The Complexity of Reconfiguring Network Models

Y. Ben-Asher * K.-J. Lange \dagger D. Peleg \ddagger A. Schuster §

Abstract

This paper concerns some of the theoretical complexity aspects of the reconfigurable network model. The computational power of the model is investigated under several variants, depending on the type of switches (or switch operations) assumed by the network nodes. Computational power is evaluated by focusing on the set of problems computable in constant time in each variant. A hierarchy of such problem classes corresponding to different variants is shown to exist and is placed relative to traditional classes of complexity theory.

^{*}Department of Mathematics and Computer Science, The Haifa University, Haifa, Israel. E-mail: yosi@mathcs2.haifa.ac.il

[†]Department of Computer Science, Technische Universität München, 80290 München, Germany. E-mail: lange@informatik.tu-muenchen.de

[‡]Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel. E-mail: peleg@wisdom.weizmann.ac.il. Supported in part by an Allon Fellowship, by a Bantrell Fellowship and by a Walter and Elise Haas Career Development Award.

[§]Department of Computer Science, Technion, Technion City, Haifa, Israel 32000. E-mail: assafs@cs.technion.ac.il

1 Introduction

In sequential computation there is one widely acceptable model, namely, the *von-Neumann* model. In contrast, there is still no such popular equivalent for parallel computation. In particular, it is not clear which parallel model of computation is the best candidate to bridge the "hardware - software gap," as discussed in [Val90]. The PRAM family is usually considered as the ideal computational environment, for its freedom of restrictions on memory access. At the other extreme, the Fixed Connection Network model (FCN) is viewed to be a "realizable" parallel environment, since each processing element is connected to a constant number of other elements. Recent developments in technology have made several other computational models viable. Such models may be as strong as (or even stronger than) the PRAM model on the one hand, and on the other hand exhibit realizability of the same level as (or even higher than) that of the FCNs.

One of the most promising parallel models of computation is the *Reconfigurable Network* (RN) model. The basic idea of the RN model is to rely on bus communication, and enable flexible connection patterns, by allowing nodes to connect and disconnect their adjacent edges in various patterns. This yields a variety of possible bus topologies for the network, and enables the program to exploit this topological variability in order to speed up the computation.

Informally, a reconfigurable network operates as follows. Essentially, the edges of the network are viewed as building blocks for larger *bus* components. The network operates in rounds, and dynamically reconfigures itself at each round, where an allowable configuration is a partition of the network into several connected components, or, a set of edge-disjoint buses. A crucial point is that the reconfiguration process is carried out *locally* at each processor (or *switch*) of the network. That is, at the beginning of each round during the execution of a program on the RN, each switch of the network fixes its *local configuration* by partitioning its collection of edges into some combination of subsets. Adjacent edges that are grouped by a switch into the same subset are viewed as (hardware) connected, so that they form a bus. Any processor connected to an edge participating in the construction of a certain bus, may choose to listen to any incoming or passing message transmitted on that bus.

The basic assumption concerning the behavior of the reconfigurable model (as well as any other bus model) is that in any configuration, the time it takes to transmit along any bus is constant, regardless of the bus length. This assumption is theoretically false, as the speed of signals carrying information is bounded by the speed of light. Hence with very fast processors, and assuming that the operation rate of the parallel machine has to equal that of the individual processors (i.e., that each processor cycle includes a round of communication), the actual bus lengths that can be implemented are more limited. This partially explains why the RN model and other bus models have not gained wide acceptance initially.

Recently, however, implementations were suggested for the RN model, involving a variety of newly developed technologies, including optical communication and optical computing devices. Several dynamically reconfiguring machines involving thousands of switches were actually built [TCS89, GK89, LM89, MKS89, WLH⁺87], showing that the RN model is implementable in massively parallel architectures.

Motivated by the existing implementations, there has been some work on the algorithmic

and computational aspects of the RN model. Nakatani [Nak87] considered comparison-based operations like merging, sorting and selection on reconfigurable arrays. Miller, Stout, Reisis and Kumar [MPRS87] and Reisis and Kumar [RP87] considered parallel computations and data movement operations on the reconfigurable mesh. In a recent series of papers, summarized in [Wan91], Wang, Chen and others present many constant time algorithms for RN's. In [BS91, Sch91] the parameter of *bus-usage* is suggested as a measure for the efficiency of RN algorithms. Other papers consider image processing and fault tolerance on RN's.

This expanding volume of algorithms and results calls for a more systematic approach and a theoretical evaluation of the classes of problems solvable using RN's. In particular it is evident that RN's solve large sets of problems in constant time. This power is attributed to the exponential number of global configurations that may be taken by the network at each step. When the problem is solvable by reconfiguring locally according to the input, then the global configuration gives the result instantaneously. Thus, for example, it is shown in [BPRS91] how to sort in constant time using one RN model, and how to solve a *PTIME*-complete problem in constant time using another (stronger) RN model. Some comparisons and simulations of basic RN models are presented there as well.

In an earlier work, Moshell and Rothstein [MR79] investigated the computational complexity of the *Bus Automata* (BA). The BA model is similar to the RN model. It is composed of a *d*-dimensional array of finite automata with modifiable channels allowing long-distance communication. Moshell and Rothstein showed that large classes of problems are solvable in constant time on the BA. For example, they showed that the languages recognizable in constant time by a one-dimensional BA are exactly the regular languages.

In this work we extend the ideas from [BPRS91] in order to evaluate the theoretical power of several different RN models. We concentrate on the classes of problems solvable in constant time. Our approach, however, is different from the one given in [MR79] in several aspects. In particular, the underlying topologies assumed for the networks are not necessarily uniform arrays (although we do show equivalence in several cases) and the switches differ in their operation on passing messages. We show that variations in the switching assumptions result in variations in the power of the model. Finally, we present results that relate these models to space-bounded Turing machines and parallel complexity classes.

The rest of this work is organized as follows. Section 2 describes the RN model in more detail. In Section 3, the RN model is compared with the PRAM model, and some connections are established between the corresponding complexity classes. In Section 5 similar comparisons are made with respect to Turing-machine based complexity classes. Section 4 concerns the restriction of the RN model to simple two-dimensional mesh topologies. Section 6 considers the non-monotone RN model. Finally, Section 7 concludes with a discussion and some open problems.

2 Reconfigurable Models of Computation

2.1 The General Model

A reconfigurable network (RN) is a network of switches operating synchronously. The switches residing at the nodes of the network perform the same program, taking local reconfiguring decisions and calculations according to the input and locally stored data. In this paper we focus on networks of bounded degree, hence the number of possible configurations at a node is constant. Input and output locations are specified by the problem to be solved, so that initially, each input bit (or item) is available at a single node of the network, and eventually, each output bit (or item) is stored by one.

A single node of the network consists of a computing unit, a buffer and a switch with reconnection capability. The buffer holds either an input or an output item, or something that was previously read from adjacent buses. The power (instruction set) of the computing unit is not central to the discussion, although it varies from section to section. For example, for the simulations of Turing machines by RN's we assume no computation power at all, so that no arithmetic or logic operations are allowed. For the simulations of PRAM's (and by PRAM's) we assume the processor power of the simulating and simulated models to be the same. In many cases, the sole objective of the computing unit is to decide the next state of the switch¹ according to the data stored at the local buffer. In simulating other models by RN's, the size of the buffers typically remains small. If a *word* (whose length is determined by the bus bandwidth) is moved on the bus in a single step, then the size of the buffer need only be a constant number of words.

A single round (or step) of an RN computation is composed of the following substeps.

- Substep 1: The network selects a *configuration* H of the buses, and reconfigures itself to H. This is done by local decisions taken at each switch individually, depending on the input, the contents of messages previously read from adjacent buses and local computation results. This substep may in principle extend over a (constant) number of processor cycles.
- Substep 2: One or more of the processors connected by a bus transmit a message on the bus. These processors are called the *speakers* of the bus.
- Substep 3: Some of the processors connected by the bus attempt to read the message transmitted on the bus by the speaker(s). These processors are referred to as the *readers* of the bus.

Remark 1: It is sometimes helpful to make use of a message that has no inherent meaning (except for its origin and destination, determined by the bus configuration). Such a message is referred to as a *signal*. In such cases, information is conveyed by the knowledge of which of the readers succeed in actually detecting the signal.

At each round, a bus may take one of the following three states:

¹In the sequel we ignore these distinctions, and use the terms *switch*, *node* and *processor* interchangeably.

- *Idle*: no processor transmits,
- Speak: there is a single speaker,
- *Error*: there is more than one speaker.

An *Error* state, reflecting a collision of several speakers, is detectable by all processors connected by the corresponding bus, but the messages are assumed to be destroyed. (This definition follows the common model, but it is worth commenting that a number of other reasonable alternatives exist. For example, it is sometimes assumed that collisions go undetected. On the other extreme, a stronger model which may be useful is one assuming that the outcome of a collision is more informative, and yields some partial function of the transmitted messages, e.g., their logical "OR".)

The most popular reconfiguring network in the existing literature is the mesh. An $n \times m$ mesh consists of an $n \times m$ array of switches beginning with switch (0,0) at the upper left corner and ending with switch (n-1, m-1) at the lower right corner of the mesh. Each switch has four I/O ports (L, R, U, D) for its Left, Right, Up and Down neighbors (except those on the perimeter of the mesh, which have three ports, and those in the corners which have only two ports each). For example, if port U is connected to port L and ports R and D are disconnected, then we denote by (U - L) the configuration of the switch.

2.2 Variations on Operations

The general RN model, as presented above, does not specify the exact operation of the switches. As already shown in [BPRS91], the specific operation determines the power of the model. We consider the following four basic variants.

- **General RN:** The switch may partition its collection of edges into any combination of subsets, where all edges in a subset are connected as building blocks for the same bus. Thus the possible configurations are any network partition of edge-disjoint connected subgraphs.
- Linear RN (LRN): The switch may partition its collection of edges into any combination of connected pairs and singletons. Hence buses are of the form of a path (or a cycle) and the global configuration is a partition of the network into paths, or a set of edge-disjoint linear buses.
- **Directed RN (DRN):** This model is similar to the Non-Linear RN model, except that edges are directed, so messages travel in one direction only. Consequently, each connected subset of edges is split into *in-edges* and *out-edges*. A message entering the switch for the first time via either one of the in-edges, proceeds via all the out-edges connected to it.
- Non-Monotone RN (NMRN): This model is the same as the Directed RN model, but a switch has an additional "inversion" capability. When this operation is activated by the switch, a signal going via the switch is inverted. That is, a "0" ("no signal") turns into a "1" ("signal on") and vice versa.

By way of illustration, let us consider again the reconfigurable mesh operating in the LRN model. A switch may take one out of ten possible local configurations: (L - R, U - D), (L - D, R - U), (L - U, R - D), (U - D), (L - R), (R - D), (L - U), (R - U), (L - D), (L - D), and (). When the mesh operates in the RN model, five more local configurations are possible: (L - R - U), (R - U - D), (U - D - L), (D - L - R), and (L - R - U - D).

Discussion: It is important to observe that the notion of a *bus* for DRN's and NMRN's is somewhat different than that of LRN's and RN's. The most significant difference is that while the RN architecture is based on "passive" wires, the DRN and NMRN models make use of more "active" (hence slower) devices along the way.

Another technical difference involves the way the destination set of a message is determined in the DRN and NMRN models. This is done as follows. Suppose some processor z transmits at round t, and let H_t denote the global configuration that was chosen by the network during step t. Then the message issued by z on some connected set of out-edges reaches the subgraph of H_t consisting of all nodes that may be reached from z by a *directed* path starting at those out-edges.

The notion of *bus error* for DRN's and NMRN's changes, too. A node y detects an error during step t if, in the configuration H_t , y is reachable from two different speakers. Hence it may happen that a message issued by some speaker z will be correctly received by a reader, while other readers that are reachable from z detect an error since they are reachable from other speakers too.

Hence from an architectural point of view, the reference to the channel devices used in the DRN and NMRN as "buses" may be a bit stretched, and the assumption of constant propagation delay is not as justified in current technologies (although the development of very fast active switches is currently being investigated by various industries). Nevertheless, since the present paper focuses on a theoretical comparison of the computational power of the various models, we shall opt for uniformity of framework and terminology, by maintaining both the basic constant-delay assumption and the use of the term "buses" to describe the communication mechanism in all four models under discussion. Further research may be necessary to adjust our results to a more accurate model, taking these differences into account by modifying the assumptions on the propagation delay.

2.3 Complexity Classes

Let Σ denote a symbol-set and let $\Sigma^* = \bigcup_{i \ge 1} \Sigma^i$. A problem A is a mapping $A : \Sigma^* \longmapsto \Sigma^*$. Using standard reductions, the discussion can be restricted to Boolean problems $A : \Sigma^* \longmapsto \{0, 1\}$. An input-instance I for A is said to be solved by presenting A(I). An RN family, $\mathcal{R} = \{R_N\}_{N \ge 1}$, of reconfiguring networks is a set containing a network construction R_N for each natural N. We say that the family \mathcal{R} solves a problem A if for every N, R_N solves all size N inputs for A, $\{I : |I| = N\}$.

We consider two measures for computation complexity in the RN model.

Time: T(R) is the worst-case number of rounds it takes for the computation of the reconfiguring network R to terminate,

Size: S(R) is the number of switches in the reconfiguring network R.

A reconfiguring network family $\mathcal{R} = \{R_N\}$ has time complexity f(N) if for every $N \ge 1$, a computation of R_N terminates within $T(R_N) = O(f(N))$ rounds for all valid input instances of length N. The family \mathcal{R} has size complexity g(n) if for every $N \ge 1$, R_N consists of $S(R_N) = O(g(N))$ switches.

The description $\mathcal{D}(R)$ of a reconfigurable network R, is a list of S = S(R) triplets of the form $\langle x, \Gamma^x, \mathtt{Rules}^x \rangle$, one for each node x of the network. In this description, x is the node's id, Γ^x is the list of immediate neighbors of x in the underlying topology R, and \mathtt{Rules}^x is a set of configuration and output rules for x (depending on the inputs, the current round and the data read from adjacent buses in previous rounds). Since we focus on constant-degree networks and constant-time programs, we may assume that a triplet consists of $O(\log S)$ bits. The total network description is thus of size $O(S \log S)$ bits.

The class of reconfigurable networks $\mathcal{RN}(f(N), g(N))$ in the RN model, is the set of families \mathcal{R} with the following properties:

- (a) \mathcal{R} is of time complexity f(N) and size complexity g(N), and
- (b) \mathcal{R} is uniformly generated in $SPACE(\log(g(N)))$, i.e. there exists a Turing machine (TM), M, that given N produces the description of R_N using $O(\log(g(N)))$ cells of its working tape.

Similar classes are defined analogously for the LRN, DRN and NMRN models. Correspondingly, these are denoted $\mathcal{LRN}(f(N), g(N))$, $\mathcal{DRN}(f(N), g(N))$ and $\mathcal{NMRN}(f(N), g(N))$.

We define the set of problems RN(f(N), g(N)) to include any problem A for which there exists a network family $\mathcal{R} \in \mathcal{RN}(f(N), g(N))$ solving it. The problem sets LRN(f(N), g(N)), DRN(f(N), g(N)) and NMRN(f(N), g(N)) are defined analogously.

Some natural relationships exist among the above classes. For example, since a switch in the RN model can simulate a switch in the LRN model, we immediately have:

Lemma 2.1 For any two functions f(N) and g(N), $LRN(f(N), g(N)) \subseteq RN(f(N), g(N))$.

We also need a notion of uniformity for the time/size functions. A function f(N) is said to be constructible if it is computable by a TM M_f having N as its input and using O(f(N)) cells of its working tape.

3 PRAM Algorithms and RN's

In this section we consider the question of how powerful polynomial size RN's are, compared to parallel models of computation with a shared memory unit. In particular we are interested in the common PRAM model (cf. [KR90]).

Theorem 3.1 A T-step computation of an N-switch RN with E edges can be simulated by an O(E)-processor CRCW PRAM in time $O(T \log N)$.

Proof: Let R be an N-switch, E-edge RN. A CRCW PRAM algorithm for simulating R is constructed as follows. The PRAM gets as its input both the adjacency matrix of the RN and the input to the RN. Each round of the RN is simulated by the PRAM in four phases as described below.

(1) The first phase incorporates only N of the PRAM processors, each simulating a single switch of R. This phase is dedicated to simulating the internal computation taken by the RN switches, in which the bus splitting, speaking, reading and the (virtual) local configuration are decided.

Once a switch decides on a certain local configuration, its edges are grouped into connected sets. Thus the global configuration of R can be represented by an augmented graph \tilde{R} by splitting each switch s of R into several logical copies, $C(s) = \{s_1, \ldots, s_l\}$, one for each connected component of its edges. Each original RN switch whose degree in R is d, is represented in \tilde{R} by at most d nodes. The total number of nodes in this augmented graph \tilde{R} is thus at most 2E, and each of these nodes has degree at most d. The crucial observation here is that the connected components of \tilde{R} represent the buses in R.

- (2) Each of the nodes of R is emulated by a CRCW processor. In the second phase, the local configuration of a switch s in R is read by each processor emulating a node s' ∈ C(s) in R, where s' connects several of the edges of s. The processor emulating s' needs also the id of the processors emulating neighboring nodes. This information is disseminated relatively fast; if d is the highest degree of any switch in R, then the second phase requires O(d log d) reading steps, namely, constant time.
- (3) The processors of the CRCW PRAM, standing for nodes of the global configuration graph \tilde{R} , construct a balanced spanning tree for each bus (connected component) using the $O(\log N)$ -time connectivity algorithm of [SV82].
- (4) The speakers of each bus use the tree constructed at phase (3) to broadcast messages (and detect errors). This can be achieved in $O(\log N)$ time via standard doubling techniques.

Phases (1) and (2) require constant time, and phases (3) and (4) require $O(\log N)$ time each. Hence each step of the RN is simulated by the PRAM in $O(\log N)$ time, and the theorem follows.

A connection analogous to Thm. 3.1 was established in [BPRS91] between the LRN and EREW PRAM models.

Theorem 3.2 [BPRS91] A T-step computation of an N-switch LRN with E edges can be simulated by an O(E)-processor EREW PRAM in time $O(T \log N)$.

¿From the two theorems we get

Corollary 3.3 A problem of input size N that is computable by a T(N)-step, polynomial-size LRN (respectively, RN), has an $O(T(N) \log N)$ -step EREW (resp., CRCW) PRAM program. In particular, a problem having $O(\log^{K} N)$ -step, polynomial-size LRN's (resp., RN's) with uniformly generated underlying topologies is in (uniform) EREW^(K+1) (resp., CRCW^(K+1)).

In other words, the corollary implies that problems that are "inherently sequential", i.e., that are "non parallelizable" using traditional parallel models, maintain this property under the RN and the LRN models. Theorem 5.28 implies that this meta-claim holds for the DRN model, too. In contrast, the results of Section 6 imply that this is not the case for the NMRN model.

As already mentioned, many problems requiring $\Omega(\frac{\log N}{\log \log N})$ steps on a CRCW PRAM (or $\Omega(\log N)$ steps on an EREW PRAM) with polynomial number of processors, can be computed by a constant-time polynomial-size RN. The following theorem shows that this is not the case for the opposite direction.

Theorem 3.4 [BPRS91, WC90b] A (priority) CRCW PRAM with P(N) processors, M(N) memory cells and T(N) time can be simulated by a O(T(N))-step, $P(N) \times M(N)$ mesh operating in the LRN model.

4 Universality of the Mesh

In this section we show that the two-dimensional mesh is computation universal and achieves high speedup. We say that a problem A is in the class $Mesh_LRN(t(N), r(N), c(N))$ if for each N, the $r(N) \times c(N)$ mesh solves all size N inputs to A in t(N) steps. Similar definitions apply for the RN, DRN and NMRN models. Let us first review several known results for the LRN model.

We follow [KR90] for the definitions of *circuits* and their *depth*. Given a family of (bounded fan-in) circuits $C = \{C_i\}, i \ge 1$, we say that C is in CKT(D(N)) if the depth of C_N is O(D(N)) for each N. The size of a circuit is its number of edges. A circuit C of size |C| is *uniform* if its description can be generated by a Turing machine using $O(\log |C|)$ workspace (see Sec. 5.1). A problem A is in CKT(D(N)) if there is a family of uniform circuits $\{C_N\}_{N\ge 1}$ in CKT(D(N)), that solves A.

Lemma 4.1 [BPRS91] $CKT(d) \subseteq LRN(O(1), 2^{(1+\epsilon)d})$ for every $\epsilon > 0$.

In particular, putting $d(N) = O(\log N)$ or $d(N) = O(\log^2 N)$ we have

Theorem 4.2 [BPRS91]

 $\begin{array}{lll} NC^1 & \subseteq & LRN(O(1), poly(N)) \\ NC^2 & \subseteq & LRN(O(1), N^{O(\log N)}) \end{array} .$

In fact, the result is stronger; there exist uniform "universal" constructions computing all functions of the same circuit complexity. The following lemma, the construction and the reconfiguring program that follows, all use the constructions of Barrington [Bar86] and the subsequent [CL89, Cle90]. It is important to note that the results are constructive and uniformly generated.

Lemma 4.3 [BPRS91] For every fixed $\epsilon, c > 0$ there exists a (universal) LRN network of size $O(N^{c(1+\epsilon)})$ computing in constant time all functions that are computable by circuits of depth $c \log N$.

The above results can be used to prove the universality of the mesh, as follows. Consider a problem A for which there are circuits of depth at most $c \log N$ solving all N-sized inputs. We now describe a simulation method that constructs, for a given N, an $N \times N^{c(1+\epsilon)}$ mesh solving all N-sized inputs in constant time. We refer to this mesh as the universal mesh for A and N.

The Universal Mesh

The construction of the rectangular $N \times poly(N)$ mesh is similar to the universal LRN construction of [BPRS91, Sect. 3,4] (also cf. [Sch91]), so we omit most details of the construction, and focus on the differences. The K-to-K permutation networks that are used in [BPRS91] are replaced by $K \times K$ meshes. Note that the LRN mesh supports any permutation of the leftmost column switches to the rightmost column switches in a single round (given that the local configurations are computed in advance).

For the problem A and a given N, the universal mesh is composed of an initializing network and a row of $N^{(1+\epsilon)c}$ meshes of size $K \times K$. The row of meshes is easily embedded in a rectangle of size $K \times KN^{(1+\epsilon)c}$.

The initializing network carries the switching information necessary to determine the permutations to be taken by the $K \times K$ meshes. This information consists of both the bits that are determined by the emulated circuit and the N bits composing the input instance to the problem. These (binary) inputs are given at the leftmost column of the mesh, so that the *i*th input bit is given at switch (i, 0). The bulk of the input, namely the circuit data, determine the choice of two specific permutations for each $K \times K$ mesh. We assume that these permutations are computed and distributed in advance. In other words, we assume that it is known which problem is to be solved, and each switch has two local configurations, which are written in it when the mesh is created.

The computation proceeds by an initialization phase and a single computation step, in which a signal is sent from the source to exactly one of the sinks of the network. In the initialization phase, the N input bits are used in each $K \times K$ mesh for choosing one of the two permutations. Note that by the construction of the universal LRN (see [BPRS91, Sch91]), a single mesh uses only a single input bit in order to determine which of the two permutations is appropriate. There may be, however, many meshes using the same input bits. Hence the initializing network is constructed of N buses that are wired along the row of meshes. These carry the input bits and are read at the appropriate columns.

We have the following result.

Lemma 4.4 Every problem for which there are circuits of depth at most $c \log N$ solving all N-sized inputs, is computable in constant time by a universal $N \times N^{c(1+\frac{2}{\log N+2})}$ mesh.

Proof: The total width of the construction described above is N. The K rows of the computing network are also used for the initialization step. The construction length is $KN^{(1+\epsilon)c}$. Note that we assume $K \leq N$. Indeed, minimizing the number of switches involved by using the relation given in [BPRS91, Theorem 4.3], we find that $K = O(2\sqrt{2c \log N})$ and $\epsilon = \sqrt{2/(c \log N)}$ are the optimal choices. Since the size of the input necessitates a rectangle of width at least N, it is possible to reduce ϵ further by choosing K = N and $\epsilon = 2/(\log N - 2)$.

As an immediate corollary we get

Theorem 4.5 $NC^1 \subseteq Mesh_LRN(O(1), N, poly(N)).$

Next, let us discuss scaling techniques, enabling the use of a fixed-size mesh for solving increasingly larger problems. Suppose that we would like to compute some function having a circuit of depth $O(c \log N)$ on a given LRN mesh M whose dimensions are fixed (and are not a function of N). Let $L_1 \times L_2$ be the dimensions of M. The $N \times KN^{(1+\epsilon)c}$ rectangle used in Theorem 4.5 may be embedded on M, e.g., in a snake-like form. If the rectangle fits into M in its entirety, then we are done. As N gets larger, however, computation can not be completed in a single sweep. Rather, it is executed in *supersteps*. Each superstep involves the embedding of part of the rectangle on M, sending the inputs to the embedded columns and computing by sending the signal along the embedded part of the rectangle. The signal is transmitted from a switch at the first (embedded) column. The switch detecting it on the last (embedded) column is recorded and is used for transmission at the next superstep.

There are several cases to consider while setting the parameters involved in the simulation. If $L_1 \ge N$ and $L_2 > 2N$ then the embedding and simulation is as described above. We need $L_1 \ge N$ for the width of the rectangle and we need $L_2 > 2N$ for the "curves" of the embedded rectangle.

Corollary 4.6 If $L_1 \ge N$ and $L_2 > 2N$, then a problem having a $O(c \log N)$ depth circuit is computable by the $L_1 \times L_2$ LRN mesh in $O(N^{(1+2/(\log N-2))c+1}/(L_1(L_2-2N)))$ steps.

The requirements $L_2 > 2N$ and $L_1 \ge N$ may be eased considerably, by choosing $K \ll N$. Then, we need $L_2 > 2K$ and $L_1 \ge K$. The price for this modification is in the original rectangle construction becoming longer, so that the computation takes more steps. Also, since there are less than N rows in the embedded rectangle, inputs are transmitted in N/L_1 steps. Input *i* is read at step j + 1 by all the switches of row *p* of *M*, where $i \equiv p \pmod{L_1}$. This input procedure is executed only once at the beginning of the computation.

Observe that in the above setting there may be up to L_1/K rectangle columns embedded on the same column of M. These may be seeking for different input bits. Since K consecutive columns seek for the same input bit, if $L_1/K \leq NK/L_1$ then there is enough time for all embedded columns to get their inputs during the input procedure. If, on the other hand, $L_1/K > NK/L_1$ then there is a need to either spend more time on the input procedure or on the computation part. **Corollary 4.7** For all $K \ge 16$, if $L_1 \ge K$, $L_2 > 2K$ and $L_1^2/K^2 \le N$, then a problem that is computable by a circuit of depth $O(c \log N)$ can be computed by the $L_1 \times L_2$ LRN mesh in $O(N/L_1 + KN^{(1+2/(\log K-2))c}/(L_1(L_2 - 2K)))$ steps.

Using the construction of Cai and Lipton [CL89], this bound can be improved further for the case $K \geq 5$. The computation takes $O(N/L_1 + N^{1.81c}/(L_1(L_2 - 2K)))$ steps on the $L_1 \times L_2$ LRN mesh.

We now turn to showing that in the RN model, any general network R can be simulated by a mesh M whose size is approximately the square of that of R's size.

Lemma 4.8 $RN(O(1), S(N)) = Mesh_RN(O(1), O(S(N)), O(S(N))).$

Proof: The non-trivial direction is to show that

 $RN(T, S(N)) \subseteq Mesh_RN(O(T), O(S(N)), O(S(N)))$.

Let R be a network in the RN model, having S = S(N) switches. Let E denote the set of edges of $R, E = \{e_1, e_2, \dots, e_h\}$. Since R is of constant degree, h = O(S). Consider the reconfigurable mesh M of size $h \times h$. M simulates a single step of R with the following algorithm.

Basically, the *i*'th column and the *i*'th row provide M with the communication channel supported in R by the edge e_i . Their intersection with the other columns and rows is connected if the corresponding edges are connected to e_i at the simulated step. Suppose that the columns and rows of M are connected in this way. Then by induction on the distance of e_i and e_j , it can be shown that the switches of row and column j read a message issued by a switch of row/column i if and only if e_i and e_j belong to the same connected component (in R, during the simulated step).

We denote by s_k^R the k'th switch of R, and by $s_{i,j}^M$ the (i,j)'th switch of the mesh M.

Algorithm UNIFORM_RN:

The algorithm is composed of two parts per each emulated step, an *initialization* part and an *emulation* part. The initialization part involves several steps, while the emulation part involves a single emulation step. During the steps of the initialization (except for the emulation of the first step) the configurations connect each row in a linear bus, while disconnecting "vertical" connections. Information is transmitted by several switches of the row and gathered by all others.

Initialization part: We describe in detail the initialization part of step t, for some t > 1. Suppose that e_j and e_i are connected to the same switch of R. Then, at some step of the initialization part, $s_{i,j}^M$ transmits on row i any message that it "read" on column j during the emulation part of the emulated step t-1. We note that the order of transmission on the i'th row during the initialization part may be determined in advance when the network is constructed, or simply by the natural order of id's of edges incident to e_i in R. Also, since R has constant degree, the number of transmitting switches during this part (for any row) is bounded by O(1). **Emulation part:** Let $e_i = (s_{k1}^R, s_{k2}^R)$. As a result of the initialization part, each switch of row i of M stores all the data that is stored by s_{k1}^R and s_{k2}^R after the (t-1)'st step. The switch $s_{i,j}^M$ emulates the configuration decision taken by the switch connecting (or disconnecting) e_i and e_j . If the configuration connects e_i and e_j in R during the t'th step, then $s_{i,j}^M$ connects all its edges during the emulation step. Else it connects its row edges to each other, and likewise for its column edges.

For a switch s_k^R , let e_i be an edge that is attached to it in R. Suppose e_i is connected by s_k^R with several other edges at the emulated step t. Suppose also that in that set of connected edges, e_i is the edge having the lowest-*id*. Using the information read during the initialization part, $s_{i,i}^M$ also decides to transmit a message m at the emulation step, depending on whether s_k^R transmits m at the emulated step on the set of edges to which e_i is connected.

It is left to show how the initialization part of the emulation of the first step is carried out. The main issue involves making the inputs that appear in R at some switch s_k^R known to all the switches in the rows of M corresponding to all the edges attached to s_k^R . These inputs appear at $s_{l,l}^M$ in M, where e_l is an arbitrary edge incident to s_k^R . During the first step (of the initialization part of the emulation of the first emulated step) $s_{l,l}^M$ transmits the inputs on column l of M, where it is read by all the switches. Suppose $e_m = (s_{l,1}^R, s_{l,2}^R)$ and suppose that the inputs that appear in $s_{l,1}^R$ and $s_{l,2}^R$ in R, appear in $s_{l,1,1}^M, s_{m,l,2}^M, \ldots$ in M. Then after the first step the inputs that are required at row m are known to $s_{m,l,1}^M, s_{m,l,2}^M, \ldots$ Next these switches transmit the inputs on row m (in a pre-determined order) so that all the switches of that row read them.

5 Relations to Turing Machines

In this section we show some basic relations between classes of problems computable in constant time by polynomial-size RN's and classes of problems solvable by space bounded TM's.

5.1 Notation and Basic Definitions

Let us first give some notation and review the definitions for the components of the TM M. The reader is referred to [HU79] for an introduction to related terminology that is not explained here (although, for the sake of simplicity, we somewhat deviate from the definitions given there).

A TM has a finite control consisting of a set Q of states, |Q| constant, an input tape and a work tape, each tape with its corresponding read/write head. The tape symbols are taken from an alphabet $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_{|\Sigma|}\}$, for constant $|\Sigma|$. A single step of the TM consists of any or all of the following operations: change the state of the finite control, read the symbols pointed to by either the input or the work heads, print a new symbol at the location pointed to by the work head, and move the tape heads, independently, one cell left (L) or right (R) or keep them stationary (S).

The TM is formally denoted by a tuple $\langle Q, \Sigma, \delta, b, q_0, q_a, q_r \rangle$, where $b \in \Sigma$ is the *blank* symbol, $q_0 \in Q$ is the initial state, $q_a, q_r \in Q$ are the *final states* in which the machine terminates its computation when the input string is accepted or rejected, respectively, and $\delta : Q \times \Sigma^2 \mapsto$ $Q \times (\Sigma \times \{L, R, S\})^2$ is the *next-move* function.

A descriptor of the TM is a four-tuple $d = \langle q, W, i, w \rangle$, where $q \in Q$ is the finite-control state, W is the contents of the (entire) work tape, and i (respectively, w) is the tape location to which the input (resp., work) head is pointing. Let M be a TM with an input of size N and a work tape of size f(N). The number of different valid descriptors for M is bounded (for some constant $c \geq 1$) by

$$|Q| \cdot N \cdot f(N) \cdot |\Sigma|^{f(N)} = O(N \cdot c^{f(N)}) .$$

$$\tag{1}$$

Suppose a TM M assumes a descriptor d at the beginning of a certain step t of some computation. Let $\hat{\delta}$ denote the specialization of the next-move function, δ , obtained by fixing the contents of the work tape and the finite-control state according to d. There are at most $|\Sigma|$ descriptors that are possible values of $\hat{\delta}$, i.e., descriptors of the beginning of the next step, depending on the contents of the input tape in the location pointed to by the input head. Similarly, there were at most $9 \cdot |Q| \cdot |\Sigma|$ valid descriptors for M at the beginning of the previous step, as either of the heads may have moved, and a single location of the work tape and the finite-control state may have changed.

5.2 Space Bounded TM's and Size Bounded RN's

The main relation between RN's and TM's is expressed in the following lemma, which is proved below. Here, L is the set of problems solvable by a deterministic TM having $O(\log N)$ workspace.

Lemma 5.1 There exists a constant c > 0 such that for every constructible f(N),

 $SPACE(f(N)) \subset RN(O(1), c^{\max(f(N), \log N)})$.

Putting $f(N) = O(\log N)$, Lemma 5.1 implies

Theorem 5.2 $L \subseteq RN(O(1), poly(N))$.

In particular, all logspace reductions are carried in constant time in the RN model using a polynomial number of switches. This will be useful later when we consider the class *PTIME* and its relation to the NMRN model. We can further generalize Lemma 5.1 and drop the constructibility restriction for TM's with high space requirements.

Lemma 5.3 There exists a constant c > 0 such that for every $f(N) = \Omega(N)$

 $SPACE(f(N)) \subseteq RN(O(1), c^{f(N)})$.

Proof: The value f(N) for a certain problem A may be computed by running the corresponding TM T_A over all inputs of length N, counting the size of the work space and evaluating the maximum. This procedure takes only $O(N + \log f(N))$ additional work space, thus f(N) is constructible by definition, hence by Lemma 5.1, $A \in RN(O(1), c^{f(N)})$.

Lemma 5.3 implies a universality result for the RN model. The same result was previously shown for the Bus Automata model [Rot76].

Corollary 5.4 For every decidable problem A there exists a family of RN's solving it in constant time.

Proof: The maximum (over all inputs of size N) work space f(N) that is used by the TM T_A for solving A is finite (though not necessarily constructible). Now use Lemmas 5.1 and 5.3.

Note that circuits are also a universal model, hence the results from [BPRS91] that are reviewed at Section 4 imply a stronger version of Corollary 5.4, namely a universality result for the LRN model.

Corollary 5.5 For every decidable problem A there exists a family of LRN's solving it in constant time. \blacksquare

The remainder of this subsection is dedicated to proving Lemma 5.1. Let us first restate the problem.

We are given a TM, $M = \langle Q, \Sigma, \delta, b, q_0, q_r, q_a \rangle$, solving a problem A while using a work space of size O(f(N)), for a constructible f(N). Let us denote by M_f the TM that, given N, produces f(N) using O(f(N)) space.

We need to show the existence of a uniformly generated family of networks R_N in the RN model, where for every $N \ge 1$, R_N solves A for all input instances of size N, and R_N is of size $O(c^{f(N)})$ for some constant c.

Proof of Lemma 5.1:

The proof is constructive. That is, we show a TM U, that receives N as its input, computes f(N) by emulating M_f , and generates the description $\mathcal{D}(R_N)$ of the network R_N as its output while using O(f(N)) space.

The program of R_N consists of two steps: initialization and computation. Let us first describe only the part of the network that corresponds to the computation step, and remark on the modifications necessary for the first (initialization) step at the end.

For each descriptor d of the TM M, the network R_N contains a corresponding switch, denoted s(d). By Eq. (1), the number of different descriptors (hence the size of R_N) is bounded by $O(Nc^{f(N)})$ for some constant c.

An edge connecting two switches s(d), s(d'), represents an allowable transition of M between the corresponding two descriptors d, d'. In the computation step, the switch settings are given by δ , the next-move function of M. The function δ is encoded for each switch s(d) in its set Rules^d of configuration rules. Thus, Rules^d specifies the next descriptor γ_l^d to be assumed by M, depending on the input symbol $\sigma_l \in \Sigma$ found on the input tape at the location of the input head while M assumes d.

The description $\mathcal{D}(R_N)$ output by U consists of a list of triplets $(d, \Gamma^d, \mathtt{Rules}^d)$, one for each descriptor d of M. The neighborhood relation $\Gamma^d = (\Gamma^d_{out}, \Gamma^d_{in})$ is determined by δ as follows. The set Γ^d_{in} contains all the descriptors from which d may result in a single step of M. The list

 $\Gamma_{out}^d = \{\gamma_1^d, \dots, \gamma_{|\Sigma|}^d\}$ contains the descriptors which may follow d at the next step, depending on the input symbol found at the location of the input tape when M assumes d. Note that both lists contain a constant number of elements, hence R_N is bounded-degree.

We may view the edges of R_N as though they were directed, since if y is a neighbor of x and appears in Γ_{out}^x , then x is not in Γ_{out}^y . In this case, we say that the edge (x, y) is virtually directed towards y and virtually directed out of x. Clearly, $\Gamma_{out}^d \cap \Gamma_{in}^d = \emptyset$, since otherwise an "infinite loop" may occur in some computation, contradicting the assumption that M always stops after a finite number of moves.

In order for U to generate the description $\mathcal{D}(R_N)$, it keeps a counter for the descriptors. For each descriptor d, U generates the next descriptors $\gamma_1^d, \dots, \gamma_{|\Sigma|}^d$, where γ_i^d is generated by emulating M (and its next-state function, δ), starting from the machine configuration given by d, and the input symbol $\sigma_i \in \Sigma$. This also gives the encoding for \mathtt{Rules}^d at the computation step, i.e., the configuration rules for the corresponding switch at that step. The set Γ_{in}^d is generated for d in a gradual manner, by adding a new entry d' whenever discovering a descriptor d' from which d may have resulted.

The total space used by U is O(f(N)) for computing f(N) by emulating M_f , and $O(\log(\zeta(M)))$ for handling the descriptor counter, where $\zeta(M)$ is the number of different descriptors assumed by M. Thus, by Eq. (1), the total work space used by U is bounded by $O(f(N) + \log N)$.

Finally, the description of R_N includes also a part concerning the initialization step. In particular, for every descriptor d, the neighborhood relation Γ^d contains also a set of edges I^d , consisting of edges to two descriptors PREV(d) and SUCC(d), the "previous" and "next" triplets in the description of the RN. That is, we assume that the triplets are generated in batches having the same input head position. Thus the previous and the next triplets always have the same input head location (except for "boundary cases" such as $\langle q_0, \vec{b}, i, 1 \rangle$, where \vec{b} denotes the work tape full of blanks). The set **Rules** contains also the configuration rules for the initialization step, to be described directly later.

Suppose R_N is constructed from a description generated by U as described above. It remains to show how it computes A, given an input I of size N.

Algorithm TM_SIMULATION:

Initialization Step: At this step, each switch s(d) connects its I^d edges. Consequently, a linear bus is formed, connecting all descriptors with the same input head location. The network R_N is configured into N connected components, each consisting of all switches corresponding to descriptors having some fixed location of the input head. The j'th input appears at the switch $s(d_{(j)})$ representing the descriptor $d_{(j)} = \langle q_0, \vec{b}, j, 1 \rangle$. The switch $s(d_{(j)})$ transmits the j'th input symbol to the rest of the switches on the linear bus to which it is connected.

Computation Step: Suppose a switch s(d) received an input symbol $\sigma_j \in \Sigma$ at the initialization step. Then, during the computation step, s(d) connects all its neighbors from Γ_{in}^d together with $\gamma_j^d \in \Gamma_{out}^d$. All other neighbors remain disconnected. After the configuration is set, the switch $s(d_0)$ corresponding to the descriptor $d_0 = \langle q_0, \vec{b}, 1, 1 \rangle$ transmits a signal on the bus it is connected to.

Claim 5.6 The signal transmitted by $s(d_0)$ at the computation step is detected by a single switch corresponding to a final state.

Proof: It is rather straightforward to show that the signal is detected by the switch corresponding to the final state which is reached by M on the given input. Informally, a sequence of valid moves of M induces a connected path in R_N .

It remains to be shown that the signal can not be detected by any other "final switch". Consider s_f , a switch corresponding to a final state q_f which is not the one reached by M on the given input. Assume by contradiction that s_f detects the signal. Thus during the computation step, there is a path E in R_N (having no loops) connecting $s(d_0)$ to s_f . Going along E from $s(d_0)$ to s_f , let s_l be the last switch corresponding to a descriptor which was assumed by M during its computation on the given input.

The state q_f is final, so the edge in E connecting to s_f must be virtually directed towards s_f . Since there is at most one edge that is both virtually directed out of a switch and is connected at the computation step, then by induction all edges along E are "virtually directed from s_l towards s_f ". However this implies that there are two edges virtually directed out of s_l (one along E and the other along the computation path taken by M), which are connected during the computation step, a contradiction.

Having the claim we conclude that at the end of the second step the result is known to both switches corresponding to final states, and may further be broadcast at successive steps. This concludes the proof of Lemma 5.1.

5.3 Logspace TM's and Size Bounded LRN's

Let us next relate linear RN's to space bounded TM's. The main result of this section is the equivalence of L and LRN(O(1), poly(N)). This is proved in the following two theorems.

Theorem 5.7 $LRN(O(1), poly(N)) \subseteq L$.

Proof: Let A be a problem solvable by the family $\mathcal{R}_A = \{R_N\}_{N\geq 1}$ in the LRN model in d = O(1) steps. The size of the network R_N , solving all inputs of length N to A, is bounded by some polynomial S(N). Let M_A^0 be a TM that, given N, outputs a description of R_N while using at most log $S(N) = O(\log N)$ work space. A minor modification of M_A^0 yields a logspace machine \widehat{M}_A^0 that, when given N and $i, 1 \leq i \leq S(N)$, outputs the description of the *i*'th switch in R_N including its local configuration during the first computation step.

In general, let M_A^j denote a logspace machine which, given some input I of size N, outputs the configuration description taken by R_N at the (j+1)'st step on the input I. The description includes the local configuration taken by every switch at the (j+1)'st step and the contents of the switch buffers at the beginning of that step. Note that given M_A^j , it is easy to construct a logspace machine \widehat{M}_A^j that when given N, I and i (for some $1 \leq i \leq S(N)$), outputs the configuration of the *i*'th switch of R_N at the (j+1)'st step of R_N 's computation on the input I. Finally, observe that presenting a logspace machine M_A^d completes the proof, as the description includes also the contents of "output buffers". Consider some configuration H of a network R in the LRN model. Given as input the configuration description of H together with the id's of two nodes s and t in R, we construct a logspace machine, M_{reach} , solving the question whether s is connected to t in H. The machine M_{reach} keeps several pointers to the input. For every exit from s, the machine visits node by node the whole linear bus determined in H by this exit. Each node x reached by M_{reach} is compared to t, and the next node of the bus is determined by the local configuration data of x.

Having defined M_A^j and \widehat{M}_A^j for $j = 1, 2, \dots, d$ and M_{reach} , the theorem is proved by induction on d, the number of steps of R_N . We construct a logspace machine M_A^d which, given N and any input instance I of size N, solves A by emulating R_N on I. The machine M_A^d outputs the description of the configuration and the contents of buffers of R_N at the beginning of step d+1(if the computation terminates at step d then only the contents of the buffers is important).

Clearly we have M_A^0 . Suppose that we have constructed M_A^{d-1} By the above discussion, we also have \widehat{M}_A^{d-1} . The machine M_A^d uses both logspace machines M_{reach} and \widehat{M}_A^{d-1} in order to determine for each switch s its local configuration at step d. This is accomplished by iterating over all switches of R_N . For each switch t we iterate over all other switches s, using M_{reach} to determine if s and t are connected. If s transmits a message during step d and is connected to t then the message is written into the buffers of t. M_{reach} uses \widehat{M}_A^{d-1} in order to obtain the local configuration of switches at step d-1 of the emulated network R_N .

Note that the construction of M_A^d uses \widehat{M}_A^{d-1} rather than M_A^{d-1} . Thus the computation uses only $O(\log N)$ cells of the work tape for each level of the (depth d) recursion.

Theorem 5.8 $L \subseteq LRN(O(1), poly(N)).$

The theorem will be proved by using the following lemmata.

Definition 5.9 CYCLE is the following decision problem. The input is a permutation on N vertices, i.e. a directed graph of out-degree 1 (given by its adjacency matrix), with two special vertices a and b. The answer is '1' if a and b are on the same cycle.

Lemma 5.10 [CM87] CYCLE is complete for L with respect to NC^1 reductions.

Lemma 5.11 $CYCLE \in Mesh_LRN(O(1), N, N)$.

Proof: Let the (i, j)'th switch get the (i, j)'th bit in the input adjacency matrix. Thus there is precisely one set bit in the input that is associated with the j'th column of the mesh (for all $1 \leq j \leq N$), indicating that j is moved to k in the input permutation. During the first step, this information (i.e., k) is transmitted by (k, j) to all the switches of the j'th column. During the second step, this information is transmitted by (j, j) to all the switches of the j'th column. Intuitively, after the third step, j is transmitted by (j, k) to all the switches of the k'th column. Intuitively, after the third step each switch at the j'th column knows both the element to which j is moved and the element that is moved to j in the input permutation.

Let a and b be given as input to node (1,1). This node transmits them to all the switches (0,*) on the top row of the mesh in the fourth and the fifth steps. During the sixth step this information is also transmitted on the a'th and the b'th columns.

Assume that in the input permutation, j is moved to k and l is moved to j. Consider the switches of the j'th column, all of which know about k and l. During the seventh step all the switches of the j'th column configure (U - D, L - R), except for (l, j) and (j, j), whose configurations, Conf(l, j) and Conf(j, j), are defined as follows.

$$\begin{cases} Conf(l,j) = (L-D) \text{ and } Conf(j,j) = (U-L), & \text{if } l < j \text{ and } k < j, \\ Conf(l,j) = (L-D) \text{ and } Conf(j,j) = (U-R), & \text{if } l < j \text{ and } k > j, \\ Conf(l,j) = (R-U) \text{ and } Conf(j,j) = (D-R), & \text{if } l > j \text{ and } k < j, \\ Conf(l,j) = (R-U) \text{ and } Conf(j,j) = (D-L), & \text{if } l > j \text{ and } k > j. \end{cases}$$

Claim 5.12 As a result of the above setting, each cycle in the input permutation induces a cycle in the mesh.

Proof: The node (l, j) directs a signal coming on the *l*'th row to (j, j). In turn, (j, j) forwards this signal to (j, k), where it is forwarded to (k, k), and so on, until the signal reaches (l, l) which closes the cycle by directing it back to (l, j).

During the seventh step, switches (a, a) and (b, b) transmit some (arbitrary) signal on the bus in which their two connected edges take part. They also listen on that bus. An error state occurs iff a and b are in the same cycle of mesh edges, indicating a '1' answer to the *CYCLE* decision problem.

The answer can be broadcast during the eighth step to all other switches of the mesh. This completes the proof of Lemma 5.11.

Using Lemma 5.11 we can now complete the proof of Theorem 5.8.

Proof of Theorem 5.8: Lemma 5.11 implies that $CYCLE \in LRN(O(1), N^2)$. To complete the proof of the theorem, we only need to show that the (NC^1) reduction from any logspace problem can also be done in the LRN model with a polynomial size network and in constant time. This follows from the general fact that LRN's are closed under composition. Specifically, let $g : \{0,1\}^N \mapsto \{0,1\}$ be a Boolean function computed by a logspace Turing machine. By Lemma 5.10 there is a polynomial p(N) = m and functions $y_1, ..., y_m$, each on $x_1, ..., x_N$, so that $g(x_1, ..., x_N) = CYCLE(y_1, ..., y_m)$. If we have the Boolean values $y_1, ..., y_m$ at the appropriate size then by Lemma 5.11 we are done.

Let us assume w.l.o.g. that the reduction produces the adjacency matrix of the graph which is the input to CYCLE. Then the size of the mesh computing CYCLE, call it \mathcal{M} , is about $\sqrt{m} \times \sqrt{m}$. Each value y_i is an NC^1 function and thus, by Lemma 4.4, can be computed by a universal polynomial size mesh in constant time. Let \mathcal{M}_i denote the universal mesh computing y_i , and let s_i denote the switch in \mathcal{M} in which y_i is expected as input when \mathcal{M} computes y_i . All that remains to be done is to connect s_i to \mathcal{M}_i and let \mathcal{M}_i compute y_i in an initialization phase. It is left to move the inputs to all the meshes \mathcal{M}_i . By Lemma 4.4, all of these meshes are in the form of rectangles of width N, where the *i*th row is expecting the *i*th input bit. Thus it is straightforward to connect all of them for distributing the input.

Finally, note that the description of the whole construction is uniformly generated by a logspace Turing machine. The only subtlety here is that when writing the description of the

universal mesh \mathcal{M}_i that is to compute y_i , the set **Rules** of configuration rules corresponds to the function $y_i = f_i(x_1, ..., x_N)$. More specifically, each switch should have two possible configurations, out of which it chooses (in the computation step, see Lemma 4.4) according to one of the input bits. Fortunately, the logspace Turing machine which writes the description of \mathcal{M}_i can determine these configurations when given the description of the circuit computing y_i [BPRS91, Cle90, CL89]. The circuit itself is uniformly generated by a logspace machine (see Sec. 4).

5.4 Symmetric TM's and RN's

A concept intermediate between determinism and nondeterminism is symmetry introduced by Lewis and Papadimitriou. A symmetric Turing machine is a nondeterministic one in which each transition may be executed in both directions: forwards and backwards. For the technical details we refer to [LP82] The corresponding logspace complexity class is denoted by SL. The main result of this section will be that RN(O(1), poly(N)) conincides with SLH, the Symmetric Logspace Oracle Hierarchy which we introduce below.

The reachability problem for directed graphs is complete for NL ([Sav70]). If we restrict this problem to undirected graphs, we get a problem, complete for symmetric logspace.

Definition 5.13 s-t CONNECTIVITY (or UGAP), is the following decision problem. The input is an undirected graph \mathcal{G} on N vertices (given by its adjacency matrix), with two special vertices s and t. The answer is '1' if s and t are in the same connected component in \mathcal{G} .

Lemma 5.14 ([LP82]) The s-t CONNECTIVITY problem is complete for SLwith respect to NC^1 -reductions.

There are two standard mechanisms to construct hierachies over complexity classes: bounded alternation and Turing reducibilities, i.e.: the use of oracle machines. In the case of nondeterministic space classes the resulting hierarchies both collapse on the first level and coincide with the original nondeterministic class (see [Imm88, Sze88]). For symmetric space the hierarchy based on alternation was introduced in [Rei84]. In the following we shortly introduce an oracle based analogue and then prove that it coincide with RN(O(1), poly(N)).

There are two main possibilities to relativize space bounded classes, i.e.: to equippe space bounded machines with an oracle mechanism: In *LL relativization*, the approach of Ladner and Lynch ([LL76]), the machine may use all of its power to generate oracle queries, while in *RST relativization*, the approach of Ruzzo, Simon, and Tompa, the queries have to be generated deterministically ([RST84]). The later is equivalent to a model, where the oracle machine does not generate any query, but simply gives its current instantaneous description to the oracle, which in this model also has access to the input word of the base machine. That means that an oracle set *B* is now a subset of $\{(d, I) \in X^* \times X^* | /d/ = \log(/I/)\}$. In this form we can carry over RST relativization to symmetric space: certain states of the finite control of the symmetric base machine are designated as *query states*. With each query state *q* there are associated two *answer states* q_+ and q_- . *Query descriptors* are those containing a query state. Given an oracle set *B* as above, an input word *I*, and a query descriptor d = (q, W, i, w), where *q* is a query

begin of changes state, we regard d to be symmetrically connected with $d_+ := (q_+, W, i, w)$ if $(d, I) \in B$, and with $d_- := (q_-, W, i, w)$ in case of $(d, I) \notin B$. For a symmetric logspace bounded oracle machine M and an oracle set B we denote the set of all words accepted by M with oracle B by $L\langle M, B \rangle$. As usual, the use of parentheses is reserved for the LL mechanism, while the use of the RST relativization is indicated by using angles. Further on, let $SL^{\langle B \rangle}$ be the set of all languages accepted by symmetric logspace bounded oracle machines with oracle B, and for a class \mathcal{B} set $SL^{\langle B \rangle}$ be the union over B in \mathcal{B} of all $SL^{\langle B \rangle}$. Using ideas of the proof of Theorem 5.8 it is possible to show the following analogue of the nondeterministic case, which we state without proof:

Proposition 5.15 For each oracle set B we have

$$SL^{\langle B \rangle} = SL^{\langle LOG(B) \rangle} = SL^{\langle L^B \rangle}$$

We are now in the position to define the symmetric logspace oracle hierarchy:

Definition 5.16 a. $O\Sigma_0^{SL} := SL$ and $O\Sigma_{k+1}^{SL} := SL^{\langle O\Pi_k^{SL} \rangle}$ for nonnegative k. **b.** $O\Pi_k^{SL} := \{X^* \setminus B \mid B \subseteq X^*, B \in O\Sigma_k^{SL}\}$ for nonnegative k.

c. $SLH := \bigcup_k O\Sigma_k^{SL}$.

We mention in passing, that the whole symmetric logspace alternation hierarchy is contained in L^{SL} which is a subset of $O\Sigma_2^{SL}$. This resembles exactly the situation in the nonderministic case before the result of Immerman and Szelepcsenyi ([RST84, Imm88, Sze88]).

Based on this definition we can state the first half of the main result of this subsection:

Theorem 5.17 a. For each positive integer k we have $RN(k, poly(N)) \subseteq L^{O\Sigma_k^{SL}}$ and hence **b.** $RN(O(1), poly(N)) \subseteq SLH$

Proof: The proof of part a. follows via induction over the running time k of the RN: For k = 1 we have to show $RN(1, poly(N)) \subseteq L^{SL}$. To simulate one step of an RN $\mathcal{R} = \{R_N\}_{N\geq 1}$ on an input I of length N, a deterministic logspace base machine M, repeatedly simulating the logspace machine U describing the circuit, can go through all pairs (s, t) of nodes of R_N and by asking an oracle from SL, can find out whether s and t are connected. In this way M can detect the global state of R_N reached after one step and hence can decide whether I is to be accepted.

Now lets assume the statement to hold for RN's of running time k and let $\mathcal{R} = \{R_N\}_{N \ge 1}$ be an RN of running time k + 1 recognizing a language $A \subseteq X^*$. By the induction hypothesis we know that the outcome of R_N after k steps of computation is representable as an element of $L^{O\Sigma_k^{SL}}$. By the construction of this proof it will follow that this pertains to the whole global situation of R_N after k steps. Thus the set *Configuration*(k) of all local configurations and the contents of the switch buffers, is an element of $L^{O\Sigma_k^{SL}}$. Now, let us consider the sets *Neighbour*(k):= $\{\langle x, y \rangle \mid \text{after k steps switch } x \text{ is a direct neighbour of switch } y \}$ and $Transneighbour(\mathbf{k}) := \{\langle x, y \rangle \mid \text{after k steps switch } x \text{ is a transitive neighbour of switch } y \}$. Obviously, a deterministic logspace machine can recognize $Neighbour(\mathbf{k})$ when it has access to the oracle set $Configuration(\mathbf{k})$ That is, we have $Neighbour(\mathbf{k}) \in L^{Configuration(k)}$. The transitive closure $Transneighbour(\mathbf{k})$ of the symmetric relation $Neighbour(\mathbf{k})$ can now be recognized by a symmetric logspace machine when given $Neighbour(\mathbf{k})$ as an oracle. Using Proposition 5.15 we get $Transneighbour(k) \in SL^{\langle Neighbour(k) \rangle} \subseteq SL^{\langle L^{Configuration(k)} \rangle} = SL^{\langle Configuration(k) \rangle} \subseteq O\Sigma_{k+1}^{SL}$. Now, similar to the case k = 1, a logspace base machine can recognize Configuration(k + 1) if it has oracle access to Configuration(k) and Transneighbour(k). But the set $Configuration(k) \cup Transneighbour(k)$ is an element of $O\Sigma_{k+1}^{SL}$, since this class is closed under union. In total, we have $Configuration(k + 1) \in L^{O\Sigma_{k+1}^{SL}}$ and hence $RN(k + 1, poly(\mathbf{N})) \subseteq L^{O\Sigma_{k+1}^{SL}}$.

Remark: It is up to now not known whether SL is closed under complement. If this should be the case the previous theorem could be strengthened to $RN(O(1), poly(N)) \subseteq SL$.

Nisan showed in [Nis92] that the probabilistic class RL is contained in $SC^2 := DTISP(\text{pol}, \log^2 n)$. Here SC denotes Steven's class (see e.g. [KR90]) of all problems computable by polynomial time, polylog space bounded Turing machines. As consequences we get:

Corollary 5.18 $RN(O(1), poly(N)) \subseteq SC^2$

Proof: It is sufficient to show that $SL^{SC^2} \subseteq SC^2$, i.e.: that SC^2 is closed under symmetric logspace Turing reducibilities. But given a conditional instance A of UGAP, that is an adjacency matrix of a symmetric relation where the entries are membership problems in SC^2 , we can simulate Nisan's algorithm on the UGAP instance; each time this algorithm tries to read from the input matrix, we first solve the SC^2 -problem encountered in the corresponding entry of the input.

In a simular way, it is possible to show that the containment of SL in $DSPACE(\log^{1.5} n)$ (see [NSW92]) yields:

Corollary 5.19 $RN(O(1), poly(N)) \subseteq DSPACE(\log^{1.5} n).$

In the second half of this subsection we will prove the converse of Theorem 5.17.

Theorem 5.20 $SLH \subseteq RN(O(1), poly(N))$

The proof will be done by proving $O\Sigma_k^{SL} \subseteq RN(O(1), poly(N))$ via induction over k. To show the case k = 1, i.e.: $SL \subseteq RN(O(1), poly(N))$, we need the following lemma stating that *s*-*t CONNECTIVITY* can be decided on the mesh in a constant number of steps. This result has been established before by Wang and Chen [WC90a]. However, the proof we give next may be of independent interest, since it shows how to solve the problem on an N^2 -switch $N \times N$ mesh (the original proof used an N^4 -switch $N^2 \times N^2$ mesh, or an N^3 -switch non-mesh reconfigurable network).

Lemma 5.21 s - t CONNECTIVITY $\in Mesh_RN(O(1), N, N)$.

Proof: We prove the lemma by exhibiting a program for solving the s-t connectivity problem on the mesh. Let the (i, j)th switch get as input the (i, j)th bit in the adjacency matrix of the input graph. During the first two steps, the identity of s and t which is input, say, at switch (0,0) is broadcast on the upper most row (0,*). During the third step, all processors having a '0' as their input bit (from the adjacency matrix) connect (U - D, L - R), and all those having a '1' and all "diagonal" switches ((i,i) for all $0 \le i \le n-1)$ connect (U - L - D - R). Then, processors (0,s) and (0,t) transmit some signal on the bus connected to their D port.

Claim 5.22 The connected components in the input graph relate in a one-to-one fashion to the connected components of the global configuration that is assumed by the mesh during the third step.

Proof: Consider the *i*th column of the mesh during the third step. This column connects to precisely all rows j such that i and j are neighbors in the input graph. A similar observation is true for the *i*th row. By induction, the claim follows.

Having Claim 5.22 we conclude that during the third step, (0, s) and (0, t) read an error state on the bus which is connected to their D port, iff they are connected in the input graph. This information can be broadcast to all mesh switches during the fourth step. Thus the outlined program computes s - t connectivity, completing the proof of Lemma 5.21.

The next corollary now follows from combining Lemma 5.14 with Lemma 5.21, or alternatively the results of [WC90a].

Corollary 5.23 $SL \subseteq RN(O(1), poly(N))$.

Proof: Lemma 5.21 implies that s - t CONNECTIVITY is in $RN(O(1), N^2)$. By Lemma 5.14 s - t CONNECTIVITY is complete for SL with respect to NC^1 reductions. Thus, we only need to show that the reduction from any problem in SL can also be done in the RN model with polynomial size and in constant time. The details and the construction are very much the same as in the proof of Theorem 5.8, and are therefore omitted.

Proof of Theorem 5.20: To conclude the proof we have to show the induction step from k to begin of k+1. So we assume $O\Sigma_{k'}^{SL}$ to be a subset of RN(O(1), poly(N)) for every $k' \leq k$. Further on, let M be a symmetric logspace oracle machine and let B be an element of $O\Sigma_{k}^{SL}$. We have to show that $A := L\langle M, B \rangle$ can be recognized in O(1) steps by a polynomially sized RN $\mathcal{R} = (R_N)_{N>1}$.

By the induction hypothesis there exists an RN $\mathcal{R}_0 = (R_{0,N})_{N \geq 1}$ recognizing B in some k steps. In the following, let I be an arbitrary but fixed input to M of length N. All configurations and descriptors considered from now on will be of size or length log N.

We will now design an RN R_N simulating M with oracle B on inputs of length N. R_N will consist in *base switches* and *oracle switches*. For each descriptor d of M we use a base switch s(d). With each query descriptor d of M we associate a copy $R_{0,N}^d$ of $R_{0,N}$, built of oracle switches. The details of the interconnection of these switches are very similar to those of Lemma 5.1. The triplets $(d, \Gamma^d, \operatorname{Rules}^d)$ specifying the switch s(d) corresponding to the descriptor d, now contains edges to all switches which correspond to descriptors which are

possibly reachable by d in one step given an arbitrary input. Observe that these edges are undirected, since M is a symmetric TM.

The steps performed by R_N are as follows:

- Step 1: Initialization: distribute to each switch of R_N its required input bits of I. These could be up to three, due to the technical details used in symmetric machines.
- **Steps** 2,..., \hat{k} + 1: The oracle switches in all subnetworks $R_{0,N}^d$ compute in \hat{k} steps whether $\langle d, I \rangle$ is an element of B or not. If this is the case, $R_{0,N}^d$ establishes a connection between the base switches s(d) and $s(d_+)$. Otherwise, s(d) and $s(d_-)$ are connected by $R_{0,N}^d$.
- Step $\hat{k} + 2$: In this last step the interconnection of the switch is done according to the transition structure of M on input I: the base switches switch all those edges which correspond to transitions which are consistent with the input I. Then the switch $s(d_0)$ corresponding to the initial descriptor $d_0 := \langle q_0, \vec{b}, 1, 1 \rangle$ transmits a signal on the bus it is connected to. Obviously, M accepts its input I with oracle B, if and only if the switch corresponding accepting end descriptor detects this signal.

We finally mention that the construction of \mathcal{R} is sufficiently uniform to be describable by a deterministic logspace machine.

As a consequence of Theorems 5.17 and 5.20 we get the main result of this subsection:

Corollary 5.24 RN(O(1), poly(N)) = SLH

end of changes

5.5 Non-Deterministic TM's and DRN's

The next-move mapping δ of a non-deterministic TM may have several choices for the next machine configuration, given a certain descriptor. The non-deterministic TM accepts its input if there exists any sequence of choices of moves that leads to an accepting state. In particular, NL is the set of problems solvable by a non-deterministic TM having $O(\log N)$ workspace. The main result of this section is that NL = DRN(O(1), poly(N)).

Lemma 5.25 There exists a constant c > 0 such that for every constructible f(N),

 $NSPACE(f(N)) \subseteq DRN(O(1), c^{\max(f(N), \log N)})$.

Putting $f(N) = O(\log N)$, Lemma 5.25 implies

Theorem 5.26 $NL \subseteq DRN(O(1), poly(N))$.

Proof of Lemma 5.25: The proof starts with a construction of a TM U, similar to the one described in the proof of Lemma 5.1. The machine U uses M, the non-deterministic Turing

machine solving a problem A, in order to construct for every N a DRN R_N solving A on all inputs of size N. The description of DRN's is similar to the one given in the proof of Lemma 5.1 for RN's, except that the triplets $(d, \Gamma^d, \operatorname{Rules}^d)$, specifying the switch s(d) corresponding to the descriptor d in the description of the DRN, include directions for the edges, and that there may be several relevant out-edges for each input symbol and Turing machine configuration (descriptor). As before, we set $\Gamma^d = (\Gamma^d_{out}, \Gamma^d_{in}, I^d)$, with the following changes. The set Γ^d_{in} contains s(d)'s in-edges, and Γ^d_{out} is the set of s(d)'s out-edges. $\Gamma^d_{out} = \gamma_1^d \cup \cdots \cup \gamma_{|\Sigma|}^d$, where γ_j^d is the set of possible moves when the input symbol at the input head location (specified by the descriptor d) is the j'th symbol in Σ , namely σ_j . As before, I^d contains the pair of descriptors PREV(d) and SUCC(d), directed into and out of the switch, respectively. As in the undirected case, for any switch $s(d), \Gamma^d_{out} \cap \Gamma^d_{in} = \emptyset$.

Let R_N be a DRN constructed according to the description output by U when given the input N. Again, we have an initialization step and a computation step.

Algorithm NTM_SIMULATION:

The initialization step is the same as the one from the Algorithm TM_SIMULATION in the proof of Lemma 5.1.

Computation Step: Suppose a switch s(d) received an input symbol $\sigma_j \in \Sigma$ at the initialization step. Then, during the computation step, s(d) connects all the in-edges from Γ_{in}^d to all the out-edges from γ_j^d . All other out-edges remain disconnected.

After the configuration has been taken, the switch $s(d_0)$ corresponding to the descriptor $d_0 = \langle q_0, \vec{b}, 1, 1 \rangle$ transmits a signal on the bus it is connected to.

Claim 5.27 Given an input instance I of size N, the signal transmitted by $s(d_0)$ at the computation step reaches a switch s(d) if and only if there is a sequence of choices of moves of M on I reaching the descriptor d.

Proof: We prove one direction of the claim by induction. Suppose that, given the input I, there is a sequence of l moves by M leading to d. For l = 0 we have $d = d_0$. For l > 0, suppose the claim holds for all switches corresponding to sequences of at most l - 1 moves. Consider a switch s(d') corresponding to a descriptor d' of M. The descriptor d' is reachable in l - 1 moves of M on I, and d is reachable from d' in a single move, given the input I. By the assumption, the signal transmitted by $s(d_0)$ reaches s(d'). By the definition of the DRN model and the construction of R_N , there is an edge e directed from s(d') to s(d). The edge e is connected to all the in-edges of s(d'), given the input I. Thus the signal reaches s(d), too.

As for the other direction, if the signal reaches s(d) then there is a (directed) path $P = e_1, e_2, \dots, e_l$ from $s(d_0)$ to s(d). Each edge along the path corresponds to a valid move of M on I, hence P corresponds to a sequence of such moves reaching d.

In particular, Claim 5.27 implies that there is a switch s_a corresponding to a final accepting state q_a which detects the signal if and only if there is a sequence of choices of moves by M on I leading to q_a . This concludes the proof of Lemma 5.25.

It is important to note that Claim 5.27 does not hold when the undirected RN model is used. This is because the signal transmitted by $s(d_0)$ may take in-edges in the "opposite" direction and hence reach switches corresponding to descriptors that may not be reached by M in a valid sequence of moves. This is actually an inherent problem of the RN model; when taking some configuration, the switch can not control the exits that an incoming signal may take. This observation may conceivably lead to upper bounds on the computational power of the RN model in order to "separate" it from the DRN model.

Theorem 5.28 $DRN(O(1), poly(N)) \subseteq NL$.

Proof: As the proof is similar to that of Theorem 5.7, we describe here only the necessary modifications. Let A be a problem solvable by the family $\mathcal{R}_A = \{R_N\}_{N\geq 1}$ in the DRN model in d = O(1) steps. The "answer" to A is either a "1" or a "0". As before, we say that the machine outputs a 1 (respectively, 0) if it terminates in the *accept* state q_a (resp., the *reject* state q_r). A non-deterministic machine NM_A^d solving (all length-N inputs of) A may reject when given an input instance I (of length N) for which the answer is a 1. The machine should always accept when the answer is a 0. Moreover, NM_A^d should have a sequence of choices of moves leading to an accepting state if the answer is a 1.

The construction of NM_A^d is similar to that of M_A^d in the proof of Theorem 5.7, except that the logspace machine M_{reach} given there is replaced by a non-deterministic logspace machine NM_{reach} . Let H denote some configuration of a network R in the DRN model. When the configuration description of H together with the id's of two nodes s and t in R, are given as an input instance, NM_{reach} determines whether there is a (directed) bus in R, starting at s and reaching t. For three distinguished symbols 0, 1 and 2 in the alphabet Σ , SM_{reach} outputs 1 on its output tape if there exists a bus leading from s to t, 0 if no such bus exists, and 2 if the answer is not determined by the machine.

Given NM_{reach} , the construction of NM_A^d follows that of M_A^d , where calls to M_{reach} are replaced by calls to NM_{reach} . The only difference is that if NM_{reach} outputs a "2" value then NM_A^d enters the reject state immediately. Otherwise, the computation proceeds as described in the proof of Theorem 5.7.

It remains to show the construction of NM_{reach} . We observe that when each network node is replaced by a cluster of (a constant number of) graph nodes, each holding a single set of connected edges, then NM_{reach} actually addresses the *reachability* problem: "given two nodes s, t in a directed graph G, is there a (directed) path starting at s and leading to $t\Gamma$ ". Obviously, the original directed-network reachability problem is of the same complexity as the graph reachability problem. Reachability is known to be (complete) in NL, thus there is a non-deterministic logspace Turing machine NM_{yes} computing it. On the way to proving that NL = Co - NL [Imm88, Sze88], it is shown in [Imm88] that the opposite question, namely the directed *unreachability* problem, is also in NL. This is established by presenting a nondeterministic logspace Turing machine NM_{no} computing unreachability. Our machine NM_{reach} executes both NM_{yes} and NM_{no} on its input (viewed as the graph reachability problem). It then outputs 0 if NM_{no} accepts, 1 if NM_{yes} accepts, and 2 if both NM_{yes} and NM_{no} reject. Finally we note that, although no direct simulation is shown, the equivalence of NL and DRN(O(1), poly(N)) implies that $DRN(O(1), poly(N)) \subseteq CRCW^1$ (cf. [KR90]).

6 Non-Monotonic Reconfiguring Networks

This section investigates the power of the NMRN model. It is instructive to note that this model is more powerful than the commonly accepted parallel models (assuming $PTIME \neq NC$), even those equipped with a shared memory unit. The main result of this section is the following theorem.

Theorem 6.1 PTIME = NMRN(O(1), poly(N)).

One direction of the theorem is implied by the following lemma:

Lemma 6.2 [**BPRS91**] Any circuit of depth d and constant fan-in can be simulated by a non-monotonic RN in constant time and maximum bus length d.

Proof: The switches of the NMRN model have a complete set of operators $\{\neg, \lor\}$. It is thus straightforward to construct an RN whose underlying topology reflects the topology of the simulated circuit.

In particular, if the RN model allows for buses of polynomial length then every problem in *PTIME* can be simulated in this way in constant time, although the simulation is not "uniform" or universal, and each circuit needs to be simulated individually. Since any problem A in *PTIME* is solvable by a family of circuits C_A of polynomial depth, we immediately have the first direction of Theorem 6.1, namely, *PTIME* \subseteq *NMRN*(O(1), poly(N)).

Lemma 6.2 implies that the *Circuit Value Problem (CVP)* can be solved in constant time by a non-monotonic RN. However, this result has a somewhat non-uniform flavor. As a more illustrative example to the power of non-monotonic RN, let us consider the well studied problem of computing the *Lexicographically-First Maximal Independent Set (Lex-MIS)* in a given graph. This problem is often used as a canonical example for a *PTIME*-complete problem just as is the CVP. In [BPRS91] it is shown that Lex - MIS can be solved in constant time by a $\sqrt{N} \times \sqrt{N}$ mesh in the *NMRN* model.

Lemma 6.3 [BPRS91] $Lex - MIS \in Mesh_NMRN(O(1), \sqrt{N}, \sqrt{N}).$

From Lemma 4.8 and Theorem 5.2 we have

 $L \subseteq Mesh_RN(O(1), poly(N), poly(N)) \subseteq Mesh_NMRN(O(1), poly(N), poly(N))$.

Thus we finally conclude the following.

Theorem 6.4 $PTIME \subseteq Mesh_NMRN(O(1), poly(N), poly(N)).$

Proof: The problem is solved after the appropriate reduction to Lex - MIS.

The remaining direction in the proof of Theorem 6.1, namely

$$NMRN(O(1), poly(N)) \subseteq NMRN(poly(N), poly(N)) \subseteq PTIME$$
, (2)

involves a simple emulation of the networks and induction. Let $A \in NMRN(O(1), poly(N))$. Then A has a family $\mathcal{R} = \{R_N\}_{N \geq 1}$ in the NMRN model, which is uniformly generated by some Turing machine $T_{\mathcal{R}}$. The emulation begins by producing the network description $\mathcal{D}(R_N)$ for the appropriate N, by an application of $T_{\mathcal{R}}$. Given the initial description of the network (or given its description for any time t), and the input instance, a polynomial TM emulates the computation of every switch, producing a description of the network configuration at the beginning of the second step (resp., at the beginning of the (t + 1)'st step). Broadcasting messages at any step involves, say, a depth-first scan of the connected components.

This concludes the proof of Eq. (2), and hence Theorem 6.1.

7 Summary and Open Questions

Our results for the relations between reconfigurable complexity classes and parallel and traditional complexity classes are summarized in Figure 1. Established connections are drawn by arrows. Downward vertical arrows hold trivially and are omitted.

The importance of these relations is in indicating how hard a problem may turn to be. For example, consider the *Transitive Closure* (TC) problem. Wang and Chen [WC90a] showed that

$$TC \in RN(O(1), N^3)$$
; $TC \in Mesh_RN(O(1), N^2, N^2)$.

The TC problem is related to the s - t CONNECTIVITY problem. The solution of the TC for a graph G gives the answer for s - t CONNECTIVITY for all pairs of nodes s and t in G. According to our results showing $TC \in LRN(O(1), poly(N))$ implies $L = L^{SL}$, thus resolving a long-standing problem. Thus, although the constant time RN algorithm for TC is not very complicated, we expect a solution for TC in the (constant-time polynomial-size) LRN model to be either very difficult or impossible.

There are many other open questions. For example, the relation of the RN model to CREW PRAM models is not fully understood. It is also open whether

$$LRN(O(1), poly(N)) \stackrel{?}{\subseteq} Mesh_LRN(O(1), poly(N), poly(N))$$

The informal statement of these questions reflects the search for a deeper understanding of the role of switch operation in determining the power of the model. The results of this paper show that the answers correspond to complex issues in the theory of computational complexity.

Perhaps most important of all is the quest for more accurate complexity models for reconfiguring networks. Our more powerful RN classes, DRN and NMRN, are founded on the same basic underlying assumptions of the general model, and particularly, the assumption that propagation delay is independent of the bus lengths. This assumption may be unjustified for these classes, due to the use of active devices along the transmission paths traversed by messages. Consequently, further study of these stronger RN classes should attempt to formulate a more realistic set of complexity assumptions for them. The computational power of programs in these classes should then be re-evaluated under these more refined complexity assumptions.

Acknowledgements

We are grateful to Ilan Newman for helping us with the proof of Theorem 5.8. Independently begin of of us, observations related to Theorems 5.17 and 5.20 were recently made by Koji Nakano and change that the proof of Theorem 5.8. Independently begin of the second seco



Figure 1: Reconfigurable complexity classes (classes are constant-time and polynomial-size).

References

- [Bar86] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC¹. In Proceedings of the 18th Symposium on Theory of Computing, pages 1-5. ACM, 1986.
- [BPRS91] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. Journal of Parallel and Distributed Computing, pages 139–153, 1991. Special issue on Massively Parallel Computation.
- [BS91] Y. Ben-Asher and A. Schuster. Data gathering on reconfigurable networks. Technical Report Technical Report #696, Technion, October 1991.
- [CL89] J. Cai and R.J. Lipton. Subquadratic simulations of circuits by branching programs. In Proceedings of the 30th Symposium on Foundation of Computer Science, pages 568-573. IEEE, 1989.
- [Cle90] R. Cleve. Toward optimal simulations of formulas by bounded-width programs. In Proceedings of the 22nd Symposium on Theory of Computing, pages 271–277. ACM, 1990.
- [CM87] S.A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. Journal of Algorithms, 8:385–394, 1987.
- [GK89] J.P. Gray and T.A. Kean. Configurable hardware: A new paradigm for computation. In C.L. Seitz, editor, Proceedings of the 10th Caltech conference on VLSI, pages 279-295, Cambridge, MA, March 1989. MIT Press.
- [HU79] J.E. Hopcroft and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley Publishing Company, 1979.
- [Imm88] N. Immerman. Nondeterministic space is closed under complementation. SIAM J. on Computing, 17(5):935-938, 1988.
- [KR90] R.M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North Holland, Amsterdam, 1990.
- [LL76] R. Ladner and N. Lynch. Relativization of questions about log space computability. Math. Systems Theory, 10:19–32, 1976.
- [LM89] H. Li and M. Maresca. Polymorphic-torus network. IEEE Trans. on Computers, 38(9):1345-1351, 1989.
- [LP82] P. Lewis and C.H. Papadimitriou. Symmetric space-bounded computation. Theoret. Comput. Sci., 19:161–187, 1982.
- [MKS89] O. Menzilcioglu, H.T. Kung, and S.W. Song. Comprehensive evaluation of a twodimensional configurable array. In Proceedings of the 19th Symposium on Fault-Tolerant Computing, pages 93-100, Chicago, Illinois, 1989.
- [MPRS87] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout. Parallel computations on reconfigurable meshes. Technical Report TR IRIS#229, Dept. of Computer Science, University of Southern California, 1987.

- [MR79] J.M. Moshell and J. Rothstein. Bus automata and immediate languages. Information and Control, 40:88-121, 1979.
- [Nak87] T. Nakatani. Interconnections by Superposed Parallel Buses. PhD thesis, Princeton University, 1987.
- [Nis92] N. Nisan. $RL \subseteq SC$. In 24th Ann. ACM STOC, pages 619–623, 1992.
- [NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In Proc. of 34th Annual IEEE Symposium on Foundations of Computer Science, 1992.
- [NY] K. Nakano and H. Yasuura. Personal communication.
- [Rei84] J. H. Reif. Symmetric complementation. Journal of the Association for Computing Machinery, 31(2):401-421, April 1984.
- [Rot76] J. Rothstein. On the ultimate limitations of parallel processing. In Proceedings of the International Conference on Parallel Processing (best paper award), pages 206-212, 1976.
- [RP87] D. Reisis and V.K. Prasanna-Kumar. VLSI arrays with reconfigurable busses. In Proceedings of the 1st International Conference on SuperComputing, pages 732-742, 1987.
- [RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. J. Comp. System Sci., 28:216-230, 1984.
- [Sav70] W. Savitch. Relationships between nondeterministic and deterministic tape complexities.
 J. of Computer and System Sciences, 4:177-192, 1970.
- [Sch91] A. Schuster. Dynamic Reconfiguring Networks for Parallel Computers: Algorithms and Complexity Bounds. PhD thesis, Hebrew University, Jerusalem, ISRAEL, August 1991.
- [SV82] Y. Shiloach and U. Vishkin. An $O(\log N)$ parallel connectivity algorithm. Journal of Algorithms, 3:57-67, 1982.
- [Sze88] R. Szelepcsenyi. The method of forced enumeration for nondeterministic automata. Acta Informatica, 26(3):279–284, 1988.
- [TCS89] X. Thibault, D. Comte, and P. Siron. A reconfigurable optical interconnection network for highly parallel architecture. In Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation, 1989.
- [Val90] L.G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science. North Holland, Amsterdam, 1990.
- [Wan91] B.F. Wang. Configurational computation: a new algorithm design strategy on processor arrays with reconfigurable bus systems. PhD thesis, National Taiwan University, June 1991.
- [WC90a] B. Wang and G. Chen. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(4), 1990.
- [WC90b] B.F. Wang and G.H. Chen. Two dimensional processor array with a reconfigurable bus system is at least as powerful as CRCW model. *Information Processing Letters*, 36(1):31-36, 1990.

[WLH+87] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, and D.B. Shu. The image understanding architecture. Technical Report COINS TR 87-76, University of Massachusetts at Amherst, 1987.