Bull. Nov. Comp. Center, Comp. Science, 19 (2003), 61–71
© 2003 NCC Publisher

Concurrent selection of the shortest paths and distances in directed graphs using vertical processing systems*

A.S. Nepomniaschaya

In this paper, we propose a new implementation of Dijkstra's shortest path algorithm on a model of associative parallel processors with the vertical data processing (the STAR-machine) to obtain for every vertex of a directed graph the distance along with the shortest path from the source vertex. We prove correctness of the corresponding procedure and evaluate its time complexity.

1. Introduction

Finding the shortest paths in networks is an acute problem in the combinatorial optimization. An important version of the shortest path problem is the single-source problem. Given a directed n-vertex and m-edge weighted graph G with a distinguished vertex s, the single-source shortest path (SSSP) problem is to find for each vertex v the length of the shortest path from s to v. When all edge weights are non-negative, the most efficient solution gives Dijkstra's sequential shortest path algorithm [1]. It sorts vertices according to their distances from the source vertex. Dijkstra's algorithm runs in $O(m+n\log n)$ time both on the RAM model [7], when the priority queue is realized using the Fibonacci heap, and on the EREW PRAM model [2], when the priority queue is given by means of relaxed heaps.

To obtain an efficient implementation of Dijkstra's algorithm, we use associative (content addressable) parallel processors of the SIMD type with bit-serial (vertical) data processing and simple single-bit processing elements (PEs) called vertical processing systems in [5]. These systems provide a massively parallel search by contents, enable one to use two-dimensional tables as basic data structures, and the processing of sufficiently large array segments [5, 15].

In [4], a special case of Dijkstra's algorithm for finding the shortest path between two vertices in unweighted undirected graphs has been represented on the associative array processor LUCAS. In [14], there is a specification of Dijkstra's shortest path algorithm on the orthogonal machine.

*Partially supported by the Russian Foundation for Basic Research under Grant 03-01-00399.

In [12], we have proposed an efficient implementation of Dijkstra's algorithm for directed graphs on the STAR-machine, which is an abstract model of the vertical processing systems. The corresponding procedure returns the distance matrix in whose every i-th row there is the length of the shortest path from s to v_i. We have also shown how to extend this implementation in a natural way to restore the shortest path from the source vertex to a given vertex. To this end, we construct the distance matrix along with a special matrix which is used in the sequel to restore the shortest path from s to a given vertex v_k. We have obtained that the corresponding procedures take $O(hn)$ time each, where h is the number of bits required for coding the maximum weight of the shortest paths from the source vertex. It is assumed that each elementary operation of the STAR-machine takes one unit of time.

In this paper, we propose a new implementation of Dijkstra's algorithm on the STAR-machine to *simultaneously* obtain for every vertex of a directed graph the distance and the shortest path from the source vertex. We prove correctness of the corresponding procedure and evaluate its time complexity. We obtain that this procedure takes $O(n \max(h, n))$ time on the STAR-machine having no less than n PEs.

2. Model of associative parallel machine

Our model is based on a Staran-like associative parallel processor [6, 8]. It is defined as an abstract STAR-machine of the SIMD type with the vertical data processing [9]. 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 employed.

The input binary data are loaded in the matrix memory in the form of two-dimensional tables in which each data item 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.

An associative processing unit is represented as h vertical registers 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 elementary operations.

The STAR-machine run is described by means of the language STAR [9] which is an extension of Pascal. Let us briefly consider the STAR constructions being used in this paper. To simulate the 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 components of Y to $'1'$;
CLR(Y)	sets all 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) $'1'$ of Y, $i \geq 0$;
STEP(Y)	returns the same result as FND(Y) and then resets the first $'1'$ found to $'0'$.

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

TRIM(i, j, w)	returns the substring of w having the form $w(i)w(i+1)\ldots w(j)$, where $1 \leq i < j \leq \lvert w \rvert$;
ROW(i, T)	returns the i-th row of the matrix T;
COL(i, T)	returns the i-th column of the matrix T.

Note that all the operations for the type **slice** are also performed for the type **word**.

Remark 1. Note that the STAR statements [9] are defined in the same manner as for Pascal. We will use them later when discussing our procedures.

Following [14], we assume that each elementary operation of the STAR-machine takes one unit of time. Therefore we will measure *time complexity* of the algorithm by counting all elementary operations performed in the worst case.

3. Preliminaries

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

A *weight matrix* of G is an $n \times n$ matrix which contains the arc weights as entries. We assume that $w(u, v) = \infty$ if $(u, v) \notin E$.

Note that the weights are non-negative integers represented as binary strings.

An *adjacency matrix* A for G is an $n \times n$ Boolean matrix in which $a_{ij} = 1$ if $(v_i, v_j) \in E$ and $a_{ij} = 0$, otherwise.

A *path* from u to v in G is a finite sequence of vertices $u = v_1, v_2, \ldots, v_k = v$, where $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k-1$ and $k > 0$. The *shortest path between two vertices* or the *distance* in a weighted graph is a path with the minimum sum of weights of its arcs.

A *tree* is a connected acyclic graph. A *tree of the shortest paths* is a subgraph T of G being a tree with the root vertex s which contains all vertices of G and such that the path from s to any vertex v in T is the shortest path from s to v in G.

Now, consider a group of the basic procedures to be used in the sequel. Implementation of these procedures on the STAR-machine can be found in [10, 11]. These procedures make use of the given global slice X to select by ones the positions of rows being used in the corresponding procedure.

The procedure MATCH(T, X, v, Z) defines the positions of those rows of the given matrix T which coincide with the given pattern v written in the 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 MIN(T, X, Z) defines the positions of those rows of the given matrix T, where minimum elements are located. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW(i, T) is the minimum element of the matrix T and $X(i) = '1'$. To return the minimum element of T, we define the position of the first $'1'$ in Z and then select the corresponding row of T.

Note that the procedures MATCH and MIN are based on the corresponding algorithms first examined in [3].

The procedure SETMIN(T, F, X, Y) defines the positions of rows of the matrix T being less than the corresponding rows of the matrix F. It returns the slice Y, where $Y(j) = '1'$ if and only if ROW$(j, T) <$ ROW(j, F) and $X(j) = '1'$.

The procedure ADDC(T, X, v, F) adds the binary word v to those rows of the matrix T which are selected by ones in X, and writes down the result into the corresponding rows of the matrix F. The rows of F, which are selected by zeros in X, will be set to $'0'$.

The procedure ADDV(T, R, X, F) writes the result of adding the rows of matrices T and R selected by ones in the slice X into the corresponding rows of the matrix F. Note that this procedure is based on the associative algorithm from [6].

The procedure TMERGE(T, X, F) writes into the matrix F those rows of the given matrix T which are selected by ones in X. The rows of the matrix F, which are selected by zeros in X, are not changed.

The procedure WCOPY(v, X, F) writes the binary word v into those rows of the matrix F which are selected by ones in X. The rows of the matrix F, which are selected by zeros in X, will be set to $'0'$.

The procedure TCOPY1(T, j, h, F) writes h columns from the given matrix T, starting with the $(1 + (j - 1)h)$-th column, into the result matrix F.

The procedure CLEAR(j, F) sets zeros in all j columns of the matrix F.

In [10, 11], we have shown that basic procedures take $O(k)$ time each, where k is the number of bit columns in the corresponding matrix.

4. A new implementation of Dijkstra's algorithm on the STAR-machine

Consider the main idea of Dijkstra's algorithm [1]. It assigns the temporal labels $l(v)$ for each vertex $v \in V$ so that $l(v) \geq \text{dist}(s, v)$, where $\text{dist}(s, v)$ is the *distance* from the source vertex s to the vertex v. These labels are constantly decreased by means of a certain iteration procedure, and at each step only a unique temporal label becomes invariant. The algorithm constructs a set of the vertices $F \subseteq V$ in such a way that the current shortest path from s to any vertex of F passes only through vertices in F. Initially, $F = \{s\}$, $l(s) = 0$ and $\forall v \notin F\ l(v) = \infty$. Let F consist of k vertices $(1 \leq k < n)$ and u be the last vertex added to F. Then the $(k+1)$-th vertex for the set F is defined as follows.

At first, we define all the arcs (u, v_i), where $v_i \notin F$. Then for every vertex $v_i \notin F$, we determine the label $l(v_i) = \text{dist}(s, u) + w(u, v_i)$. After that among the vertices $v_i \notin F$ being adjacent with a vertex from F, we select such a vertex v whose label has the minimum value and include it in the set F. On terminating the algorithm, $l(v_i)$ is the weight of the shortest path from s to v_i for all $v_i \in V$.

To perform these steps, the label $l(v)$ is minimized as follows. For all arcs $(u, v) \in E$ if $l(u) + w(u, v) < l(v)$ then $l(v) := l(u) + w(u, v)$.

Infinity will be implemented by the value $\sum_{i=1}^{n} w_i$, where w_i is the maximum weight of the arcs incident to the vertex v_i. Let h be the number of bits necessary for coding the infinity. Then the weight matrix consists of hn columns and any vertex v_i of G is associated with the i-th *field* having h bit columns. On the STAR-machine, a graph will be represented as a weight matrix.

Now, we propose the main idea of the associative parallel algorithm to *simultaneously* select both the shortest path and the distance for all the vertices of the given graph G.

First, by means of the method being used in the procedure DIJKSTRA [12], we define the current vertex v_k which is included into the tree of the shortest paths and the shortest distance from s to v_k. Then we define such a vertex v_i from the tree of the shortest paths which is the next to last one in the shortest path from s to v_k. The shortest path from s to v_k is obtained from the shortest path from s to v_i by adding the vertex v_i to it.

This associative parallel algorithm will be given as procedure DistPath using the following input parameters: a graph G given as a weight matrix; the source vertex s; the number of vertices n; the number of bits h required for representing infinity, and the binary representation of infinity *inf*.

The procedure returns both the distance matrix D, where in every i-th row there is the distance from s to v_i, and the matrix *Paths*, where in every j-th column the positions of vertices included in the shortest path from s to v_j are selected by ones.

This algorithm runs as follows.

1. Define the *position* of the current vertex v_k, which is added to the tree of the shortest paths, and the distance from s to v_k, denoted by $\mathrm{dist}(s, v_k)$.
2. At first, define *positions* of those vertices v_j from the shortest paths tree for which there is an arc entering v_k, and then compute in parallel $\mathrm{dist}(s, v_j) + w(v_j, v_k)$.
3. Select the *position* of such a vertex v_i from the tree of the shortest paths for which $\mathrm{dist}(s, v_i) + w(v_i, v_k) = \min_j\{\mathrm{dist}(s, v_j) + w(v_j, v_k)\}$. After that, select the i-th column of the matrix *Paths*, put $'1'$ in its i-th position, and write down this in the k-th column of *Paths*.

The algorithm terminates when all vertices of G are included into the shortest paths tree.

To present the procedure DistPath, we need the following auxiliary procedures.

The procedure $\mathrm{ADJ}(T, h, n, \mathit{inf}, A)$ uses the weight matrix T, the number of bits h for coding the infinity, the number of vertices n, and the binary code of infinity *inf*. It returns the adjacency matrix A for the matrix T.

The procedure $\mathrm{WTRANS}(w, h, n, R)$ uses the given binary string w and the above explained parameters h and n. It returns the matrix R, where in each i-th row there is the string $v_i = \mathrm{TRIM}((i-1)h + 1, ih, w)$. In other words, the matrix R is a transpose of the given string w.

In Appendix we will show that the procedure ADJ takes $O(hn)$ time, while the procedure WTRANS requires $O(n)$ time.

Now, we present the procedure DistPath.

```
proc DistPath(T: table; s,h,n: integer; inf: word;
               var D,Paths: table);
/* Here, $T$ is the weight matrix, $s$ is the source vertex, $h$ is the number of bits
   required for representing infinity, inf is the binary representation of infinity. */
var A,R1,R2: table; U,X,Z: slice(T);
    k,j: integer; v1,v2: word;
1.  Begin ADJ(T,n,h,inf,A);
2.    CLEAR(n,Paths);
3.    SET(U); U(s):='0';
4.    k:=s; WCOPY(inf,U,D);
/* Here, $k$ saves the last vertex included in the set $F$. */
5.    while SOME(U) do
6.      begin v1:=ROW(k,T);
7.        WTRANS(v1,h,n,R1);
/* The result of transposing the $k$-th row of the matrix $T$ is saved
   in the matrix $R1$. */
8.        MATCH(R1,U,inf,X);
9.        X:=X xor U;
/* We indicate by '1' in the slice $X$ positions of the vertices which do not belong
   to $F$, but they are adjacent to the vertex $v_k$. */
10.       v2:=ROW(k,D);
11.       ADDC(R1,X,v2,R2);
/* The result of adding $l(v_k)$ and the weight of the arc directed from $v_k$ to $v_i$
   is written in every $i$-th row of $R2$, which corresponds to $X(i) = '1'$. */
12.       SETMIN(R2,D,X,Z);
13.       TMERGE(R2,Z,D);
/* We decrease the label $l(v_i)$ to $l(v_k) + w(v_k, v_i)$ in every $i$-th row
   of the matrix $D$ which corresponds to $Z(i) = '1'$. */
14.       MIN(D,U,X); k:=FND(X);
15.       U(k):='0';
/* A new vertex is included in $F$. */
16.       X:=COL(k,A);
17.       X:=X and (not U);
/* Positions of vertices from $F$ for which there is an arc entering $v_k$
   are selected by ones in the slice $X$. */
18.       TCOPY1(T,k,h,R1);
/* The $k$-th field of the matrix $T$ is stored in the matrix $R1$. */
19.       ADDV(R1,D,X,R2);
```

```
    /* The weights of paths from s to v_k selected by ones in X are written
       in the corresponding rows of the matrix R2. */
20.         MIN(R2,X,Z);
21.         i:=FND(Z);
    /* The vertex v_i is the next to the vertex v_k in the shortest path from s to v_k. */
22.         X:=COL(i,Paths); X(i):='1';
23.         COL(k,Paths):=X
24.      end;
25. End.
```

Remark 2. It is easy to verify that the basic procedure SETMIN($R2, D, X, Z$) (line 12) is applied to the vertices of G which have not yet been included into the tree of the shortest paths. Therefore in the result slice Z the positions of such vertices are selected by ones. Hence, execution of the basic procedure TMERGE($R2, Z, D$) (line 13) does not change the rows of D, where distances from the source vertex to the vertices belonging to the tree of the shortest paths, are written.

Theorem 1. *Let $G = (V, E, w)$ be a directed weighted graph with the source vertex s and $|V| = n$. Let T be its weight matrix, where every arc weight uses h bits and let inf be the binary representation of infinity. Then the procedure DistPath($T, s, h, n, inf, D, Paths$) returns the matrix D, where in every i-th row there is the distance from s to v_i, and the matrix Paths, where in every i-th column positions of vertices included into the shortest path from s to v_i are selected by ones. It takes $O(n \max(h, n))$ time on the STAR-machine having no less than n PEs.*

Proof. We prove this by induction on the number of vertices q in the tree of the shortest paths F.

Basis is verified for $q = 1$. On performing lines 1–2, we obtain the adjacency matrix A for the given weight matrix T and the matrix *Paths* consisting of zeros. After performing line 3, the vertex s is included into the tree of the shortest paths F. On performing line 4, the distance from s to s is written in the s-th row of the distance matrix D and it is equal to zero, and the variable k saves s.

Step of induction. Let the assertion be true for $1 \leq q \leq l \leq n-1$. We will prove it for $l = q + 1$. By the induction hypothesis, the first l vertices are included in the tree F by setting zeros in the corresponding positions of the slice U, the variable k saves the current vertex included in F, and for every vertex v_i from F the distance from s to v_i is written in the i-th row of D and *positions* of vertices belonging to the shortest path from s to v_i are selected by ones in the i-th column of the matrix *Paths*.

Since $U \neq \Theta$, we perform the $(l+1)$-th iteration. By analogy with proving correctness of the procedure DIJKSTRA [12], on performing lines 6–15, a new vertex v_k is added to F and the shortest path from s to v_k is written in the k-th row of D. In view of Remark 2, adding the vertex v_k to F does not change the distances from s to other vertices of F written in the matrix D.

On fulfilling lines 16–17, *positions* of those vertices from F, for which there is an arc entering v_k, are selected by ones in the slice X. As a result of performing lines 18–19, we compute in *parallel* the sums $\text{dist}(s, v_j) + w(v_j, v_k)$ for all the vertices v_j selected by ones in X and save the results in the corresponding rows of the matrix $R2$. Then by means of the basic procedure $\text{MIN}(R2, X, Z)$ (line 20), we select a path from s to v_k having the minimum value among all the paths selected by ones in X. Now after performing line 21, we determine such a vertex v_i which is the next one to the last vertex v_k in the shortest path from s to v_k. Since the vertex v_i has been included in the tree of the shortest paths before v_k, we obtain $i \neq k$. By the induction assumption, the shortest path from s to v_i is written in the i-th column of the matrix *Paths*.

Finally, on performing lines 22–23, the shortest path from s to v_k is written in the k-th column of the matrix *Paths*. Hence, after including the vertex v_k in the tree of the shortest paths F, the distance from s to v_k is written in the k-th row of the matrix D and the shortest path from s to v_k is written in the k-th column of the matrix *Paths*. Since the slice U consists of zeros, the procedure terminates.

Now, we evaluate the time complexity of the procedure DistPath. We first observe that execution of lines 1-4 takes no more than $O(hn)$ time in view of the auxiliary procedure WTRANS. Inside the cycle the basic procedures take $O(h)$ time each, while the auxiliary procedure WTRANS takes $O(n)$ time. Since the cycle is executed n times, we obtain that the procedure DistPath takes $O(n \max(h, n))$. $\square$

5. Conclusions

In this paper, we have proposed a new efficient implementation of Dijkstra's shortest path algorithm on the STAR-machine which allows one for every vertex of a directed graph to *simultaneously* obtain the distance and the shortest path from the source vertex. In [12], to restore the shortest path from the source vertex to a given vertex f, we first construct the distance matrix along with an auxiliary matrix being the protocol of this computation. Then by means of this protocol, we restore the shortest path starting from the vertex f. Here, taking into account the main advantages of vertical processing systems, we can determine the vertex along with the arc which is added to the tree of the shortest paths at each iteration. We have proved correctness of the procedure DistPath and evaluate its time complexity.

In the same manner, one can modify the implementation of the Bellman-Ford shortest path algorithm on the STAR-machine [13] to determine for every vertex of a directed graph the distance along with the shortest path from the source vertex.

References

[1] Dijkstra E.W. A note on two problems in connection with graphs // Numerische Mathematik. – 1959. – Vol. 1. – P. 269–271.

[2] Driscoll J.R., Gabow H.N., Shrairman Ruth, Tarjan R.E. Relaxed heaps: an alternative to Fibonacci heaps with applications to parallel computation // Communications of the ACM. – 1988. – Vol. 31, № 11. – P. 1343–1354.

[3] Falkoff A.D. Algorithms for parallel–search memories // J. ACM. – 1962. – Vol. 9, № 10. – P. 488–510.

[4] Fernstrom C., Kruzela J., Svensson B. LUCAS associative array processor. Design, programming and application studies. – Berlin: Springer-Verlag, 1986. – (Lecture Notes in Computer Science; Vol. 216).

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

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

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

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

[9] 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.

[10] 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"/ Lecture Notes in Computer Science. – Berlin: Springer-Verlag, 1996. – Vol. 1181. – P. 203–213.

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

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

[13] 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 / Lecture Notes in Computer Science. – Berlin: Springer-Verlag, 2001. – Vol. 2127. – P. 285–292.

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

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

Appendix

Here, we propose the following auxiliary procedures.

```
proc ADJ(T: table; h,n: integer; inf: word; var A: table);
/* Here, A is the adjacency matrix for the given weight matrix T. */
var i: integer; X,X1,Y: slice; R: table;
Begin
  SET(X);
  for i:=1 to n do
    begin
      TCOPY1(T,i,h,R);
      MATCH(R,X,inf,Y);
      X1:= not Y;
      COL(i,A):=X1
    end;
End.
```

```
proc WTRANS(w: word; h,n: integer; var R: table);
/* Here, the string w will be cut into n pieces each of length h. */
var v: word; i: integer;
Begin  for i:=1 to n do
    begin
      v:=TRIM((i-1)h+1,ih,w);
      ROW(i,R):=v
    end;
End.
```