1	1	00110
1	0	11001
1	0	11001
1	0	00100
1	0	11001
1	0	11001
1	0	11001
0	1	00011
1	0	00101

$N = 3$

Slice		Table D
U	Z	
0	0	00000
1	0	00110
1	0	11001
1	1	00110
0	1	00100
1	0	11001
1	0	11001
1	0	11001
0	0	00011
1	0	00101

$N = 4$

Slice		Table D
U	Z	
0	0	00000
1	0	00110
1	0	11001
1	0	00110
0	0	00100
1	1	00110
1	0	11001
1	0	11001
0	0	00011
0	1	00101

$N = 5$

Slice		Table D
U	Z	
0	0	00000
0	1	00110
1	0	11001
1	0	00110
0	0	00100
1	0	00110
1	0	11001
1	0	11001
0	0	00011
0	0	00101

$N = 6$

Slice		Table D
U	Z	
0	0	00000
0	0	00110
1	1	11001
0	1	00110
0	0	00100
1	0	00110
1	0	11001
1	1	11001
0	0	00011
0	0	00101

$N = 7$

Slice		Table D
U	Z	
0	0	00000
0	0	00110
1	0	01000
0	0	00110
0	0	00100
0	1	00110
1	0	11001
1	1	01000
0	0	00011
0	0	00101

$N = 8$

Slice		Table D
U	Z	
0	0	00000
0	0	00110
0	1	01000
0	0	00110
0	0	00100
0	0	00110
1	1	01010
1	0	01000
0	0	00011
0	0	00101

$N = 9$

Slice		Table D
U	Z	
0	0	00000
0	0	00110
0	0	01000
0	0	00110
0	0	00100
0	0	00110
1	0	01010
0	1	01000
0	0	00011
0	0	00101

7. Conclusions

In this paper, we have shown how to perform Dijkstra's sequential algorithm on associative parallel processors with vertical data processing. Such an architecture allows one to perform all the steps of Dijkstra's algorithm in parallel and to avoid the sorting of weights by fulfilling the corresponding search. We have also shown how to extend this implementation for defining the shortest path from the source vertex to every given vertex of G. To this end along with constructing the distance matrix we obtain a special matrix (protocol) to save in any iteration both the position of the vertex appended to the tree of the shortest paths and positions of its descendants whose conditional shortest paths are decreased. Then, using this protocol we restore the shortest path beginning with its last column. We have obtained that on the STAR-machine having no less than n processing elements each of the procedures DIJKSTRA and RECOV takes $O(rn)$ time. In this paper we use a graph representation as a weight matrix because it is best suited for implementing Dijkstra's algorithm on associative parallel processors. Really, if a graph is given as a list of triples (arc vertices and the weight), then we have to use both an additional list of graph vertices and an array for saving the numbers of those vertices which are incident to the vertex added to the tree of the shortest paths in every current iteration.

References

[1] Dijkstra E.W. A note on two problems in connection with graphs // Numerische Mathematik. – 1959. – № 1. – P. 269–271.

[2] Fernstrom C., Kruzela J., Svensson B. LUCAS Associative Array Processor. Design, Programming and Application Studies. – Berlin: Springer–Verlag, 1986. – (Lect. Notes Comput. Sci.; Vol. 216).

[3] Fet Y.I. Vertical processing systems: a survey // IEEE Micro. – 1995 (February). – P. 65–75.

[4] Fredman M.L., Tarjan R.E. Fibonacci heaps and their uses in improved network optimization algorithms // J. ACM. – 1987. – Vol. 34, No 3. – P. 596–615.

[5] 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.

[6] Nepomniaschaya A.S. An associative version of the Prim–Dijkstra algorithm and its application to some graph problems // Andrei Ershov Second Intern. Memorial Conf. "Perspectives of System Informatics". – Berlin: Springer–Verlag, 1996. – P. 203–213. – (Lect. Notes Comput. Sci.; Vol. 1181).

[7] Nepomniaschaya A.S., Taborskaya O.V. Effective representation of some graph problems on associative parallel processors // Proc. of the XII-th Int. Symp. on Comput. and Inform. Sci. (ISCIS XII). – Antalya: Bogazici University Printhouse, 1997. – P. 430–437.

[8] Nepomniaschaya A.S. Solution of path problems using associative parallel processors // Proc. of the Int. Conf. on Parallel and Distributed Systems (ICPADS'97). – Seoul: IEEE Computer Society Press, 1997. – P. 610–617.

[9] Otrubova B., Sykora O. Orthogonal computer and its application to some graph problems // Parcella'86. – Berlin: Academie Verlag, 1986. – P. 259–266.

[10] Potter J.L. Associative Computing: A Programming Paradigm for Massively Parallel Computers. – N.Y., London: Kent State Univ., Plenum Press, 1992.

[11] ASC – An associative computing paradigm / J. Potter, J. Baker, A. Bansal, C. Asthagiri, S. Scott, C. Leangsuksun // Computer: Special Issue on Associative Processing. – 1994. – Vol. 27, No 11. – P. 19–24.

[12] Thorup M. Undirected single source shortest paths in linear time // Proc. of 38-th IEEE Symp. on Foundations of Comput. Sci. (FOCS'97). – 1997. – P. 12–21.