Parallel and Distributed Solutions for the Optimal Binary Search Tree Problem

Mitică Craus

"Gh.Asachi" Technical University of Iasi
Computer Engineering Department
6600 Iaşi, Romania
craus@cs.tuiasi.ro

Abstract. A parallel and a distributed implementation for a very important problem in the searching theory, the optimal binary search tree (BST) problem, is presented and analyzed. Implemented as a VLSI array, the algorithm for building the optimal BST uses $O(n^2)$ processors and has the parallel time complexity $O(n)$. A search is solved in $O(\log n)$ time. On a cluster of computers, the binary search tree is organized on two levels: the first level corresponds to the BST of searching intervals and the second level is the level of the BST for effective searching within an interval. A hybrid solution is also considered. The best variant depends on the hypothesis of the searching problem.

1 Distributed Searching Structures and Algorithms

Usually, a wide area distributed database has not a single index of all information. That is the case of a national or regional network of electronic libraries or a distributed internet searching engine.

Which is the most adequate data structure for a distributed index? There are a lot of searching data structure (optimal binary search trees, AVL trees, 2-3 trees, 2-3-4 trees, B-trees, red-black trees, splay trees, digital search trees, tries [1,5,6]) but are they enough for a distributed index? The answer seems to be negative.

The internal relationships of an index usually have the topology of a digraph and this data structure is not adequate for efficient search algorithms. Building spanning tree [3] in a dynamically manner on the digraph representing the key relations in the index may be a way to use the search algorithms for trees. On the other hand, a distributed index imposes the distribution of the digraph over the network.

Also, the search algorithms for distributed search data structures differ from that for locally search data structures.

In a distributed search engine, a single user query starts in one place and the result should be returned to that one place. Centralized control of the searching process will direct where to search next and determine when the search is completed. With a fully distributed search, there is no central control.

D. Grigoras et al. (Eds.): IWCC 2001, LNCS 2326, pp. 103–117, 2002.

Both searching with centralized control and with no central control have advantages and disadvantages.

The main advantage of a centralized search is the control of the searching process but a centralized controller managing the entire process can become a bottleneck.

The advantages of a distributed search is that it offers greater processing power with each remote server performing part of the search, and avoids the bottleneck of a centralized controller managing the entire process. One problem with no central control is that often the same area is searched multiple times. Another problem of distributed searching is uniformity of evaluation. With several search engines each performing their searches using their own criterion for ranking the results, the synthesis of the various results is a difficult problem.

Which is the best solution for searching in a distributed index: centralized or distributed search? This is a problem. The answer seems to be the distributed search but the centralized search has its advantages that cannot be neglected.

Our paper is concerning with the problem of building a distributed search structure adequate to very fast centralized searches. This structure is a distributed binary search tree. It may be used for indexing a wide area distributed database. There are some researches in the indexing information retrieval area [4] but we have no information to be one referring to a distributed index based on binary search tree, organized in such a manner that the keys with the high probability of searching are reached faster then the others.

2 Optimal Binary Search Tree Problem

Let us consider $A = (a_1, a_2, ..., a_n)$ a sequence of data items increasing sorted on their keys $(k_1, k_2, \ldots, k_n)$. With these items it follows to be built a binary search tree [1].

Each item a_i is searched with the probability $p_i, i = 1, 2, \ldots, n$. Let us denote by q_i the probability to search the item x with the property that $a_i < x < a_{i+1}, i = 0, \ldots, n$, where $a_0 = -\infty, a_{n+1} = +\infty$. In these conditions $\sum_{i=1}^{n} p_i$ is the probability of success, $\sum_{i=0}^{n} q_i$ is the fail probability and $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$.

Given $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_0, q_1, q_2, \ldots, q_n)$,the optimal BST is that for which the medium time to solve the search operations has the minim value. In the perspective of medium time computing, the BST is inflated with a set of pseudo-vertices corresponding to the failure intervals. The pseudo-vertices become the new leaves of the BST. A pseudo-vertex e_i will correspond to the search operations for values that belong to the interval (a_i, a_{i+1}) (see Fig. 1).

The time cost for a such tree is the medium time for solving a searching operation i.e:

$$\text{cost}(T) = \sum_{i=1}^{n} p_i * \text{level}(a_i) + \sum_{i=0}^{n} q_i * (\text{level}(e_i) - 1)$$

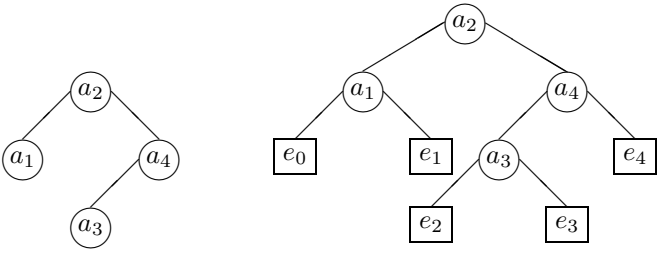

Fig. 1. Inflating a BST with pseudo-vertices e_i

where $\texttt{level}(a_i)$ represents the number of comparisons that are executed when it is searched the item a_i.

The expression $p_i * \text{level}(a_i)$ represents the cost of the searching for the item a_i and $q_i * (\text{level}(e_i) - 1)$ is the cost for a searching in the interval (a_i, a_{i+1}). From $\texttt{level}(e_i)$ the value 1 is subtracted, because the decision that the searched value belong to an interval (a_i, a_{i+1}) is not produced at the level of the pseudo-vertex but at the level of its parent.

For the sequence A, a set of binary search trees can be built. One of them is optimal. Let us denote this tree by T.

From the dynamic programming point of view, the tree T is the result of a sequence of decisions referring the vertex that becomes root of the optimal BST corresponding to a sequence $(a_{i+1}, a_{i+2}, ..., a_j), i \in \{0, 1, \ldots, n-1\}, j \in \{1, 2, \ldots, n\}, i < j$.

Let use denote by $c_{i,j}$ the cost of an optimal BST that has the vertices from the sequence $(a_{i+1}, a_{i+2}, ..., a_j)$ and the pseudo-vertices from the sequence $(e_i, e_{i+1}, ...e_j)$.

Lemma 1. The matrix $C = (c_{i,j})_{i,j=0,...,n}$ can be computed by the following recurrence:

$$c_{i,j} = \min_{i+1 \le k \le j} \{c_{i,k-1} + c_{k,j} + w_{i,j}\} \tag{1}$$

where $w_{i,j} = q_i + \sum_{l=i+1}^{j} (p_l + q_l)$

Proof. Let us consider the sequence $(a_1, a_2, ..., a_r, ..., a_n)$ increased sorted and let us suppose that during the first step the item a_r is selected to become root.

According to the optimality principle, it follows :

The left side subtree L is optimal and it is built with the sequences $(a_1, a_2, ..., a_{r-1})$ and $(e_0, e_1, ..., e_{r-1})$:

$$\text{cost}(L) = \sum_{i=1}^{r-1} p_i * \text{level}(a_i) + \sum_{i=0}^{r-1} q_i * (\text{level}(e_i) - 1)$$

The right side subtree R is optimal and it is built with the sequences $(a_{r+1}, a_{r+2}, ..., a_n)$ and $(e_r, e_{r+1}, ..., e_n)$:

$$\text{cost}(R) = \sum_{i=r+1}^{n} p_i * \text{level}(_i) + \sum_{i=r}^{n} q_i * (\text{level}(e_i) - 1)$$

Obs. The values `level()` are considered in the subtrees L, R.
The cost associated to the tree T becomes:

$$\text{cost}(T) = p_r + \text{cost}(L) + \text{cost}(R) + \sum_{i=1}^{r-1} p_i + \sum_{i=0}^{r-1} q_i + \sum_{i=r+1}^{n} p_i + \sum_{i=r}^{n} q_i.$$

The sums $\sum_{i=1}^{r-1} p_i + \sum_{i=0}^{r-1} q_i + \sum_{i=r+1}^{n} p_i + \sum_{i=r}^{n} q_i$ represent the additional values that appear because the tree T introduces a new level.

$$\text{cost}(T) = p_r + \text{cost}(L) + \text{cost}(R) + q_0 + \sum_{i=1}^{r-1} (p_i + q_i) + q_r + \sum_{i=r+1}^{n} (p_i + q_i)$$

Using the notation $w_{i,j} = q_i + \sum_{l=i+1}^{j} (p_l + q_l)$, it results:

$$\text{cost}(T) = p_r + \text{cost}(L) + \text{cost}(R) + w_{0,r-1} + w_{r,n} = \text{cost}(L) + \text{cost}(R) + w_{0,n}$$

Following the definition of $c_{i,j}$ we have:

$$\text{cost}(T) = c_{0,n},\ \text{cost}(L) = c_{0,r-1} \text{ and } \text{cost}(R) = c_{r,n}$$

The above relation becomes

$$c_{0,n} = c_{0,r-1} + c_{r,n} + w_{0,n} = \min_{1 \le k \le n} \{c_{0,k-1} + c_{k,n} + w_{0,n}\}$$

Generalizing, it results the recurrence (1).
Obs. $c_{i,i} = 0$.

As the Lemma 1 states, the recurrence (1) is used to compute the matrix of the values of optimums that, at its turn, will be the base for building the optimal BST.

Theorem 1. The binary search tree built following the relation (1) is optimal.

Proof. The relation (1) assures that $c_{0,n}$ is minimum over all costs of the binary search trees built from the sequence $(a_1, a_2, ..., a_n)$. Thus, the binary

search tree built following the relation (1) has minim cost, consequently it is optimal.

After a BST is built other two important problems are the following: searching for a value and updating the tree.

The searching algorithms depend of the BST implementation. The implementation on a single sequential computer involves searching algorithms different of that corresponding to the implementations on VLSI arrays or clusters of computers.

Also, the updating problem is different for sequential computers, VLSI arrays, and cluster of computers. Often, it is necessary to rebuild the entire tree. This is an important reason for designing efficient algorithms for the building of optimal BST's. The parallel or distributed algorithms should be an alternative.

A distributed implementation of a BST is justified when the necessary space for storing the tree exceeds the memory resources of the host. This situation may appear in database of astronomical dimension (library, search engine).

3 VLSI Implementation of the Optimal Binary Search Tree Problem

3.1 A Technique to Implement Uniform Recurrences into VLSI Arrays

For an uniform recurrence there exists some techniques to implement it into a VLSI architecture. One of them is given by C. Guerra in [2] and can be resumed as it follows:

Let us consider an uniform recurrence given by the following relation:

$$c(\bar{i}) = f(c(\bar{i} - \bar{d}_1), ..., c(\bar{i} - \bar{d}_s)) \tag{2}$$

where: f is a given function, $\bar{i}$ is a vector from a index set I^n and $\bar{d}_1, ..., \bar{d}_s$ are constant vectors that belong to the set I^n and define the data dependence ($[\bar{d}_1, \ldots, \bar{d}_s]$=$D$).

We consider now a VLSI architecture [7] defined as follows: each module is assigned to a label $l \in L^{n-1} \subset \mathbf{Z}^{n-1}$; the communication model is described by the matrix $\Delta = [\delta_1, ..., \delta_s]$, where δ_i is the vectorial label difference of the adjacent modules.

A linear time-space transformation is defined by the pair

$$\Pi = \begin{bmatrix} T \\ S \end{bmatrix} \tag{3}$$

where: $T : I^n \rightarrow \mathbf{Z}$, is the *timing* function and $S : I^n \rightarrow L^{n-1}$ is the *space* function.

The associated transformation matrices are also denoted by T and S.

To have a correct execution order of the operations, T has to satisfy the following conditions:

$$T(\bar{d}_i) > 0, \forall \bar{d}_i \in D \tag{4}$$

The system (4) may have k solutions ($k \geq 0$). If $k \geq 2$ than it is selected the solution that minimizes the global execution time.

The function S has to satisfy the conditions (5):

$$S(\bar{i}) = S(\bar{j}) \Rightarrow T(\bar{i}) \neq T(\bar{j}), \forall \bar{i}, \bar{j} \in I^n \tag{5}$$

This means that different computations cannot be executed at the same time in the same place (module).

3.2 The Conversion of the Optimal BST Non-uniform Recurrence into a System of Two Uniform Recurrences

The recurrence (1) is not uniform. This handicap may be surpassed. Using again Guerra's methodology, the recurrence (1) can be converted into a system of two uniform recurrence. We will follow step by step the technique used by Guerra for a recurrence of type $c_{i,j} = \min_{i<k<j}\{f(c_{i,k}, c_{k,j})\}$ and we will adapt it to the recurrence (1).

The relation (1) defines an index set $I^3 = \{(i,j,k); 0 \leq i < j \leq n, i < k \leq j\}$ corresponding to the minimization operation and an index set $I^2 = \{(i,j); 0 \leq i < j \leq n\}$ of index pairs associated to the variable c. This variable appearing on both sides of the relation (1) introduces non-constant data dependencies. A dependence matrix is defined as the difference of the index vectors of the variable c on the left and right side of any assign statement.

So, the dependence matrix

$$D_{ij} = \left[\begin{bmatrix} i \\ j \end{bmatrix} - \begin{bmatrix} i \\ k-1 \end{bmatrix},\ k = i+1, \ldots, j \quad \begin{bmatrix} i \\ j \end{bmatrix} - \begin{bmatrix} k \\ j \end{bmatrix},\ k = i+1, \ldots, j \right]$$

looks as follows:

$$D_{ij} = \begin{bmatrix} 0 & \ldots & 0 & 0 & -1 & -2 & \ldots & i-j \\ j-i & \ldots & 2 & 1 & 0 & 0 & \ldots & 0 \end{bmatrix}$$

The first step of the process of converting the non-uniform recurrence (1) in a uniform one consists of adding the absent indices to variables on the left or right side of the relation (1). For adding the index k to the variable c_{ij} on the left side of (1) it is necessary to specify an appropriate ordering for the computations corresponding to the vector (i,j,k) in order to introduce as much parallelism as possible. Guerra's strategy is to identify among the computations indexed by the set I^3 chains of linearly dependent computations, i.e. computations that have to be performed in a certain order. To do this, from (1) is extracted a subset C of assignments corresponding to constant dependencies. A linear time function T is derived only for such subset. If τ is the actual timing function for the set I^2, than it has to satisfy the conditions $\tau(i,j) \geq T(i,j), \forall (i,j) \in I^2$. The set C is characterized by the matrix D resulted from the intersection of the matrices $D_{ij}, \forall (i,j) \in I^2$:

$$D = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

The linear time transformation $T : I^2 \to \mathbf{Z}$ has to satisfy the condition (4). That is:

$$TD = [\,T_1 \quad T_2\,] \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} > [\,0 \quad 0\,]$$

It results $T_2 > 0$ and $T_1 \leq -1$.

The optimal values that satisfy the above equations are $T_2 = 1$ şi $T_1 = -1$.

Thus the function T is defined by:

$$T(i,j) = j - i$$

The function T will be uses to find a schedule of the computations indexed by I^3 according with the availability of the variable $c_{i,k-1}$ and $c_{k,j}$ on the right side of (1). The function τ has to satisfy $\tau(i,j) > \tau(i',j')$ if $T(i,j) > T(i',j')$). Thus, a partial ordering "$>_T$" on I^3 can be defined as follows:

$$(i,j,k') >_T (i,j,k'') \Leftrightarrow \max\{T(i,k'-1), T(k',j)\} > \max\{T(i,k''-1), T(k'',j)\}$$

The minimal elements with respect to "$>_T$" are:

$$\begin{cases} (i,j,(i+j+1)/2), & \text{if } i+j \text{ is odd} \\ (i,j,(i+j)/2) \text{ and } (i,j,(i+j)/2+1), & \text{if } i+j \text{ is even} \end{cases}$$

The partial ordering "$<_T$" produces decompositions of the I^3 into chains that correspond to linearly ordered computations. Among all the possible decompositions, that for which the chains are sorted according to the index k is of interest. It results a decomposition in two such chains as it follows:

$$\begin{cases} (i,j,(i+j+1)/2), (i,j,(i+j+1)/2-1), ..., (i,j,i+1) \\ (i,j,(i+j+1)/2), (i,j,(i+j+1)/2+1), ..., (i,j,j) \end{cases} \text{if } i+j \text{ is odd}$$

and

$$\begin{cases} ((i,j,(i+j)/2), (i,j,(i+j)/2-1), ..., (i,j,i+1) \\ (i,j,(i+j)/2+1), (i,j,(i+j)/2+2), ..., (i,j,j) \end{cases} \text{if } i+j \text{ is even}$$

Now, the recurrence (1) can be restructured into a system of two uniform recurrences, each corresponding to a chain. The order of the computations in each uniform recurrence is given by the ordering defined on the chain. These computations are described by the following algorithm:

for $i = 0$ **to** n **do**
  $c_{i,i} \leftarrow 0;\ w_{i,i} \leftarrow q_i;$
end for
for $i = 0$ **to** n **do**
  $a"_{i,i,i} \leftarrow c_{i,i};\ c_{i,i,i} \leftarrow c_{i,i};$
end for

for $l = 1$ **to** n **do**
  for $i = 0$ **to** $n - l$ **do**
    $j \leftarrow i + l$;
    `/* INIT LOOP-1 and LOOP-2 */`
    if $(i + j) = odd$ **then**
      $k \leftarrow (i + j + 1)/2$;
      `/*A1*/`
      $a'_{i,j,k-1} \leftarrow a"_{i,j-1,k-1}$;
      if $k = i + 1$ **then**
        `/*A2*/`
        $b'_{i,j,k} \leftarrow c_{i+1,j,j}$
      else
        $b'_{i,j,k} \leftarrow b'_{i+1,j,k}$;
      end if
      $c'_{i,j,k-1} \leftarrow a'_{i,j,k-1} + b'_{i,j,k}$; $c"_{i,j,k} \leftarrow c'_{i,j,k-1}$;
    else
      $k \leftarrow (i + j)/2$;
      $a'_{i,j,k-1} \leftarrow a'_{i,j-1,k-1}$;
      if $k = i + 1$ **then**
        $b'_{i,j,k} \leftarrow c_{i+1,j,j}$
      else
        $b'_{i,j,k} \leftarrow b'_{i+1,j,k}$;
      end if
      $c'_{i,j,k-1} \leftarrow a'_{i,j,k-1} +' b_{i,j,k}$;
      $k \leftarrow (i + j)/2 + 1$;
      if $k = j$ **then**
        `/*A3*/`
        $a"_{i,j,k-1} \leftarrow c_{i,j-1,j-1}$
      else
        $a"_{i,j,k-1} \leftarrow a"_{i,j-1,k-1}$;
      end if
      `/*A4*/`
      $b"_{i,j,k} \leftarrow b'_{i+1,j,k}$;
      $c"_{i,j,k} \leftarrow a"_{i,j,k-1} + b"_{i,j,k}$;
    end if
    `/*LOOP-1*/`
    for $k = \text{round}((i + j)/2) - 1$ **downto** $i + 1$ **do**
      $a'_{i,j,k-1} \leftarrow a'_{i,j-1,k-1}$;
      if $k = i + 1$ **then**
        $b'_{i,j,i+1} \leftarrow c_{i+1,j,j}$
      else
        $b'_{i,j,k} \leftarrow b'_{i+1,j,k}$;
      end if
      $c'_{i,j,k-1} \leftarrow \min(c'_{i,j,k}, a'_{i,j,k-1} + b'_{i,j,k})$;
    end for

/*LOOP-2*/
for $k = \text{trunc}((i+j)/2) + 2$ **to** j **do**
if $k = j$ **then**
$a"_{i,j,k-1} \leftarrow c_{i,j-1,j-1}$
else
$a"_{i,j,k-1} \leftarrow a"_{i,j-1,k-1}$;
end if
$b"_{i,j,k} \leftarrow b"_{i+1,j,k}$;
$c"_{i,j,k} \leftarrow \min(c"_{i,j,k-1}, a"_{i,j,k-1} + b"_{i,j,k})$;
end for
$w_{i,j} \leftarrow w_{i,j-1} + p_j + q_j$;
/*A5*/
$c_{i,j,j} \leftarrow \min(c'_{i,j,i}, c"_{i,j,j}) + w_{i,j}$;
$c_{i,j} \leftarrow c_{i,j,j}$;
end for
end for

3.3 Mapping the Optimal BST Uniform Recurrences into a VLSI Array

After the transformation of the recurrence (1) in two uniform recurrences, we can apply Guerra's technique to map it into hardware.

Timing Function: Using the above new specification of the recurrence (1), from LOOP-1 and LOOP2 it will be extracted the matrices of constant data dependencies D_1 and D_2, respectively:

$$D_1 = \begin{array}{c} \begin{matrix} c' & a' & b' \end{matrix} \\ \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \end{array} \quad D_2 = \begin{array}{c} \begin{matrix} c'' & a'' & b'' \end{matrix} \\ \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

The dependencies between variables in LOOP-1 and LOOP-2 (global dependencies) are defined be the assignments A1 - A5 .

The goal is to find a linear time transformation for each set of local data dependencies so that they satisfy the global dependencies. Let be $\lambda = [\lambda_1 \quad \lambda_2 \quad \lambda_3]$, $\sigma = [\sigma_1 \quad \sigma_2 \quad \sigma_3]$ and $\tau = [\tau_1 \quad \tau_2 \quad \tau_3]$ the transformation vectors corresponding to the LOOP-1 , LOOP-2 and the assignment A5, respectively.

From (4) it results:

$$\lambda \bar{d} > 0 \text{ pentru } \bar{d} \in D_1; \quad \sigma \bar{d} > 0 \text{ for } \bar{d} \in D_2$$

This means:

$$\lambda_1 \leq -1 \quad \lambda_2 \geq 1 \quad \lambda_3 \leq -1$$

$$\sigma_1 \leq -1 \quad \sigma_2 \geq 1 \quad \sigma_3 \geq 1$$

The global dependencies A1-A5 add the following

$$\lambda\,[\,i \quad j \quad (i+j+1)/2-1\,]^t > \sigma\,[\,i \quad j-1 \quad (i+j+1)/2-1\,]^t$$

$$\lambda\,[\,i \quad j \quad i+1\,]^t > \tau\,[\,i+1 \quad j \quad j\,]^t$$

$$\sigma\,[\,i \quad j \quad j-1\,]^t > \tau\,[\,i \quad j-1 \quad j-1\,]^t$$

$$\sigma\,[\,i \quad j \quad (i+j)/2+1\,]^t > \lambda\,[\,i+1 \quad j \quad (i+j)/2+1\,]^t$$

$$\tau\,[\,i \quad j \quad j\,]^t \geq \max\{\lambda\,[\,i \quad j \quad i\,]^t, \sigma\,[\,i \quad j \quad j)\,]\}$$

It results for the timing function the following transformation matrix:

$$\begin{bmatrix}\lambda\\ \sigma\\ \tau\end{bmatrix} = \begin{bmatrix}\lambda_1 & \lambda_2 & \lambda_3\\ \sigma_1 & \sigma_2 & \sigma_3\\ \tau_1 & \tau_2 & \tau_3\end{bmatrix} = \begin{bmatrix}-1 & 2 & -1\\ -2 & 1 & 1\\ -2 & 1 & 1\end{bmatrix}$$

Thus,

$$\lambda\,[\,i \quad j \quad k\,]^t = -i+2j-k,\ \sigma\,[\,i \quad j \quad k\,]^t = -2i+j+k \text{ and } \tau\,[\,i \quad j \quad j\,]^t = -2i+2j.$$

Space Function: Let us consider an architecture Mesh-Connected (Fig.2) given by the pair $[L^2, \Delta]$, where L^2 is the set of labels (x, y) assigned to the computing modules (the processors) and Δ is a matrix that defines the array interconnections:

$$\Delta = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & -1\end{bmatrix}$$

Let us denote by S', S'' and S the space functions corresponding to the computations from the `LOOP-1, LOOP-2` and the assignment `A5`, respectively. The associated transformation matrices will be:

$$S' = \begin{bmatrix}S'_{11} & S'_{12} & S'_{13}\\ S'_{21} & S'_{22} & S'_{23}\end{bmatrix} \text{ for the LOOP} - 1$$

$$S'' = \begin{bmatrix}S''_{11} & S''_{12} & S''_{13}\\ S''_{21} & S''_{22} & S''_{23}\end{bmatrix} \text{ for the LOOP} - 2$$

$$S = \begin{bmatrix}S_{11} & S_{12} & S_{13}\\ S_{21} & S_{22} & S_{23}\end{bmatrix} \text{ for the assignment A5}$$

The coefficients of the matrices S', S'' and S have to respect the condition (5) and the restrictions imposed by the global dependencies. This means the following: if two variables belonging to different modules are processed at the moment t and respectively t' with $t - t' = d$, then the distance between this two modules cannot be greater than d. By the distance it is understood the number of wires that link the two modules.

Thus, from $A1$ is results:

$$S'\,[\,i \quad j \quad (i+j+1)/2-1\,]^t = S''\,[\,i \quad j-1 \quad (i+j+1)2-1\,]^t + \bar{d}_1; \qquad \bar{d}_1 \in \Delta$$

This because $\lambda\,[\,i \quad j \quad (i+j)/2\,]^t - \sigma\,[\,i \quad j-1 \quad (i+j)2\,]^t = 1$ imply execution of the two computations $S'\,[\,i \quad j \quad (i+j)/2\,]^t$ and $S''\,[\,i \quad j-1 \quad (i+j)2\,]^t + \bar{d}_1)$, in the same processor or in adjacent processor (connected by a directed link).

Similarly, from $A2 - A4$ it is obtained:

$$S'\,[\,i \quad j \quad i+1\,]^t = S\,[\,i+1 \quad j \quad j\,]^t + \bar{d}_2; \quad \bar{d}_2 \in \Delta$$
$$S''\,[\,i \quad j \quad j-1\,]^t = S\,[\,i \quad j-1 \quad j-1\,]^t + \bar{d}_3; \quad \bar{d}_3 \in \Delta$$
$$S''\,[\,i \quad j \quad (i+j)/2+1\,]^t = S'\,[\,i+1 \quad j \quad (i+j)2+1\,]^t + \bar{d}_4;$$
$$\bar{d}_4 = \delta_i + \delta_j \quad \delta_i, \delta_j \in \Delta$$
$$S\,[\,i \quad j \quad j\,]^t = S\,[\,i \quad j \quad i\,]^t + \bar{d}_5; \quad \bar{d}_5 \in \Delta$$

A solution for the system of equations resulted from the application of the restrictions imposed by global dependencies, is the following:

$$S' = \begin{bmatrix} S'_{11} & S'_{12} & S'_{13} \\ S'_{21} & S'_{22} & S'_{23} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$
$$S'' = \begin{bmatrix} S''_{11} & S''_{12} & S''_{13} \\ S''_{21} & S''_{22} & S''_{23} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$
$$S = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

It results:

$$S'\,[\,i \quad j \quad k\,]^t = S''\,[\,i \quad j \quad k\,]^t = S\,[\,i \quad j \quad j\,]^t = [\,j \quad i\,]^t$$

This means that all the computations for a variable $c_{i,j}$ in the recurrence (1) are located in the processor labelled $(x, y) = (j, i)$

From

$$S'D_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{matrix} c' & a' & b' \\ \end{matrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

it is derived that $c'_{i,j,k}$ do not move along the array , $a'_{i,j,k-1}$ move to the right ($\rightarrow$) and $b'_{i,j,k}$ move up ($\uparrow$).

Analogously,

$$S''D_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{matrix} c'' & a'' & b'' \\ \end{matrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

implies that $c''_{i,j,k}$ do not move along the array , $a''_{i,j,k-1}$ move to the right ($\rightarrow$) and $b''_{i,j,k}$ move up ($\uparrow$).

In the figure 2, the VLSI array that implements the optimal BST problem is depicted.

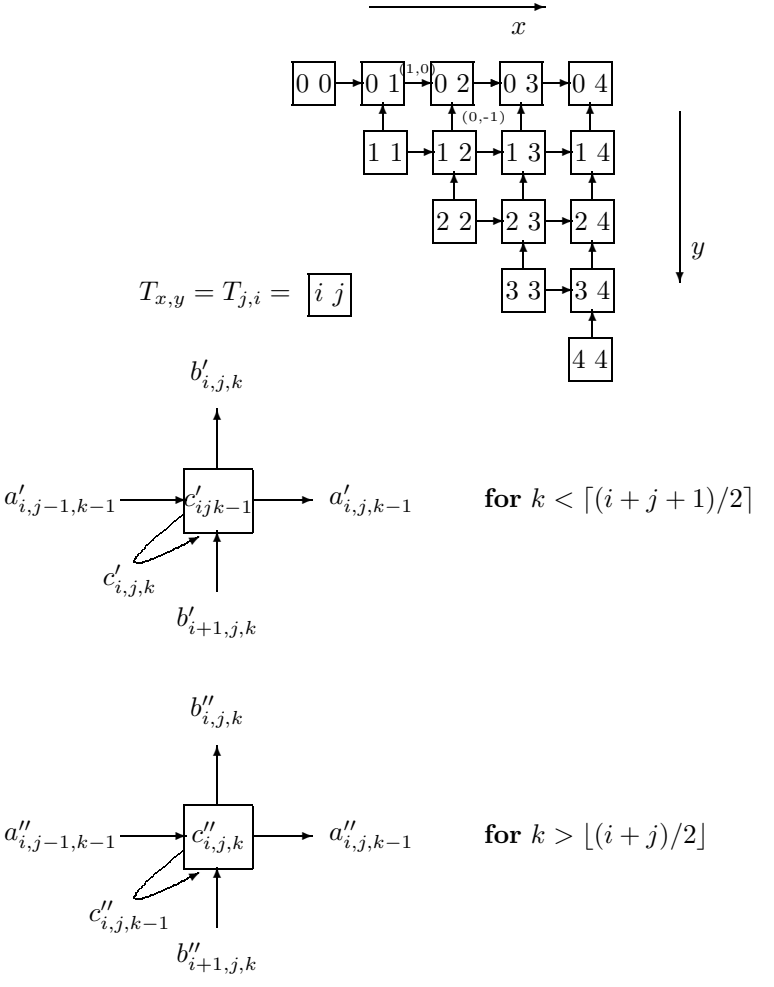

Fig. 2. The VLSI array for the optimal BST problem

4 Distributed Implementation of a BST on a Cluster of Computers

The main idea for a distributed implementation of a BST corresponding to n searching keys on a cluster of m computers is the following:

1. An optimal BST corresponding to a set of m intervals is built; each interval contains approximately the same number of searching keys;

2. For each interval, a computer in the cluster is assigned and the corresponding optimal BST is built.

4.1 The Optimal BST Corresponding to a Set of m Intervals

This step implies the creation of a partition $\mathcal{P}$ of the real numbers set $\mathbb{R}$. The member intervals are unions of subsets of keys (vertices) and key-intervals (pseudo-vertices) that contain approximately the same number of searching keys. Let us consider $\mathcal{P} = (I_1, I_2, ..., I_m)$

An interval $I_i \neq I_m$ is defined as follows:

$$I_i = e_l \cup \{a_{l+1}\} \cup e_{l+1} \cup \{a_{l+2}\} \ldots \cup e_{h-1} \cup \{a_h\}$$

$$l \in \{0, 1, \ldots, n-1\}, h \in \{1, 2, \ldots, n\}, l < h, \quad i \in \{1, \ldots, m-1\}$$

The probability of searching for a value from I_i is

$$P(I_i) = q_l + p_{l+1} + q_{l+1} + p_{l+2} \ldots + q_{h-1} + p_h$$

The last interval of the partition will be I_m

$$I_m = e_l \cup \{a_{l+1}\} \cup e_{l+1} \cup \{a_{l+2}\} \ldots \cup e_{n-1} \cup \{a_n\} \cup e_n, \quad l \in \{0, 1, \ldots, n-1\}$$

Its associated probability will be:

$$P(I_m) = q_l + p_{l+1} + q_{l+1} + p_{l+2} \ldots + q_{n-1} + p_n + q_n$$

Obs. The pseudo-vertex e_i was used in the above relations instead of the interval (a_i, a_{i+1}).

The relation $I_i \preceq I_j \Leftrightarrow i \leq j, \quad i, j \in \{1, \ldots, m\}$ is an order on $\mathcal{P}$.

With the items $I_1, I_2, ..., I_m$, a binary search tree $T_{\mathcal{P}}$ will be built. Each item I_i is searched with the probability $P(I_i)$. Unsuccessful searches are not possible.

The time cost for a such tree is the medium time for solving a searching operation. The searching for the value x means to find the interval I_i that contains x.

$$\text{cost}(T_{\mathcal{P}}) = \sum_{i=1}^{m} P(I_i) * \text{level}(I_i)$$

Let use denote by $C_{i,j}$ the cost of an optimal BST that has the vertices from the sequence $(I_i, I_{i+1}, ..., I_j)$.

Using a similar technique as the one used for obtain the recurrence (1), it results:

$$C_{i,j} = \min_{i<k<j} \{C_{i,k-1} + C_{k+1,j} + W_{i,j}\} \tag{6}$$

where $W_{i,j} = \sum_{l=i}^{j} P(I_l)$.

Theorem 2. The binary search tree built from the relation (6) is optimal.

Proof. The relation (6) states that $C_{1,n}$ is minim over all costs of the binary search trees built from the sequence $(I_1, I_2, ..., I_m)$. Thus, the binary search tree built according to the relation (6) has minim cost. This proves its optimality.

It results that the problem of building the tree $T_{\mathcal{P}}$ does not differ in essence from the ordinal problem: the building of an optimal BST for a sequence $A = (a_1, a_2, ..., a_n)$ of data items increasing sorted on their keys $(k_1, k_2, \ldots, k_n)$.

4.2 The Optimal BST Corresponding to an Interval of Keys

For each interval I, it will be build an optimal BST. Let it be T_I. The corresponding algorithm will use the same strategy as in the case of the building the tree T for the sequence $A = (a_1, a_2, ..., a_n)$. In a pre-processing step, the probabilities will be inflated to have their sum equal to 1.

4.3 The Distribution of the Trees to the Cluster Members

The computer that will compute and store $T_{\mathcal{P}}$ depends of the cluster topology. In a cluster organized as a ring of computers may be useful to have the tree $T_{\mathcal{P}}$ inside of every cluster member. In the case of a star topology it is enough to store the tree $T_{\mathcal{P}}$ in the central computer.

4.4 The Searching for a Value x

A search for value x will be executed as follows:

First, the interval I that contains x will be identified . This means a search inside of the interval-tree $T_{\mathcal{P}}$.

In the next step, the value x will be searched in the key-tree T_I corresponding to the interval I.

5 Hybrid Implementation

In a cluster of computers that can hosts VLSI arrays, the distributed and VLSI implementation can be combined in order to obtain a better performance for the problem of building and rebuilding of a wide binary search tree. This means that the VLSI implementation of the optimal BST problem may be used at the level of the cluster members.

6 Conclusion

The Optimal Binary Search Tree problem has a lot of solutions and a lot of open problems. One of them is the optimal distributed implementation.

The VLSI array implementation is optimal but for large number of keys it appears the problem of the hardware excessive cost. A solution may be to use processes instead of processors and to use the same processor to solve bigger tasks than "min" operations.

Our solution for the distributed BST implementation is not optimal but it is acceptable efficient. Clusters are difficult because of the cost of communications. The balancing of the cost of the communications inside the cluster with the cost of the local searching in the cluster members may be a way to solve the problem of the optimal distributed implementation for a BST. This is an open problem.

For an astronomical index, a cluster implementation seems to be better than external memory storage. Obvious, the internal memory search is faster that the searching on an external memory but sometime the cost of the communication between the cluster members reduces this advantage. Our solution of finding relatively fast, by means of the interval tree, the cluster member which contains the part of the tree that can answer to the search query, reduces enough the cost of the communications to make attractive the solution of distributing the BST into a cluster of computers.

References

1. Horowitz, E., Sahni, S., Anderson-Freed, S.: Fundamentals of Data Structures in C, Computer Science Press, New York (1993)
2. Guerra, C.: A Unifying Framework for Systolic Design, LNCS 227, Springer-Verlag, Berlin Heidelberg New York (1986) 46–56
3. H. Attiya, Welch, H.J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, McGraw-Hill, London (1998)
4. Rogers, W., Candela, G., Harman, D.: Space an Time Improvements for Indexing in Information Retrieval, National Institute of Standards & Technology, Gaithesburg (2001)
5. Baase, S., Van Gelder A.: Computer Algorithms: Introduction to Design & Analysis, Addison Wesley Longman (2000)
6. Heileman, G.: Data Structures, Algorithms and Object-Oriented programming, McGraw-Hill, New York (1996)
7. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kauffman (1992)