Bull. Nov. Comp. Center, Comp. Science, 3 (1995), 77–88

Effective representation of algorithm for finding a minimal spanning tree of a graph in associative parallel processor

A.Sh. Nepomniaschaya

In this paper we analyze two procedures for finding a minimal spanning tree of a graph for an abstract associative STAR-machine with bit-serial processing. These procedures are based on the Prim-Dijkstra algorithm and use different graph representations. We prove their correctness, evaluate their complexity and compare them. In addition, we compare two procedures for the same graph representation.

Key words: Associative parallel processor, bit-serial processing, parallel algorithm, complexity of algorithms, undirected weighted graph, minimal spanning tree of a graph.

1. Introduction

The revived interest in associative (content-addressable) architectures is caused by declining hardware prices due to modern technology achievements [1]. We will analyze algorithms for STARAN-like associative parallel processors (APPs) belonging to fine-grain SIMD-systems with bit-serial (vertical) processing and simple single-bit processor elements. For such computers algorithms are represented by means of languages ASC [2], Apple [3] or some special tools (for example, [4–5]). To simulate the complete parallel associative processing in micro-level, carry out massive associative searching of tabular data written in binary code and research new vertical processing algorithms a high-level language STAR was proposed in [6].

In this paper an approach for analyzing algorithms of associative processing is considered. Its novelty lies in the fact that for a group of parallel architectures a formal semantic model (the STAR-machine) is defined which is used both for programmed modeling and for theoretical analysis of algorithms. Here we use our approach for comparing some versions of realizing the Prim-Dijkstra algorithm [7, 8] for finding a minimal spanning tree of a graph. For deciding this problem on sequential computers there are different well-known algorithms, however for APPs the Prim-Dijkstra algorithm is best suited [2, 9]. Algorithm versions are written as STAR procedures.

We prove their correctness and evaluate their complexity in terms of access number to the matrix memory [4] of the model employed.

For a graph representation as a set of triples (edge nodes and their weight) we compare the procedure $MST1$ using the Prim-Dijkstra algorithm with the procedure $MSTP$ using its modification (the Baase algorithm) [10]. The improved estimate for the procedure $MST1$ has been obtained due to account of special features of associative parallel processors with vertical processing. The STAR-procedure $MSTP$ is based on the program MST written in the language ASC [2].

For improving the procedure $MST1$ estimate we consider the procedure $MST2$ which uses a special representation of graph edges as vertical pairs.

In parallel with proving the correctness of considered procedures we have verified them using our STAR-convertor on IBM PC.

This research may be useful for designing both a knowledge base for vertical processing system and specialized associative processors for finding minimal spanning trees of graphs.

2. Model of associative parallel machine

We define our model as an abstract STAR-machine of the SIMD-type with vertical data processing. It consists of the following components:

- a sequential common control unit (CU) where the programs and scalar constants are stored;
- associative processor consisting of m single-digit processor elements (PEs);
- matrix memory for the associative processor.

Data are loaded in the matrix memory in the form of two-dimensional array written in binary code. Each array element occupies an individual row and all elements have the same length (coinciding with the length of the maximal 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 arrays may be loaded in the matrix memory. In the CU a rendition table should be located allowing one to associate with each array identifier its number of columns.

The associative processor is represented as h vertical registers each of m bits. The bit columns of the data array are stored in the registers which perform the necessary Boolean operations, record the search results and ensure the word selection capability. We assume that the STAR-machine processor has a sufficient number of vertical registers to store intermediate results of data processing without using the matrix memory.

3. Review of the language STAR

In this section we briefly consider STAR constructions from [6] needed for the paper. To simulate data processing in the matrix memory the following data types are used: **integer, boolean, word, slice, table** and **array.** Data types are introduced in the same manner as in Pascal. 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. Assume that any variable of the type **slice** consists of m components which belong to $\{0, 1\}$.

Consider operations, predicates and some functions for slices.

Let X, Y be variables of the type **slice** and i, k be variables of the type **integer**. Consider that ONE-component and ZERO-component of a slice denotes its component with the value $'1'$ and $'0'$, respectively. Define 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) yields the ordinal number i of the first ONE-component of Y, $i \geq 0$;

MASK1(Y,k) sets the alternation in Y consisting of k zeros and k ones, where $k = 2^i$, $i \geq 0$. (For example, MASK1(Y,1) denotes the alternation of the form $'01'$).

The following logical operations are executed simultaneously by all corresponding components of X and Y. We introduce them in the usual way:

$X \wedge Y$ is conjunction, $X \vee Y$ is disjunction, $\neg X$ is negation. Other logical operations are constructed from these operations by means of superposition.

The following two predicates will be utilized:

ZERO(Y) yields **true** if and only if the slice Y consists of ZERO-components;

SOME(Y) yields **true** if and only if the slice Y has at least one component with value $'1'$.

We will use the following two standard functions:

Shift(Y,down,k) moves the contents of Y on k positions down, placing each component from the position N to the position $N+k$ ($N \geq 1$) and setting ZERO-components from the first through the k-th positions, inclusive;

Shift(Y,up,k) moves the contents of Y on k positions up, placing each component from the position N ($N \geq k+1$) to the position $N - k$ and setting ZERO-components to the last k components of Y.

Let T be a variable of the type **table**. We will use the following operation: $T(i)$ yields the i-th row in the matrix T $(1 \leq i \leq m)$.

Remark 1. Note that STAR statements resemble those of Pascal.

Remark 2. When we define a variable of the type **slice** we put in brackets the name of the matrix which uses it. Therefore, if the matrix consists of n rows, where $n < m$, then only first n components of the corresponding slice (column) will be used in the vertical processing.

4. Problem of finding a minimal spanning tree of a graph

In many applications a problem of finding a minimal spanning tree for undirected weighted graphs often appears. For solving it on sequential computers algorithms of Prim-Dijkstra, Kruskal, Tarjan-Cheriton and others are used. Following [2, 9], for APPs we use the Prim-Dijkstra algorithm. To consider it some notions are necessary.

Let $G=(V, E, w)$ represent an *undirected weighted graph* with the node (vertex) set $V = \{1, 2, \ldots, n\}$, the edge set $E \subseteq V \times V$ and the weight function w correlating each edge $(i, j) \in E$ with an integer $w(i, j)$.

A minimal spanning tree T_S of the graph $G = (V, E, w)$ is defined as a connected graph without loops containing all nodes from V where the sum of weights of the corresponding edges is minimal.

In this paper we examine only undirected weighted graphs having the node set $V = \{1, 2, \ldots, n\}$. Therefore, let us agree to omit these notions later on.

The Prim-Dijkstra algorithm constructs a minimal spanning tree of a graph by means of extension of a subtree of T_S. As initial edge of T_S an arbitrary edge of the graph with minimal weight is selected. Let there be k edges in T_S, where $k \geq 1$. Then the $(k+1)$-th edge is selected in the following way. It is necessary to define all edges having only one node which belongs to T_S and among them to select an edge with the minimal weight. The extension process of T_S is finished as soon as the number of edges is equal to $n - 1$.

To analyze different variants of representing the Prim-Dijkstra algorithm in associative parallel processors we use the procedures MATCH and MIN from [11]. The procedure MATCH(T, X, w, Z) defines the row positions in T coinciding with the given w. Its result is the slice Z in which $Z(i) =' 1'$, if $T(i)=w$. The procedure MIN(T, X, Z) defines the row positions in T, where the minimal element is located. It yields the slice Z in which $Z(i) =' 1'$ if $T(i)$ is the minimal matrix element. Recall that these procedures realize the

search only among the matrix rows which correspond to positions '1' in the slice X.

For algorithm analysis we will use the following notation. For a given algorithm P denote by $N(P)$ *the access number* to the STAR-machine matrix memory during its execution. Following [4], we assume that definition of the response existence among PEs does not use additional time.

5. Representation of the Prim-Dijkstra algorithm

In this Section we investigate two versions of the Prim-Dijkstra algorithm depending on graph representation forms. We construct the corresponding procedures and evaluate their complexity.

5.1. The use of representing edges as horizontal pairs

Here we represent a graph $G=(V, E, w)$ in the form of association of the matrices *left*, *right* and *weight* in which each edge $(i, j) \in E$ is matched with the triple $< i, j, w(i, j) >$, where $i \in left$, $j \in right$, $w(i, j) \in weight$.

Before considering the procedure MST1 let us informally explain the meaning of its variables: node1 and node2 of the type **word** and $S, N1$ and $N2$ of the type **slice**. The variables node1 and node2 are used for storing the left and the right nodes of the current edge which is added to the fragment of T_S. The variable S points by '1' the triple positions where the search is realized by means of the procedure MATCH. The variables $N1$ and $N2$ are used for accumulation of triple positions which are potential candidates for adding to the fragment of T_S. Other variables are used for storing intermediate results.

Now consider the procedure MST1.

```
proc MST1(left, right, weight: table; n: integer; var R: slice(left));
label 1; var i, r: integer; node1, node2: word;
S, N1, N2, X, Y, Z: slice(left);
begin CLR(R); CLR(N1); CLR(N2); SET(Z); SET(S); r := 0;
 while r ≤ n − 1 do
  begin MIN(weight,Z,X); i:=FND(X);
    R(i):='1'; r:=r + 1; if r = n − 1 then goto 1;
    node1:=left(i); node2:=right(i); S(i) :=' 0';
/* The i-th edge position is deleted from S */
    MATCH(left, S, node1, Z); N1 := N1 ∨ Z;
    MATCH(left, S, node2, Z); N1 := N1 ∨ Z;
    MATCH(right, S, node1, Z); N2 := N2 ∨ Z;
    MATCH(right, S, node2, Z); N2 := N2 ∨ Z;
```

/* Positions of potential candidates are accumulated in $N1$ and $N2$ */
$Y := N1 \wedge N2$; **if** SOME(Y) **then** $S := S \wedge \neg Y$;
/* We delete from S positions of edges which do not
belong to T_S but both their nodes are in T_S */
$Z := N1 \vee N2$; $Z := Z \wedge S$
end;
1: **end**

To prove the correctness of this procedure we will utilize the following Lemma which is verified by contradiction.

Lemma 1. *Let a graph G be represented as a matrix M_G being an association of matrices left, right and weight. Let by means of the procedure MST1(left, right, weight, n, R) a minimal spanning tree be constructed where q is the last added node. Let a graph G' be obtained from the graph G by deleting the node q together with all edges incident to it and respectively $M_{G'}$ be obtained from M_G by deleting all triples containing q. Then MST1 constructs for M_G and $M_{G'}$ minimal spanning trees having the same first $n-2$ edges.*

Theorem 1. *Let a graph G be represented as an association of matrices left, right and weight. Then the procedure MST1(left, right, weight, n, R) constructs a minimal spanning tree whose edges are indicated by positions of '1' in the slice R.*

Proof. We will prove the theorem by induction on the number of edges in the minimal spanning tree T_S of the graph G.

Basis is directly checked.

Step of induction. Assume the claim is true for graphs having no more than k nodes ($k \leq n-1$). We prove it for graphs with $k+1$ nodes. In view of inductive assumption by Lemma 1 the procedure MST1(*left*, *right*, *weight*, $k+1$, R) selects positions of the first $k-1$ edges of M_G which belong to T_S. Since $r = k-1$ then the statement **if** $r = k$ **then goto** 1 is not realized. For finding the k-th edge for T_S we carry out the following three steps:

(1) define positions of new edges (candidates) which appear after adding the $(k-1)$-th edge to T_S;

(2) define positions of edges which should be deleted from further analysis;

(3) choose the k-th edge for T_S.

At the first step, after executing the statement i:=FND(X), we define the row position in which the $(k-1)$-th edge for T_S is located. To store its nodes we carry out the statements node1:=*left*(i) and node2:=*right*(i) and after using the statement $S(i) :=' 0'$ we delete the $(k-1)$-th edge from

the further analysis. After executing the procedure MATCH(*left*, S, node$j$, Z) row positions of the matrix *left* coinciding with the node$j$ for $j = 1, 2$ are stored in the slice Z. We add these positions to the slice $N1$ by the statement $N1 := N1 \vee Z$. Similarly by means of the statement $N2 := N2 \vee Z$ we add row positions of the matrix *right* coinciding with node1 and node2.

At the second step by analogy with [2] using the statement $Y := N1 \wedge N2$ we define edge positions whose both nodes belong to T_S, but they are not in T_S. For eliminating loops such edges should be deleted from the further analysis by using the statement **if** SOME(Y) **then** $S := S \wedge \neg Y$. Then by means of the statement $Z := N1 \vee N2$ we determine the set of potential candidates. We can prove that for defining the proper candidates it is necessary to delete from Z the positions of edges included in Y and the row position where the $(k-1)$-th edge is located. Therefore we carry out the statement $Z := Z \wedge S$ and then jump to the end of the iteration statement **while** $r \leq k - 1$ **do**.

At the third step after executing the procedure MIN(*weight*, Z, X) we select the candidates with the minimal weight among candidates whose positions are marked by '1' in the slice Z. By the statement i:=FND(X) we define the position of the next (that is, k-th) edge and add it to T_S using the statement $R(i) :=' 1'$. After executing statements $r := r + 1$ and **if** $\mathbf{r = k}$ **then goto 1** we jump to the procedure exit. □

The following theorem can be immediately proved.

Theorem 2. *Each statement in the procedure MST1(left, right, weight, n, R) is essential.*

For evaluating the complexity of the procedure MST1 at first we estimate the complexity of procedures MATCH and MIN. Let m be a maximal number in the matrix *weight*. In view of [11] we obtain $N(MATCH) \leq 1 + 2\log_2 n$ and $N(MIN) \leq 1 + 3\log_2 m$, where n is the number of graph nodes. Therefore,

$$N(\text{MST1}) \leq 8 + 3\log_2 m + (18 + 3\log_2 m + 8\log_2 n)(n-2).$$

5.2. The use of representing edges as vertical pairs

Here we define a graph G in the form of matrix M^*, where each edge (i, j) is represented by means of two rows: the node i and weight $w(i, j)$ are written in the row with an odd number and the node j is written directly under the node i in the row with an even number. Assume that $i, j \in$ *vertex*, $w(i, j) \in$ *weight*. Note that the matrix *vertex* represents alternation of rows belonging to matrices *left* and *right* when the graph is given as an association of triples.

Remark 3. In [2], Potter suggests a method for recording input data directly into the matrix memory of APP. Since the matrix *vertex* consists of repeating nodes from the matrices *left* and *right*, then it can be more effectively constructed than each of the matrices *left* and *right* separately.

Consider the procedure MST2 which uses the representation of a graph in the form of the matrix M^*. Let us agree that for each edge (p, q) included in T_S its position in M^* is defined by the row where the node p and the weight $w(p, q)$ are located.

```
proc MST2(vertex, weight: table; n: integer; var R: slice(vertex));
label 1; var i, r: integer; node1, node2: word;
A, B, S, N1, N2, X, Y, Z, Z1, Z2: slice(vertex);
begin CLR(R); CLR(N1); CLR(N2); SET(S);
  MASK1(B, 1); A:=shift(B,up,1); Z := A; r:=0;
/* We use the slice B for masking rows with even numbers and
  slices A and Z for masking rows with odd numbers */
  while r ≤ n − 1 do
  begin MIN(weight,Z, X); i:=FND(X);
    R(i) :=' 1'; r := r + 1; if r = n − 1 then goto 1;
    node1:=vertex(i); node2:=vertex(i + 1); S(i) :=' 0';
    S(i + 1) :=' 0'; MATCH(vertex,S,node1,Z); Z1 := Z ∧ A;
    N1 := N1 ∨ Z1; Z2 := Z ∧ B; N2 := N2 ∨ Z2;
/* Positions of odd (respectively even) rows coinciding
  with node1 are added to slice N1 (respectively N2) */
    MATCH(vertex,S,node2,Z); Z1 := Z ∧ A; N1 := N1 ∨ Z1;
    Z2 := Z ∧ B; N2 := N2 ∨ Z2;
/* Positions of odd (respectively even) rows coinciding
  with node2 are added to slice N1 (respectively N2) */
    N2:=shift(N2,up,1); Y := N1 ∧ N2; if SOME(Y) then
      begin S := S ∧ ¬Y; Y:=shift(Y,down,1); S := S ∧ ¬Y
/* We exclude from the slice S edge positions which do not
  belong to T_S but both their nodes are in T_S */
      end;
      Z := N1 ∨ N2; Z := Z ∧ S
  end;
1: end
```

To prove the correctness of the procedure MST2 we use Lemma 2 which is cited by analogy with Lemma 1.

Theorem 3. *Let a graph G be represented as an association of the matrices vertex and weight. Then the procedure MST2(vertex, weight, n, R) constructs a minimal spanning tree whose edges are indicated by positions of '1' in the slice R.*

Proof. This theorem is proved by analogy with Theorem 1. Consider main differences in the proof.

By means of the first four statements the initial values are set into slices R, $N1$, $N2$ and S. Then in view of the graph representation after executing statements MASK1(B,1), A:=shift(B,up,1) and $Z := A$ we set the mask for separating rows with even (respectively odd) numbers in the slice B (respectively in slices A and Z). This value of Z is used in the procedure MIN(*weight*, Z, X) for defining the position of the first edge for T_S.

Step of induction. In view of inductive assumption by Lemma 2 the procedure MST2(*vertex*, *weight*, $k+1$, R) selects positions of the first $k-1$ edges of M^* which belong to T_S.

After executing the procedure MATCH(*vertex*, S, node1, Z) in the slice Z we define row positions of the matrix *vertex* coinciding with node1. Positions of the left nodes of edges coinciding with node1 are located in the rows having odd numbers. We select them by the statement $Z1 := Z \wedge A$ and add to the slice $N1$ using the statement $N1 := N1 \vee Z1$. Positions of the right nodes of edges coinciding with node1 are located in the rows having even numbers. We extract them by the statement $Z2 := Z \wedge B$ and then add to the slice $N2$ by the statement $N2 := N2 \vee Z2$. Similar reasoning is used for node2. Hence, we separate positions of rows coinciding with node1 and node2 into positions with odd numbers (slice $N1$) and positions with even numbers (slice $N2$).

Other distinctions are caused by excluding some edges from the further analysis. Consider some of them.

For each edge the position of its right node is shifted on one bit down with respect to the left node position. Therefore, before defining edge positions excluded from the further analysis it is necessary to execute the statement $N2$:=shift($N2$, up,1). For eliminating the candidate positions belonging to the slice Y we carry out the statements $S := S \wedge \neg Y$, Y:=shift(Y, down, 1) and $S := S \wedge \neg Y$.

Finally, it is not difficult to verify that positions of potential candidates are defined by means of the statement $Z := N1 \vee N2$ *after shifting* the contents of $N2$. □

Evaluate the complexity of the procedure MST2.

$$N(\text{MST2}) \leq 10 + 3\log_2 m + (24 + 3\log_2 m + 4\log_2 n)(n-2).$$

For $n \geq 4$ we obtain that $N(\text{MST1}) - N(\text{MST2}) < 4\log_2 n(n-2)$.

Note that each graph edge is processed by its own processor element in the STAR-machine. Hence, the procedure MST2 is used if the number of its edges is less or equal to half of the processor element number.

6. Representation of the Baase algorithm

In [2] an effective program MST for finding the minimal spanning tree of a graph has been written by using the associative computing language ASC. This program is based on the Baase algorithm and utilizes the same graph representation as the procedure MST1. For comparing MST1 and the program MST we have written the STAR procedure MSTP for the Potter MST program and then compare the corresponding STAR procedures.

At first we briefly consider the Baase algorithm being a modification of the Prim-Dijkstra one. It separates all graph edges into four states. Edges belong to state 1 (respectively state 4) if they are included in (respectively excluded from) the fragment of the minimal spanning tree. State 3 consists of edges which have not been considered yet. State 2 contains edges which connect a node in state 1 with a node in state 3. The algorithm iteration includes the location of the minimal weight edge in state 2, setting it to state 1 and changing the edge states which take part in this selection. The process is iterated while there are edges having state 2.

Note that the STAR procedure MSTP utilizes all variable names from the Potter MST program and for the similar purposes. Other variables are used for intermediate results.

Consider the procedure MSTP.

```
proc MSTP(nodel, noder, weight: table; var state1: slice(nodel));
var graph, reachl, reachr, state2, state3, state4,
X, Y, Z: slice(nodel); node1, node2: word; i: integer;
begin CLR(reachl); CLR(reachr); CLR(state1); CLR(state2);
  CLR(state4); SET(graph); MIN(weight, graph, X);
  i:=FND(X); state2(i):='1'; state3:= ¬state2;
  while SOME(state2) do
  begin MIN(weight, state2, X); i:=FND(X);
    state1(i):='1'; state2(i):='0'; node1:=nodel(i);
    node2:=noder(i); graph:=state2∨state3;
    MATCH(nodel, graph, node1, Z); reachl:=reachl∨Z;
    MATCH(nodel, graph, node2, Z); reachl:=reachl∨Z;
    MATCH(noder, graph, node1, Z); reachr:=reachr∨Z;
    MATCH(noder, graph, node2, Z); reachr:=reachr∨Z;
    X:=reachl∧reachr; graph:=graph∧X;
    if SOME(graph) then
      begin state2:=state2∧¬X; state3:=state3∧¬X;
        state4:=state4∨X
      end;
      Y:=reachl∨reachr; Y := Y∧state3;
      if SOME(Y) then
        begin state2:=state2∨Y; state3:=state3∧¬Y
```

```
        end;
    end;
end
```

It is not difficult to verify that

$$N(\text{MSTP}) \leq 10 + 3\log_2 m + (24 + 3\log_2 m + 8\log_2 n)(n-1).$$

Therefore, we obtain that $N(\text{MST1}) < N(\text{MSTP})$.

The improvement was obtained due to taking into account the special features of associative parallel processors with bit-serial processing. Really, for representing a graph as a set of triples on the STAR-machine at each iteration it is sufficient to store the following two sets (instead of four sets in the Baase algorithm for sequential machines):

(1) positions of edges which have been included in the minimal spanning tree fragment;

(2) positions of edges which are the real candidates for including in the fragment.

7. Conclusion

In this paper we have considered two variants for representing the Prim-Dijkstra algorithm in the STAR-machine. We have written the corresponding procedures, proved their correctness and evaluated their complexity.

Representation of a graph in the form of triples is a natural and simple one and the corresponding procedure (MST1) utilizes the minimal number of statements (Theorem 2). However, for representing the graph edges in the form of vertical pairs we have improved the procedure estimation (for MST2). Moreover, we have obtained that for $n \geq 4$ $N(\text{MST1}) - N(\text{MST2}) < 4\log_2 n(n-2)$.

For representing a graph in the form of triples there is an effective program MST which uses the Baase algorithm and is written by means of the language ASC. For comparing the procedure MST1 and the program MST we have written the STAR procedure MSTP which uses all variable names from the Potter MST program and for the same purposes. For these procedures we have obtained that $N(\text{MST1}) < N(\text{MSTP})$.

References

[1] K.E. Grosspietsch, *Associative processors and memories* In: IEEE, Micro, June, 1992.

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

[3] R.G. Lange, *High level language for associative and parallel computation with Staran*, In: Proc. of Intl. Conf. on Parallel Processing, 1976.

[4] C.C. Foster, *Content Addressable Parallel Processors*, Van Nostrand Reinhold Company, New York, 1976.

[5] J. Miklosko, R. Klette, M. Vajtersic, J. Vrto, *Fast Algorithms and their Implementation on Specialized Parallel Computers*, Special Topics in Supercomputing, Vol. 5, North-Holand, 1989.

[6] A.Sh. Nepomniaschaya, *Language STAR for associative and parallel computation with vertical data processing*, Proc. of the Intern. Conf. "Parallel Computing Technologies", Novosibirsk, USSR, 1991.

[7] R.C. Prim, *Shortest connection networks and some generalizations*, Bell System Tech. J., Vol. 36, 1957.

[8] E.W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Math., Vol. 1, 1959.

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

[10] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, MA, 1978.

[11] A.Sh. Nepomniaschaya, *Investigation of associative search algorithms in vertical processing systems*, Proc. of the Intern. Conf. "Parallel Computing Technologies", Obninsk, Russia, 1993.