Parallel Algorithms for Routing in Non-blocking Networks

by

Geng Lin

Nicholas Pippenger

Technical Report 91-8
June, 1991

Department of Computer Science
University of British Columbia
Vancouver, B.C.
CANADA V6T 1Z2

Parallel Algorithms for Routing in Non-blocking Networks

Geng Lin
Nicholas Pippenger

Department of Computer Science
The University of British Columbia
Vancouver, British Columbia V6T 1W5
CANADA

Abstract Non-blocking networks have many applications in communications. Typical examples are telephone switching networks and communication networks among processors or between processors and memory devices. We construct non-blocking networks that are efficient not only as regards their cost and delay, but also as regards the time and space required to control them. In this paper, we present the first simultaneous "weakly optimal" solutions for the explicit construction of non-blocking networks, the design of algorithms and the design of data-structures. "Weakly optimal" is in the sense that all measures of complexity (size and depth of the network, time for the algorithm, and space for the data-structure) are within one or more factors of $logn$ of their smallest possible values. In fact, we explicitly construct a scheme in which networks with n inputs and n outputs have size $O(n(logn)^2)$ and depth $O(logn)$. And we present deterministic and randomized on-line parallel algorithms to establish and abolish routes dynamically in these networks. The deterministic algorithm uses $O((logn)^5)$ steps to process any number of transactions in parallel (with one processor per transaction), maintaining a data structure that use $O(n(logn)^2)$ words and the randomized algorithm uses $O((logn)^2)$ expected steps to process any number of transactions in parallel (with one processor per transaction), maintaining a data structure that use $O(n(logn)^2)$ words.

1. Introduction

Non-blocking networks have many applications in communications. Typical examples are telephone switching networks and communication networks among processors or between processors and memory devices. Given an acyclic directed graph with a set of distinguished vertices called *inputs* and a set of other distinguished vertices called *outputs*, it is said to be a "*non-blocking*" network if, given any set of disjoint direct routes from inputs to outputs, and given any input and output not involved in these established routes, a new route that is disjoint from the established routes can be found from the requesting input to the requesting output. Interpretations of the above network in the context of telephone switching and processor communication are clear. The most frequently applied measures of complexity for non-blocking networks are the "size" (the number of single-pole single-throw switches, i.e. the number of edges) and the "depth" (the largest number of switches, i.e. edges, on any route from an input to an output). An extensive literature exists concerning the design of non-blocking networks, minimizing the size and depth (or some combination of them) as functions of the number of inputs and outputs; see Pippenger [P82] for an introductory account, and Feldman, Friedman and Pippenger [FFP88] for recent results. The most basic results are that, if a non-blocking network has n inputs and an equal number of outputs, it must have depth at least 1 (but to achieve this requires size n^2, one switch between each input and each output); it must have size at least $\Omega(nlogn)$ (but to achieve this requires depth at least $\Omega(logn)$; see Pippenger and Yao [PY]).

In this paper we combine this concern for depth and size with concern for the time taken by an algorithm that finds the routes guaranteed by the non-blocking property, and for the space taken by the data-structure used by the algorithm. Unlike the case of depth and size alone, not much progress has been made in this setting. An exception is that Arora, Leighton and Maggs [ALM] found an on-line $O(logn)$ steps parallel path selection algorithm for non-blocking networks of size $O(nlogn)$ and of depth $O(logn)$. Their proposal, however, assumes that the number of processors is proportional to the size of the network, irrespective of the number of transactions being processed. Our approach, in contrast, uses only one processor for each transaction, even if this number is as small as one.

In this paper, we explicitly construct a scheme in which non-blocking networks with n inputs and n outputs have size $O(n(logn)^2)$ and depth $O(logn)$. And we present on-line parallel algorithms to control them. The algorithms use time and space within one or more factors of $logn$ of the smallest possible values that any control algorithm (on-line or off-line, parallel or serial) may use. More precisely, we present an on-line deterministic algorithm that uses $O((logn)^5)$ steps to process any number of transactions in parallel (with one processor per transaction), maintaining a data structure that use $O(n(logn)^2)$ words and an on-line randomized algorithm that uses $O((logn)^2)$ expected steps to process any number of transactions in parallel (with one processor per transaction), maintaining a data structure that use $O(n(logn)^2)$ words. (The meanings of "*step*", "*word*" and "*transaction*" will be explained in next paragraph).

Consider a non-blocking network with $n = 2^\nu$ inputs and $n = 2^\nu$ outputs. We assume that inputs and outputs are represented as binary words of length ν and a "processor" is able to perform arithmetic and logical operations on such words of length ν. We reckon "time" in such operations, and "space" in such words. We mainly consider the parallel algorithms and their data-structures. In fact, we consider the algorithm and data-structure together as an "on-line transaction processing system", in which each "batch" of transactions (requests to establish a route and requests to abolish a route) must be processed before its successors are known. Furthermore, for a batch of t transactions, which are to be processed in parallel, only t processors are allowed to be used. In other words, our approach assumes each transaction "brings its own processor", a setting convenient in situations where routing is but one part of a larger process, and the number of processes simultaneously engaged in routing is not easily predictable.

It is observed that if a non-blocking network has n inputs and an equal number of outputs, any algorithm that controls the network must use $\Omega(1)$ steps to process a batch of transactions, and the data-structure for it must have $\Omega(n)$ words (or their equivalent), since this much space is needed to represent one of $n!$ bijection between inputs and outputs.

Our results provide the first simultaneous "weakly optimal" solutions for the explicit construction of non-blocking networks, the design of algorithms and the design of data-structures. "Weakly optimal" is in the sense that all measures of complexity (size and depth of the network, time for the algorithm, and space for the data-structure) are within one or more factors of *logn* of their smallest possible values. Our results are very practical in the sense that the construction of the networks is simple and the algorithms (both the randomized and the deterministic one) and their data-structures are easy to implement. We are optimistic that the results presented in this paper will find many applications in practice.

Our main result in this paper is summarized in the following theorem.

Theorem *There is an explicit construction for a non-blocking of n inputs and n outputs* *with size $O(n(logn)^2)$ and depth $O(logn)$, and a deterministic on-line parallel algorithm that maintains a data-structure using $O(n(logn)^2)$ words and will, for any t in the range $1 \leq t \leq n$, process t transactions using t processors in $O((logn)^5)$ steps.*

2. The Non-Blocking Networks

Suppose that we wish to construct a non-blocking network with n inputs and n outputs. Set $\gamma = \lfloor log_2(8\nu) \rfloor$, so that $2^\gamma > 4\nu \geq 2^{\gamma-1}$. Construct a Beneš rearrangeable network with $m = 2^{\nu+\gamma}$ inputs and m outputs (see Beneš [B]). Reduce the number of inputs and outputs in this network to n by retaining only every 2^γ-th input and output and discard links and switching elements that can not be reached from these n inputs and n outputs. This is to be done so that routes originating at two distinct retained inputs can meet only after passing at least $\gamma + 1$ stages of switches, and similarly for retained outputs.

Let N^+ denote the resulting network. This network is non-blocking as shown in [P82]. Indeed, consider any idle input of N^+. It has access to at least $(2^\gamma - \nu)2^\nu$ of the $2^{\nu+\gamma}$ links of the $(\gamma + \nu + 1)$-st stage regardless the status of other inputs. Similar property holds for any idle outputs. Notice that $(2^\gamma - \nu)2^\nu$ is strictly more than half (to be exact, three quarters) of $2^{\gamma+\nu}$. Thus given any idle input and idle output, a route from the input to the output that is disjoint with the established routes always exists regardless the status of other inputs and outputs. This network has size $O(\nu 2^{\nu+\gamma}) = O(\nu^2 2^\nu) = O(n(logn)^2)$ and depth $O(\gamma + \nu) = O(logn)$. This network is essentially equivalent to the one described by Cantor [C71].

3. A Randomized Algorithm

In this section, we describe a randomized parallel algorithm which processes a batch of transactions in parallel with $O((logn)^2)$ expected steps by dynamically changing a data-structure of $O(n(logn)^2)$ words. Our deterministic parallel algorithm is obtained by eliminating the randomness from this algorithm.

The data-structure for our randomized algorithm is very simple. It only keeps the up-to-date busy/idle status for each link, input and output of the network (this is necessary for any data-structure). We describe the data-structure in terms of the dual graph G of N^+. For each input, output and link of network N^+, we create a node. Two nodes are adjacent if and only if their correspondents in the network are input and output of a same crossbar. Thus we see the duality between G and N^+. With each node ζ in G, we associate a number $G(\zeta)$, which is 0 or 1 according as its corresponding link (more precisely, link or input or output) is idle or busy. A simple calculation shows this data-structure uses $O(n(logn)^2)$ words.

Notice that N^+ has $2(\gamma+\nu)+1$ stages. The subnetwork of stage $\gamma+\nu+1$ to stage $2(\gamma+\nu)+1$ is a mirror image of the subnetwork of stage 1 to stage $\gamma+\nu+1$. We refer the former to "the right hand half of N^+" called N' and the latter "the left hand half of N^+" called N. We observe that, for each input ξ of N^+, confined to N, the dual subgraph in G of links that may appear in some route starting from ξ forms a tree T_ξ. Similarly, for each output η of N^+, there is a tree T_η. It is clear that all the T_ξ's and T_η's are binary trees having depth $\gamma+\nu$ with $2^{\gamma+\nu}$ leaves. We call the common topological structure of these trees T.

We now proceed to describe our randomized algorithm. Suppose that a batch of t transactions (requests to establish or abolish a route) are to be processed by t processors (each transaction "brings its own processor"). We need not worry about interference between requests that establish routes and requests that abolish routes by the simple device of splitting each batch into two batches, one comprising only requests to establish and the other comprising only requests to abolish. As we will see, the algorithms presented in this paper to process requests to establish are easily modified to process requests to abolish, we only consider the requests to establish here.

Suppose that when a processor attempts to establish a route from input ξ to output η, it pushes two pebbles in T_ξ and T_η respectively, from their roots to a common leaf along a path P, and then determines whether or not the two subroutes (in N and N' respectively) corresponding to the subpaths in T_ξ and T_η are both idle, and whether or not no other processor has seized a link in the two subroutes. If we choose one of the $2^{\gamma+\nu}$ possible paths at random, the probability that the the corresponding subroute in N is busy is at most $1/4$, since every idle input in N has access to at least $(2^\gamma-\nu)2^\nu$ of its $2^{\gamma+\nu}$ outputs regardless the status of other inputs, noticing $4\nu < 2^\gamma$. Similarly, the probability that the subroute in N' is busy is at most $1/4$ too. Thus the probability that both subparts of the route are idle is at least $1/2$. That is to say, with randomly chosen paths, half of the requests are expected to be satisfied. For those failing to choose an idle path, do the same procedure again, and so forth. It is observed that, less than or equal to $t2^{-i}$ requests are expected not to be satisfied after i-th round of choosing. Thus after $\lceil log_2(t/\epsilon)\rceil$ rounds of choosing, the probability that all requests are satisfied is at least $1-\epsilon$.

Let us consider how the randomized algorithm updates the data-structure to reflects the addition of new routes to the state. Suppose that a processor establishes a route from input ξ to output η, i.e. two pebbles along the subpaths in T_ξ and T_η reach a common leaf α (T_ξ and T_η are embedded in graph G). It sends two bubbles back from α to ξ and η along the subpaths. It changes the value $G(\zeta)$ (from 0) to 1 for each node ζ that the bubbles encounter. Since the path is of length $2(\gamma + \nu) + 1 = O(logn)$, the updating of data-structure to reflect the addition of new routes is performed with $O(logn)$ arithmetic operations. On the other hand, determining whether or not a path is idle, and whether or not two processors seize a same link in the path is performed in $O(logn)$ arithmetic operations since G is of bounded degree (the maximum degree is 4). Thus the parallel randomized algorithm establishes (and/or abolishes) t transactions in $O((logt)(logn) = O((logn)^2)$ expected steps.

4. The Data-structure for the Deterministic Parallel Algorithm

In this section, we extend our simple data-structure for the randomized algorithm to support efficient deterministic parallel algorithms. Roughly speaking, we maintain some redundant information about the distribution of established routes, so that we can save some computation by retrieving the redundant information.

For each pair of inputs ξ_1 and ξ_2, we say their distance, $dist(\xi_1, \xi_2) = dist(\xi_2, \xi_1)$, is d $(1 \leq d \leq \nu)$, if and only if the routes starting from the two inputs may share a link after $(d+\gamma)$-th stage but cannot share any link before $(d+\gamma)$-th stage. Similarly, we define the distance $dist(\eta_1, \eta_2)$ of two outputs η_1 and η_2.

It is observed that, for each input ξ there are 2^{d-1} other inputs ξ' with $dist(\xi, \xi') = d$, for any d with $1 \leq d \leq \nu$. A similar result holds for each output. Furthermore, for any inputs ξ, ξ' and ξ'', if $dist(\xi, \xi') = d$, then $dist(\xi, \xi'') = d + \delta$ $(\delta > 0)$ if and only if $dist(\xi', \xi'') = d + \delta$ for any $1 \leq \delta \leq \nu - d$. That is to say, if the distance of two inputs is d, they share the same group of inputs of which the distance is $d + \delta$ from them. It will be much clearer if we describe the distance relationship among inputs in terms of a binary tree IND. IND is a binary tree of depth ν with 2^ν leaves. Let the leaves correspond to the inputs of N^+ in the following way. Two leaves are siblings if and only if their distance is 1; two nodes τ_1 and τ_2 at depth l are siblings if and only if the distance between leaves in the subtree rooted at τ_1 and that in the subtree rooted at τ_2 is $\nu - l + 1$ (the distance is unique, as observed above). Similarly, the distance relationship among outputs is described in terms of a binary tree $OUTD$. It is observed that two inputs (outputs, resp.) have distance d if and only if their lowest common ancestor in IND ($OUTD$ resp.) is at depth $\nu - d$.

In order to obtain an efficient deterministic parallel algorithm, we need to keep some redundant information about the distribution of established routes. The information in the data structure, on the other hand, can not be too redundant, since our algorithm dynamically updates the data structure to reflect the addition of new routes (and the subtraction of old ones). For any two inputs with distance d, the fate of whether or not the routes starting from them will block each other is determined within the first $\gamma + d + 1$ stages in N^+. Thus for each input ξ, for inputs with distance d, we confine the route distribution information to the first $\gamma + d + 1$ stages; for inputs with distance d to ξ, however, we keep their route distribution information as a whole instead of as individuals. Therefore, for each input ξ, we keep ν binary trees, of depth $\gamma + d$ having $2^{\gamma+d}$ leaves for $1 \leq d \leq \nu$, with each representing the route information of 2^{d-1} other inputs (inputs with distance d to ξ). Associated with each node ζ in such a tree is a number, measuring the number of routes that start from one of the 2^{d-1} inputs , say ξ', and contain the corresponding node of ζ in $T_{\xi'}$. Thus, for each input ξ, there are ν such trees; each input is involved in ν such trees and there are $2 \cdot 2^{\nu} - 1$ such trees in total (due to the large quantity of overlapping). Similar properties hold for outputs.

Our data-structure for deterministic algorithms is precisely described as follows. In addition to the dual graph G of N^+, we keep some redundant information about the distribution of established routes. For each node τ at depth $\nu - l$ $(\nu - 1 \geq l \geq 0)$ in IND, we keep a binary tree TR_τ (TR stands for "traffic"), which is of depth $\gamma + l$ with $2 \cdot 2^{\gamma+l} - 1$ nodes. Recalling the common structure T of T_ξ's and T_η's, we see TR_τ is a subtree of T truncated at depth $\gamma + l$. For each node ζ in TR_τ, there is a corresponding node in each tree T_ξ; for the sake of simplicity of our notation, we also denote this node by ζ. Now we associate with each node ζ in TR_τ a number $TR_\tau(\zeta)$, which is the sum of $T_\xi(\zeta)$ (i.e. the value $G(\zeta)$) over every input ξ which is a leaf in the subtree rooted at τ in IND. Similarly, for each node β at depth $\nu - l$ $(\nu - 1 \geq l \geq 0)$ in $OUTD$, make a binary tree TR_β, which is of depth $\gamma + l$ with $2 \cdot 2^{\gamma+l} - 1$ node. Associated with each node ζ is the value $TR_\beta(\zeta)$, which is the sum of $T_\eta(\zeta)$ over every output η which is a leaf in the subtree rooted at β in $OUTD$.

Let us consider the space requirements of the data structure. The graph G has less than $2(\nu + 1)2^{\gamma+\nu}$ nodes, and there is one number (0 or 1) associated with each node. There are $2 \cdot 2^{\nu-l}$ nodes τ and β at depth $\nu - l$ in trees IND and $OUTD$. For each τ or β, there is a tree of $2 \cdot 2^{\gamma+l} - 1$ nodes, and associated with each node is a number (in the range $[0, 2^\nu - 1]$). Thus there are less than

$$2(\nu+1)2^{\gamma+\nu} + \sum_{l=0}^{\nu-1}(2 \cdot 2^{\nu-l})(2 \cdot 2^{\gamma+l} - 1) < 6(\nu+1)2^{\gamma+\nu} = O(\nu^2 2^\nu) = O(n(logn)^2)$$

numbers to be stored. Therefore, the space requirement of the data-structure is $O(n(logn)^2)$ words.

5. A Deterministic Parallel Algorithm

The elimination of randomization from the randomized parallel algorithm of Section 3 is accomplished in two stages. In the first stage we greatly reduce the number of random bits (independent coin flips) used by the algorithm, by deterministically computing a large number of bits from a smaller number. In the second stage we show how to deterministically compute this smaller number of bits.

In the randomized parallel algorithm, each processor makes a random choice uniformly distributed over $2^{\gamma+\nu}$ possibilities; we may therefore regard it as making $\gamma + \nu$ successive independent random binary choices, corresponding to the $\gamma + \nu$ successive moves involved in pushing a pebble from the root to a leaf in T (T is the common structure of trees T_ξ and T_η). We may therefore imagine all of the choices of all the processors as being made in $\gamma + \nu$ successive "phases", with each of the t processors making its first choice in the first phase, and so forth.

We next observe that the analysis of the randomized algorithm was based on the assumption that all routes were independently chosen, but actually only relied on the routes being pairwise independent. Thus the analysis will remain valid if the binary choices in each phase are not completely independent, but are pairwise independent. These t pairwise independent bits can be computed deterministically from a set of $\nu = log_2 n$ completely independent random bits, using the following well known scheme.

Let M be an $\nu \times t$ matrix over $GF(2)$ in which each column is a distinct input index of the t requests (input indices are in their binary representations). Let X be a row of ν completely independent random elements of $GF(2)$, and let Y be the product XM (a row of t elements of $GF(2)$). Then the elements of Y are uniformly distributed over $GF(2)$ and pairwise independent. We may thus deterministically compute t pairwise independent bits in Y from the $\nu = log_2 n$ completely independent random bits in X.

Let us now return to the picture of pebbles being pushed from the root to a leaf in the trees T_ξ's and T_η's. As before, established routes will be replaced by pebbles at the leaves, and each processor is responsible for pushing two pebbles, one in a tree T_ξ and the other in a tree T_η. For the sake of simplicity, we label the two corresponding pebbles ξ and η as well. Given a disposition of pebbles in these trees, associate with each pebble a quantity called the "congestion", defined in the following way.

Imagine all pebbles being pushed in T (the common structure of T_ξ's and T_η's). If the pebble ξ is at a node σ at depth κ in T, the congestion of ξ is the sum of a contribution of $min\{1, 1/2^{\gamma+d-\kappa}\}$ for every pebble ξ' in the subtree rooted at σ, where $d = dist(\xi, \xi')$, plus a contribution of $min\{1, 1/2^{\gamma+d-\iota}\}$ for every pebble ξ'' at a node ρ at depth ι on the path from the root to σ, where $d = dist(\xi, \xi'')$. This quantity is easy to compute. Consider the $2^\nu - 1$ inputs other than ξ. Based on their distances to ξ, they fall into ν groups with size 2^{d-1} and of distance d to ξ, for $1 \leq d \leq \nu$. Of the inputs in the group of distance d to ξ, their lowest common ancestor in IND is at depth $\nu - d + 1$ (recall that inputs correspond to leaves in IND). Let these ancestors be $\tau_1, \cdots, \tau_\nu$. The congestion of ξ equals to the sum of $min\{1, 1/2^{\gamma+d-\kappa}\} \cdot TR_{\tau_d}(\sigma)$ over every d, $1 \leq d \leq \nu$, plus for each involved ρ, the sum of $min\{1, 1/2^{\gamma+d-\iota}\} \cdot TR_{\tau_d}(\rho)$ over every d. We say that the congestion of a request is the sum of the congestions of its two pebbles, and that the congestion of a batch of requests is the sum of the congestions of the t requests in the batch.

The success of the randomized algorithm may now be ascribed to three simple observations. Firstly, when all the pebbles of the requests in the batch are at the root of T ($\kappa = 0$), the congestion of a pebble is less than or equal to $\sum_{d=1}^{\nu}(1/2^{\gamma+d}) \cdot 2^d = \nu/2^\gamma < 1/4$, corresponding to contributions of $1/2^{\gamma+d}$ for each of the other pebbles at leaves or at the root. Secondly, if a pebble is moved from a node σ to one of its two children (chosen at random with equal probability), the expected congestion of each pebble is unaffected; indeed, each contribution to the congestion is either unaffected or undergoes a "double-or-nothing" transformation with equal probabilities. Finally, when all pebbles are pushed to leaves ($\kappa = \gamma + \nu$), the congestion of a pebble is an integer greater than or equal to 1 if it is blocked, is 0 if it is not blocked. Therefore, the number of pebbles being blocked is less than or equal to the congestion of the batch of requests. It follows from these observations that on the average, at least one-half of requests finish with both pebbles not being blocked, and are successful.

Let us now combine this picture with the notion of phases, so that each processor pushes its two pebbles down one level in their trees during each phase. If all of the binary choices involved in these pushes were completely independent, the expected congestion would be unaffected. Since the congestion is defined as a sum of pairwise contributions, its expectation is unchanged if completely independent binary choices are replaced by pairwise independent binary choices. So let t pairwise independent choices be deterministically computed from ν completely binary choices, as described above. Since the expectation over all ν choices is unaffected, it follows that there is a particular way of making the first choice for which the expectation (over the remaining $\nu - 1$ choices) does not increase. After the first choice has been made in this way, there is a particular way of making the second choice for which the expectation (over the remaining $\nu - 2$ choices) does not increase. Proceeding in this way, we arrive at particular ways of making all ν choices, from which we may deterministically compute the t pushes of pebbles.

It remains to observe that the "particular ways" whose existence was argued in the preceding paragraph can in fact themselves be deterministically computed in a simple way. For if we assign particular values to some of the ν choices, the expected congestion over the remaining choices can be computed as follows. Assigning particular values to some of the choices commits some of the t pebbles to move from the node σ at which they began the phase to one of their two children, while leaving the other pebbles equally likely to move to either of their two children (the fate of a particular pebble is sealed when all of the entries of X for which its column of M contains a 1 have been assigned particular values; otherwise, its fate is uncompromised). Thus an advantageous value for a choice can be found by tentatively assigning one value, recomputing the congestions, and rescinding the tentative assignment in favour of its alternative if the congestion increases.

By now we have finished the description of our deterministic parallel algorithm. Let us consider the performance of this algorithm. It establishes and/or abolishes any number of routes in parallel in $O((logn)^5)$ with one processor per transaction, maintaining a data structure of $O(n(logn)^2)$ words. To estimate the time complexity, we observe the following facts. Firstly, each time pebbles being pushed to their leaves, there are at least one half of the requests being satisfied, which implies $\lceil log_2 t \rceil = O(logn)$ "rounds" of pushing are sufficient to satisfy all the requests. Secondly, within each "round" of pushing, $\gamma + \nu = O(logn)$ "phases" are sufficient to push a pebble from its root to a leaf. Thirdly, in each "phase", $\nu = O(logn)$ bits in X are deterministically computed. Finally, in order to determine the value of one bit, the congestion of the batch of requests is computed, this is done with $O((logn)^2)$ steps, as the congestion of a pebble is computed in $O((logn)^2)$ steps by one processor (sum of $min\{1, 1/2^{\gamma+d-\kappa}\} \cdot TR_{\tau_d}(\sigma)$ over every d, $1 \leq d \leq \nu$, plus for every ρ involved, the sum of $min\{1, 1/2^{\gamma+d-\iota}\} \cdot TR_{\tau_d}(\rho)$ over every d)[1], and after the

[1]In fact, our algorithm computes the congestion of a pebble in $O(logn)$ steps, since pebbles are pushed "phase by phase", at most one ρ, i.e. the parent node of ξ is involved. This, in turn, implies the congestion of a batch of requests is computed in $O(logn)$ steps.

congestion of each pebble is computed, the congestion of a batch of requests in computed in $O(logn)$ steps by t processors in parallel; and the update of the data structure after determination of a bit (committing some of the t pebbles to move to one of their two children) needs $O((logn)^2)$ steps, of which one $O(logn)$ factor comes from the fact that $O(\nu) = O(logn)$ numbers in TR_{τ_d}'s, $1 \leq d \leq \nu$, are to be updated (at most two numbers in each TR_{τ_d}), and the other comes from the fact that to any one of these numbers, up to $t = O(n)$ processors may want to update it simultaneously (with each one adding 1 or subtracting 1).

References

[ALM] S. Arora, T. Leighton and B. Maggs, "On-Line Algorithms for Path Selection in a Nonblocking Network", *ACM Symp. on Theory of Computing*, 22 (1990) 149-158.

[B] V. E. Beneš, "Optimal Rearrangeable Multistage Connecting Networks", *Bell Sys. Tech. J.*, 43 (1964) 1641-1656.

[C53] C. Clos, "A Study of Non-blocking Networks", *Bell Sys. Tech. J.*, 32 (1953) 406-424.

[C71] D. G. Cantor, "On Non-blocking Switching Networks", *Networks*,1 (1971) 367-377.

[FFP88] P. Feldman, J. Friedman and N. Pippenger, "Wide-Sense Non-Blocking Networks", *SIAM J. Discr. Math.*, 1(1988) 158-173.

[LPV] G. Lev, N. Pippenger and L. G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks", *IEEE Trans. on Computers*, 30 (1981) 93-100.

[L86] M. Luby, "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM J. Computing*, 15 (19886) 1036-1053.

[L88] M. Luby, "Removing Randomness in Parallel Computation without a Processor Penalty", *IEEE Symp. on Foundations of Computer Science*, 29 (1988) 162-173.

[PY] N. Pippenger and A. C. Yao, "Rearrangeable Networks with Limited Depth", *SIAM J. Alg. Disc. Meth., Vol. 3, No. 4*, (1982) 411-417.

[P73] N. Pippenger, "The Complexity Theory of Switching Networks", *Ph. D. Thesis*, Electrical Engineering, MIT, August 1973.

[P82] N. Pippenger, "Telephone Switching Networks", *AMS Proc. Symp. Appl. Math.*, 26 (1978) 101-133.