The Graph Accessibility Problem and the Universality of the Collision CRCW Conflict Resolution Rule[*]

STEFAN D. BRUDA
Department of Computer Science
Bishop's University
Lennoxville, Quebec J1M 1Z7
CANADA
bruda@cs.ubishops.ca http://turing.ubishops.ca

Abstract: We complete the characterization of constant time computations on parallel models with reconfigurable buses: We have shown previously that directed reconfigurable multiple bus machines (DRMBM) with polynomially bounded resources and running in constant time solve exactly the same problems as nondeterministic logarithmic space bounded Turing machines, that write conflict resolution rules such as Priority or even Combining do not add computational power over the Collision rule, and that a bus of width 1 (a wire) suffices for any constant time computation on DRMBM. We now show that the same properties hold for constant time computations of directed reconfigurable networks (DRN). We then study the power of the Collision rule in the general case, linking strongly its universality with the capability of the model to compute the graph accessibility problem in constant time.

Key-words: parallel computation, real-time computation, reconfigurable multiple bus machine, reconfigurable network, CRCW conflict resolution rules, graph accessibility problem

1 Introduction

Previous work on parallel real-time computations [4] has produced a number of incidental results regarding models of parallel computations with reconfigurable buses. Specifically, a tight characterization of constant time computations on *reconfigurable multiple bus machines* (RMBMs) was offered: DRMBMs and *directed reconfigurable networks* (DRNs) with constant running time have been found to have the same computational power, which in turn is the same power as nondeterministic logarithmic space bounded Turing machines. In addition, it was shown that in the case of constant time RMBM computations there is no need for such powerful write conflict resolution rules as Priority or Combining as they do not add computational power over the easily implementable Collision rule, that a unitary bus width is enough (i.e., a simple wire as bus will do for all constant time RMBM computations), and that segmenting buses does not add computational power over fusing buses.

Motivated by such a result we complete in this paper the characterization of constant time computations on models with reconfigurable buses. We find that such properties (Collision being the most powerful resolution rule and unitary bus width being sufficient) hold for the other model with reconfigurable buses, namely the RN.

We then study whether the Conflict resolution rule is generally (i.e., not only for constant time computations) universal on DRMBMs and DRNs. We find that this is indeed the case, and we are able to do so in a more general setting. For indeed, the Collision rule is universal on any parallel model of computation that is capable of computing the graph accessibility problem (GAP) in constant time.

2 Preliminaries

Results proved elsewhere are introduced henceforth as Propositions, whereas results proved in this work are introduced as Theorems. Intermediate results are all Lemmata.

$GAP_{i,j}$ denotes be the following problem: Given a directed graph $G = (V, E)$, $V = \{1, 2, ..., n\}$ (expressed e.g., by the (boolean) incidence matrix I), determine whether vertex j is accessible from vertex i.

We denote by L [NL] the set of languages that are accepted by deterministic [nondeterministic] Turing machines that use at most $O(\log n)$ space (not counting the input tape) on any input of length n [6].

For some language $L \in \mathsf{NL}$ there exists a nondeterministic Turing machine $M = (K, \Sigma, \delta, s_0)$ that accepts L and uses $O(\log n)$ working space. K is the set of states, Σ is the tape alphabet (we consider without loss of generality that $\Sigma = \{0, 1\}$), δ is the transition

[*]This research was supported by the Natural Sciences and Engineering Research Council of Canada and by the Fond québécois de recherche sur la nature et les technologies.

relation, and s_0 is the initial state. M accepts an input string x iff M halts on x. A configuration of M working on input x is defined as a tuple (s, i, w, j), where s is the state, i and j are the positions of the heads on input and working tape, respectively, and w is the content of the working tape. There are $poly(n)$ possible configurations of M. For two configurations v_1 and v_2, we write $v_1 \vdash v_2$ iff v_2 can be obtained by applying δ exactly once on v_1 [6].

The set of possible configurations of M working on x forms a directed graph $G(M, x) = (V, E)$ as follows: V contains one vertex for each and every possible configuration of M working on x, and $(v_1, v_2) \in E$ iff $v_1 \vdash v_2$. It is clear that $x \in L$ iff some configuration (h, i_h, w_h, j_h) is accessible in $G(M, x)$ from the initial configuration (s_0, i_0, w_0, j_0). For any language $L \in \mathsf{NL}$ and for any x, determining whether $x \in L$ can be reduced to the problem of computing $GAP_{1,|V|}$ for $G(M, x) = (V, E)$, where M is some NL Turing machine deciding L.

The class of problems in NL with the addition of (any kind of) real-time constraints is denoted by NL/rt [4]. We denote by rt-$\mathrm{PROC}^M(f)$ the class of those problems solvable in real time by the parallel model of computation M that uses $f(n)$ processors (and also $f(n)$ buses if applicable) for any input of size n [3]. The following strongly supported conjecture is then established.

Claim 1 [4] rt-$\mathrm{PROC}^{\text{CRCW F-DRMBM}}(poly(n)) = \mathsf{NL}/rt$.

Two main models with reconfigurable buses have been developed in the literature: the *reconfigurable network* (or RN for short) [2] and the *reconfigurable multiple bus machine* (or RMBM) [7].

2.1 The RMBM

An RMBM [7] consists of a set of p *processors* and b *buses*. For each processor i and bus b there exists a *switch* controlled by processor i. Using these switches, a processor has access to the buses by being able to read or write from/to any bus. A processor may be able to *segment* a bus, obtaining thus two independent, shorter buses, and it is allowed to *fuse* any number of buses together by using a *fuse line* perpendicular to and intersecting all the buses. DRMBM, the *directed* variant of RMBM, is identical to the undirected model, except for the definition of fuse lines: Each processor features two fuse lines (*down* and *up*). Each of these fuse lines can be electrically connected to any bus. Assume that, at some given moment, buses i_1, i_2, ..., i_k are all connected to the down [up] fuse line of some processor. Then, a signal placed on bus i_j is transmitted in one time unit to all the buses i_l such that $i_l \geq i_j$ [$i_l \leq i_j$]. If some RMBM [DRMBM] is not allowed to segment buses, then this restricted variant is denoted by F-RMBM [F-DRMBM] (for "fusing" RMBM/DRMBM). The *bus width* of some RMBM (DRMBM, etc.) denotes the maximum size of a word that may be placed (and read) on (from) any bus in one computational step.

For CRCW (concurrent read, concurrent write; as opposed to CREW for concurrent read, exclusive write) RMBMs, the most realistic conflict resolution rule is Collision, where two values simultaneously written on a bus result in the placement of a special, "collision" value on that bus. We consider for completeness other conflict resolution rules such as Common, Arbitrary, Priority, and Combining. However, we find that all of these rules are in fact equivalent to the seemingly less powerful Collision rule (see Proposition 2.1(3)). We restrict only the Combining mode, requiring that the combining operation be associative and computable in nondeterministic linear space.

An RMBM (DRMBM, F-DRMBM, etc.) *family* $\mathcal{R} = (R_n)_{n \geq 1}$ is a set containing one RMBM (DRMBM, F-DRMBM, etc.) construction for each $n > 0$. A family $\mathcal{R}$ solves a problem P if, for any n, R_n solves all inputs for P of size n. We say that some RMBM family $\mathcal{R}$ is a *uniform RMBM family* if there exists a Turing machine M that, given n, produces the description of R_n using $O(\log(p(n)b(n)))$ cells on its working tape. We henceforth drop the "uniform" qualifier, with the understanding that any RMBM family described in this paper is uniform. Assume that some family $\mathcal{R} = (R_n)$ solves a problem P, and that each R_n, $n > 0$, uses $p(n)$ processors, $b(n)$ buses, and has a running time $t(n)$. We say then that $P \in \mathrm{RMBM}(p(n), b(n), t(n))$ (or $P \in$ F-$\mathrm{DRMBM}(p(n), b(n), t(n))$, etc.), and that $\mathcal{R}$ has *size complexity* $p(n)b(n)$ and *time complexity* $t(n)$.

It should be noted that a directed RMBM can simulate a nondirected RMBM by simply keeping all the up and down fuse lines synchronized with each other.

2.2 The Reconfigurable Network

An RN [2] is a network of processors that can be represented as a connected graph whose vertices are the processors and whose edges represent fixed connections between processors. Each edge incident to a processor corresponds to a (bidirectional) port of the processor. A processor can internally partition its ports such that all the ports in the same block of that partition are electrically connected (or fused) together. Two or more edges that are connected together by a processor that fuses some of its ports form a bus which connects ports of various processors together. CREW, Common CRCW, Collision CRCW, etc. are defined as for the the RMBM model. The *directed* RN (DRN for short) is similar to the general RN, except that the edges are directed. The concept of (uniform) RN family is identical to the concept of RMBM family. The class $\mathrm{RN}(p(n), t(n))$ [$\mathrm{DRN}(p(n), t(n))$] is the set of problems solvable by RN [DRN] uniform families with $p(n)$ processors ($p(n)$ is also called the *size complexity*) and $t(n)$ running time.

2.3 RMBM and Small Space Computations

The characterization of constant time RMBM computations described in the introduction of this paper can be formally summarized as follows:

Proposition 2.1 [4]

1. CRCW $\mathrm{DRMBM}(poly(n), poly(n), O(1))$ = NL = CRCW F-$\mathrm{DRMBM}(poly(n), poly(n), O(1))$ *with Collision resolution rule and bus width* 1.
2. $\mathrm{DRMBM}(poly(n), poly(n), O(1)) = \mathrm{DRN}(poly(n), O(1))$.
3. *For any problem P solvable in constant time by some (directed or nondirected) RMBM family using $poly(n)$ processors and $poly(n)$ buses, $P \in$ CRCW F-$\mathrm{DRMBM}(poly(n), poly(n), O(1))$ with Collision resolution rule and bus width* 1.

We shall determine a similar result for DRNs. We will adapt for this purpose two proofs used to derive Proposition 2.1. In order to make this paper self contained we include below these proofs.

Lemma 2.2 [4] CRCW $\mathrm{DRMBM}(poly(n), poly(n), O(1)) \subseteq$ NL, *for any write conflict resolution rule and any bus width.*

Proof. Consider some $R \in$ CRCW $\mathrm{DRMBM}(poly(n), poly(n), O(1))$ performing step d of its computation ($d \leq O(1)$). We need to find an NL Turing machine M_d that generates the description of R after step d using $O(\log n)$ space, and thus [5] an NL Turing machine M'_d that receives n' (the number of processors in R) and some i, $1 \leq i \leq n'$, and outputs the ($O(\log n)$ long) description for processor i instead of the whole description. We establish the existence of M_d (and thus M'_d) by induction over d, and thus we complete the proof.

M_0 exists by the definition of a uniform RMBM family. We assume the existence of M_{d-1}, M'_{d-1} and show how M_d is constructed. For each processor p_i and each bus k read by p_i during step d, M_d performs (sequentially) the following computation: M_d maintains two words b and ρ, initially empty. For every p_j, $1 \leq j \leq poly(n)$, M_d determines whether p_j writes on bus k. This implies the computation of $GAP_{j,i}$ (clearly computable in nondeterministic $O(\log n)$ space since it is NL-complete [6]). The local configurations of fused and segmented buses at each processor (i.e., the edges of the graph for $GAP_{j,i}$) are obtained by calls to M'_{d-1}. The computation of $GAP_{j,i}$ is necessary to ensure that we take p_j into account even when p_j does not write directly to bus k but instead to another bus that reaches bus k through fused buses.

If p_j writes on bus k, then M_d uses M'_{d-1} to determine the value v written by p_j, and updates b and ρ as follows: (a) If b is empty, then it is set to v (p_j is currently the only processor that writes to bus k), and ρ is set to j. Otherwise: (b) If R uses the Collision rule, the collision signal is placed in b. (c) If the conflict resolution rule is Priority, ρ and j are compared; if the latter denotes a processor with a larger priority, then b is set to v and ρ is set to j, otherwise, neither b nor ρ are modified; the Common and Arbitrary rules are handled similarly. (d) Finally, if R uses the Combining resolution rule with $\circ$ as combining operation, b is set to the result of $b \circ v$ (since the operation $\circ$ is associative, the final content of b is indeed the correct combination of all the values written on bus k).

Once the content of bus k has been determined, the configuration of p_i is updated accordingly, b and ρ are reset to the empty word, and the same computation is performed for the next bus read by p_i or for the next processor. The whole computation of M_d clearly takes $O(\log n)$ space. □

Lemma 2.3 [4] *Let $M = (K, \Sigma, \delta, s_0)$ be an* NL *Turing machine that accepts $L \in$* NL. *Then, given some word x, $|x| = n$, there exists a* CREW *or* CRCW F-DRMBM *algorithm that computes $G(M, x)$ (as an incidence matrix I) in $O(1)$ time, and using $poly(n)$ processors and $poly(n)$ buses of width* 1.

Proof. Put $n' = |V|$ ($n' = poly(n)$). The RMBM algorithm uses $n + (n'^2 - n')$ processors: The first n processors p_i, $1 \leq i \leq n$, contain x, i.e., each p_i contains x_i, the i-th symbol of x; p_i does nothing but writes x_i on bus i. We shall refer to the remaining $n'^2 - n'$ processors as p_{ij}, $1 \leq i, j \leq n'$. Each p_{ij} assembles first the configurations corresponding to vertices v_i and v_j of $G(M, x)$ and then considers the potential edge (v_i, v_j) corresponding to I_{ij}. If such edge exists, then p_{ij} writes $True$ to I_{ij}, and $False$ otherwise. There is no interprocessor communication between processors p_{ij}, thus any RMBM model is able to carry on this computation.

Clearly, given a configuration v_i, p_{ij} can compute in constant time any configuration v_l accessible in one step from v_i, as this implies the computation of at most a constant number ($O(2^k)$) of configurations. The whole algorithm runs thus in constant time. □

3 The Characterization of Constant Time RN Computations

The generality of the Collision resolution rule is not limited to RMBM computations. Indeed, the same property holds for constant time computations on RN as well. We also find that a DRN is able to carry out any constant time computation using only buses of width 1. The first main result of this paper is thus the DRN equivalent of Proposition 2.1, as follows:

Theorem 3.1 *For any problem π solvable in constant time on some variant of RN, it holds that $\pi \in$ $\mathrm{CRCW\,DRN}(poly(n), O(1))$ with Collision resolution rule and bus width* 1.

The proof of Theorem 3.1 is based on the following intermediate results.

Lemma 3.2 *For any $X \in \{\mathrm{CRCW}, \mathrm{CREW}\}$, $Y \in \{\mathrm{D}, \varepsilon\}$, and for any write conflict resolution rule, it holds that $X\,Y\mathrm{RN}(poly(n), O(1)) \subseteq$ $\mathrm{CRCW\,DRN}(poly(n), O(1))$ with the Collision resolution rule.*

Proof. First, note that CRCW DRN($poly(n)$, $O(1)$) $=$ NL for the Collision resolution rule [2]. Thus, we complete the proof by showing that, for any conflict resolution rule, $\mathrm{CRCW\,DRN}(poly(n), O(1)) \subseteq$ NL.

This result is however given by the proof of Lemma 2.2. Indeed, it is immediate that the Turing machines M_d and M'_d, $0 \leq d \leq c$ for some constant $c \geq 1$, provided in the mentioned proof work in the case of a RN R just as well as for the RMBM simulation. The only difference is that buses are not numbered in the RN case. So, we first assign arbitrary (but unambiguous) sequence numbers for the RN buses as follows: There exists an $O(\log n)$ space-bounded Turing machine that generates a description of R, since R belongs to a uniform RN family (in fact, such a Turing machine is M_0). Then, in order to find "bus k," M_d uses M_0 to generate the description of R until exactly k buses are generated. The description is discarded, except for the last generated bus, which is considered to be "bus k." Since M_0 is deterministic, it always generates the description in the same order. Thus, it is guaranteed that "bus k" is different from "bus j" if and only if $k \neq j$. The proof of Lemma 2.2 follows then unchanged.

The extra space used in the process of generating bus k consists in two counters over the set of buses (one to keep the value k and the other one to count how many buses have been already generated). The counters take $O(\log n)$ space each, since there are at most $poly(n)$ processors, and $(poly(n))^2 = poly(n)$. Thus, the overall space complexity remains $O(\log n)$, as desired. □

Lemma 3.3 *$GAP_{1,n} \in \mathrm{CRCW\,DRN}(n^2, O(1))$ with Collision resolution rule and bus width* 1.

Proof. Let R be the DRN solving $GAP_{1,n}$ instances of size n. Then, R uses n^2 processors (referred to as p_{ij}, $1 \leq i, j \leq n$), connected in a mesh. That is, there exists a (directional) bus from p_{ij} only to $p_{(i+1)j}$ if and only if $i+1 \leq n$, and to $p_{i(j+1)}$ if and only if $j+1 \leq n$, as shown in Figure 1. As shown in the figure, we also denote by E, S, N, and W the ports of p_{ij} to the buses going to $p_{i(j+1)}$, going to $p_{(i+1)j}$, coming from $p_{i(j-1)}$, and coming from $p_{(i-1)j}$, respectively.

We assume that the input graph $G = (V, E)$, $|V| = n$, is given by its incidence matrix I, and that each processor p_{ij} knows the value of I_{ij}.

The DRN R works as follows: Each processor p_{ij}, $i < j$ fuses its W and S ports if and only if $I_{ij} = True$. Analogously, each processor p_{ij}, $i > j$ fuses its N and E ports if and only if $I_{ij} = True$. Finally, each processor p_{ii} fuses all of its ports.

Then, a signal is placed by p_{11} on both its outgoing buses. If p_{nn} receives some signal (either the original one emitted by p_{11} or the signal corresponding to a collision) the input is accepted; otherwise, the input is rejected.

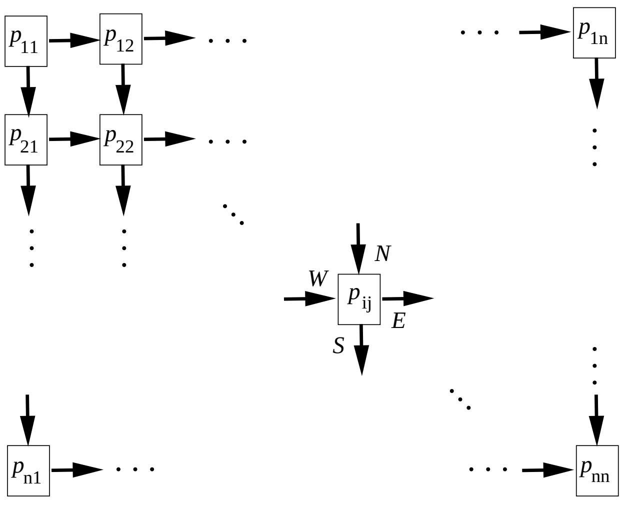

Figure 1: A mesh of $n \times n$ processors

It is immediate that R solves $GAP_{1,n}$, by an argument similar to the one for RMBM [4] (also note that a similar construction is presented and proved correct elsewhere [8]). In addition, the content of the signal received by p_{nn} is clearly immaterial, so a bus of width 1 suffices. $\Box$

Recall now that the graph $G(M, x)$ is the graph of configurations of the Turing machine M working on input x.

Lemma 3.4 *For any language $L \in \mathsf{NL}$ (with the associated NL Turing machine M accepting L), and given some word x, $|x| = n$, there exists a constant time CREW (and thus CRCW) DRN algorithm using $poly(n)$ processors and buses of width 1 that computes $G(M, x)$ (as an incidence matrix I).*

Proof. This fact is obtained by the same argument as the one presented in the proof of Lemma 2.3. Indeed, except for the distribution of input x to processors, there is no interprocessor communication; as such, any parallel machine will do.

Thus, the computation of $G(M, x) = (V, E)$ will be performed by the same mesh of processors R depicted in Figure 1, this time of size $n' \times n'$ (where $n' = |V|$). In addition, the desired input distribution will be accomplished by n additional meshes identical to R. We will denote these meshes by R_i, $1 \le i \le n$. For any $1 \le i, j \le n'$ and $1 \le k \le n$, the processor at row i, column j in mesh R_k [R], will be denoted by p_{ij}^k [p_{ij}^{n+1}]. Each processor p_{ij}^k has two new ports U and D. There exists a bus connecting port D of p_{ij}^k to port U of p_{ij}^{k+1} for any $1 \le k \le n$. The $n + 1$ meshes and their interconnection are shown in Figure 2.

At the beginning of the computation, x_k, the kth symbol of input x, is stored in a register of processor p_{11}^k, $1 \le k \le n$.

We note from the proof of Lemma 2.3 that each processor p_{ij}^{n+1} of R is responsible for checking the existence of a single edge (i, j) of $G(M, x)$. In order to accomplish this, it needs only *one* symbol $x_{h_{ij}}$ from x, namely the symbol scanned by the head of the input tape in configuration i. We assume that all the processors p_{ij}^k, $1 \le k \le n$, know the configuration i (and thus the value of h_{ij}).

It remains therefore to show now how $x_{h_{ij}}$ reaches processor p_{ij}^{n+1} in constant time, for indeed, after this distribution is achieved, R is able to compute the incidence matrix I exactly as shown in the proof of Lemma 2.3. The set of $n + 1$ meshes performs the following computation: For all $1 \le k \le n$ and $1 \le i, j \le n'$,

1. Each p_{11}^k broadcasts x_k to all the processors in R_k. To do this, all processors p_{ij}^k fuse together their N, S, E, and W ports, and then p_{11}^k places x_k on its outgoing buses.
2. Each p_{ij}^k compares k and h_{ij}, and writes $True$ in one of its registers d if they are equal and $False$ otherwise.
3. Each p_{ij}^k fuses its U and D ports, thus forming $i \times j$ "vertical" buses.
4. Each p_{ij}^k for which $d = True$ places x_k on its port D.
5. Finally, each p_{ij}^{n+1} stores the value it receives on its U port. This is the value of $x_{h_{ij}}$ it needs in order to compute the element I_{ij} of the incidence matrix.

It is immediate that the above processing takes constant time. In addition, it is also immediate that exactly one processor writes on each "vertical" bus, and thus no concurrent write takes place. Indeed, there exists exactly one processor p_{ij}^k, $1 \le k \le n$, such that $k = h_{ij}$. Therefore, we realized the input distribution.

I_{ij} is then computed by processor p_{ij}^{n+1} without further communication, as shown in the proof of Lemma 2.3. The construction of the DRN algorithm that computes I is therefore complete. Clearly, buses of width 1 are enough for the whole processing, since x is a word over an alphabet with 2 symbols. $\Box$

Given Lemmata 3.3, 3.4, and 3.2 we can now prove our first main result.

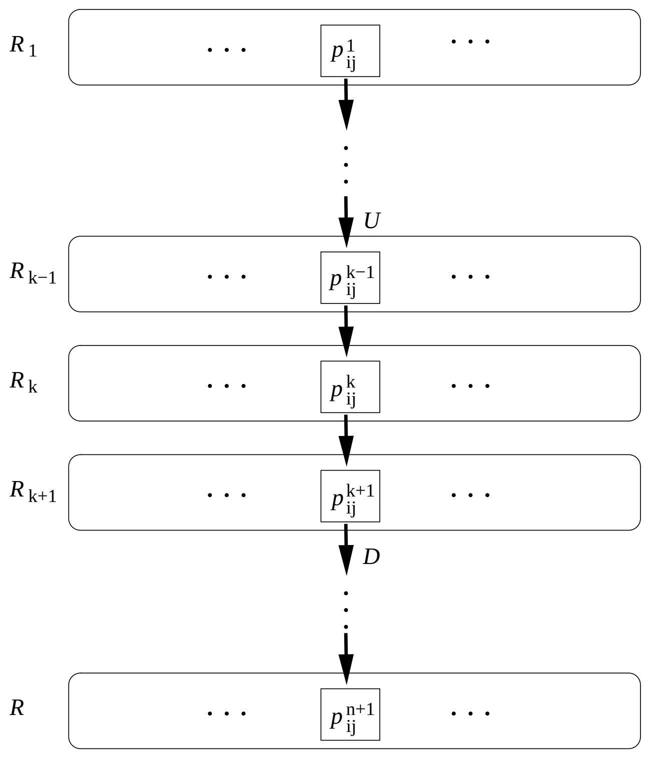

Figure 2: A collection of n meshes connected together

Proof of Theorem 3.1. That the Collision resolution rule is the most powerful follows from Lemma 3.2. It remains to be shown only that a bus width 1 suffices.

Given some language $L \in \mathsf{NL}$, let M be the (NL) Turing machine accepting L. For any input x, the DRN algorithm that accepts L works as follows: Using Lemma 3.4, it obtains the graph $G(M, x)$ of the configurations of M working on x. Then, it applies the algorithm from Lemma 3.3 in order to determine whether vertex n (halting/accepting state) is accessible from vertex 1 (initial state) in $G(M, x)$, and accepts or rejects x, accordingly. In addition, note that the values I_{ij} computed by (and stored at) p_{ij}^{n+1} in the algorithm from Lemma 3.4 are in the right place as input for p_{ij} in the algorithm from Lemma 3.3 (that uses only the mesh R). It is immediate given the aforementioned lemmata that the resulting algorithm accepts L and uses no more than $poly(n)$ processors, and unitary width for all the buses.

The proof is now complete, since all the problems solvable in constant time on RN are included in NL. □

4 GAP and the Universality of Collision

As far a constant time computation is concerned, we note an apparent contrast between the power of conflict resolution rules for models with reconfigurable buses (RMBM and RN) on one hand, and for shared memory models (PRAM) on the other hand. According to our results, Collision is the most powerful rule on RMBM and RN. By contrast, it is widely believed that the Combining CRCW PRAM is more powerful than the CRCW PRAM using the equivalent of a Collision resolution rule. To our knowledge however no proof on the matter exists to date. We believe that an investigation in this direction is an interesting pursuit. We also believe that the contrast between RN and PRAM is not only apparent (that is, we believe that the Combining CRCW PRAM is indeed more powerful than the Collision CRCW PRAM).

The reason for our belief is that the ability of some model to compute GAP in constant time is central to the constant time universality of the Collision rule, and also determines that exactly all DRMBM and DRN computations are in NL; we also note that GAP is NL-complete [6].

In light of this motivation, consider the classes $\mathbb{M}_{<GAP}$, $\mathbb{M}_{\equiv GAP}$, and $\mathbb{M}_{>GAP}$ of parallel models of computations using polynomially bounded resources (processors and, if applicable, buses), such that:

$\mathbb{M}_{<GAP}$ contains exactly all the models that cannot compute GAP in constant time, and cannot compute in constant time any problem not in NL. An example of such a model is the Common CRCW PRAM [9] (and thus the less powerful PRAM variants).

$\mathbb{M}_{\equiv GAP}$ contains exactly all the models that can compute GAP in constant time, but cannot compute in constant time any problem not in NL. This class includes the RMBM and RN.

$\mathbb{M}_{>GAP}$ contains exactly all the models that can compute GAP in constant time and can compute in constant time at least one problem not in NL. To our knowledge, no model has been proved to pertain to such a class, but a possible candidate is the *broadcast with selective reduction* model [1].

Our second main result is then formulated as follows:

Theorem 4.1 *The Collision resolution rule is universal on any model $M \in \mathbb{M}_{\equiv GAP} \cup \mathbb{M}_{>GAP}$, in the following sense:*

For any $R \in M$ with any write conflict resolution rule and $t(n)$ running time for input size n there exists an $R' \in M$ that performs the same computation as R in $O(t(n))$ time using the Collision resolution rule.

Proof. Suppose that some machine $R' \in M$ computes $GAP_{i,j}$ for a graph G with n vertices in constant time and using $poly(n)$ processors and $poly(n)$ buses (if applicable); R' exists by definition. Given the $\log n$ restriction to the size of the registers of the participating processors, a processor cannot hold but a constant number of edge descriptions. We then assume without loss of generality that m processors hold information about the m edges in G. Since the whole computation performed by R' completes in constant time, it follows that the m processors holding the edge information are made to communicate with each other in constant time by using extra $poly(m)$ resources.

This apparently irrelevant property can be put in a more interesting way: given some $R \in M$ with m processors, there exists an $R' \in M$ that includes R such that (a) all the original processors from R communicate with each other in constant time in R', and (b) R' uses at most $poly(m)$ more resources than R. Call this property the *constant-GAP-constant-communication* property.

Let now $R \in M$ be some CRCW machine with polynomially bounded resources that uses the Combining resolution rule to perform its computation. We then replace R with a variant that uses the Collision resolution rule and then we split each step i of the computation into the following constant number of steps:

1. Each processor p of R reads the content of external resources (buses, memory locations, etc.) as required and then performs the prescribed computation for step i, except that whenever p wants to write to external resource k it also writes the same value into a dedicated resource k_p (there is one such a resource for each processor). Note that the Collision value might be placed in k.
2. A machine R_g then finds which of the original resources hold a Collision value, and for each such a resource: (a) determines based on the configurations of the original processors which of these processors write into the respective resource, (b) computes the resulting value to be placed into the respective resource (instead of the Collision signal), and (c) writes the computed value into the resource.

Clearly the running time of the combination R, R_g, and the new resources k_p put together is of the same order as the running time of the original R. Indeed, Step $(2.a)$ might imply repeated computations of some $GAP_{i,j}$, whilst step $(2.b)$ is computable in logarithmic space given that the Combining operation is associative and computable in linear space. The whole process performed by R_g is thus achievable in nondeterministic logarithmic space. On the other hand $GAP_{i,j}$ is NL-complete, so R_g can be a implemented as a machine in M running in constant time.

It is also immediate that, if R uses $poly(n)$ processors and resources, then the combination also uses $poly(n)$ resources. In addition to all of these, the processors from R_g should be able to communicate in constant time with all the processors in R (to inspect their configurations); this can be accomplished however with a $poly(n)$ increase in resources according to the constant-GAP-constant-communication property stated at the beginning of the proof. The result is then established for the Combining rule. The other conflict resolution rules are considered similarly in an immediate manner, and the theorem obviously holds for any machine that does not use a conflict resolution rule (i.e., a CREW machine). $\square$

Given that both DRMBMs and DRNs are in $\mathbb{M}_{\equiv GAP}$, the following is an immediate consequence of Theorem 4.1 and completes the characterization of models with reconfigurable buses:

Corollary 4.2 *The Collision resolution rule is universal on models with reconfigurable buses. That is:*

For any $X \in \{\text{CRCW}, \text{CREW}\}$, $Y \in \{\text{D}, \varepsilon\}$, $Z \in \{\text{RN}(poly(n), \cdot), \text{RMBM}(poly(n), poly(n), \cdot)\}$, $t : \mathbb{N} \to \mathbb{N}$, and for any write conflict resolution rule, it holds that $X\,YZ(t(n)) \subseteq X\,\text{D}Z(O(t(n)))$ with the Collision resolution rule.

4.1 GAP and Real-Time Computations

Compare the previous discussion on GAP with the following immediate generalization of Claim 1:

Theorem 4.3 *For any models of computation M_1, M_2, and M_3 such that $M_1 \in \mathbb{M}_{<GAP}$, $M_2 \in \mathbb{M}_{\equiv GAP}$, and $M_3 \in \mathbb{M}_{<GAP}$, it holds that*

$$\text{rt-PROC}^{M_1}(poly(n)) \subseteq \mathsf{NL}/rt \tag{1}$$
$$\text{rt-PROC}^{M_2}(poly(n)) = \mathsf{NL}/rt \tag{2}$$
$$\text{rt-PROC}^{M_3}(poly(n)) \supset \mathsf{NL}/rt \tag{3}$$

Proof. Minor variations of the arguments used in [4] show that those computations which can be performed in constant time on M_i, $1 \leq i \leq 3$, can be performed in the presence of however tight time constraints (and thus in real time in general). Then, Relations (1) and (3) follow immediately from Claim 1.

By the same argument, rt-PROC$^{M_2}(poly(n)) \supseteq$ rt-PROC$^{\text{CRCW F-DRMBM}}(poly(n))$ holds as well. The equality (and thus Relation (2)) is given by the arguments that support Claim 1 [4]. $\square$

Thus, the characterization of real-time computations established by Claim 1 does hold in fact for any machines that are able to compute GAP in constant time. The characterization presented in Theorem 4.3 emphasizes in fact the strength of Claim 1. Indeed, as noted above, no model more powerful than the RMBM is known to exist. That is, according to the current body of knowledge, $\mathbb{M}_{>GAP} = \emptyset$. Unless this relation is found to be false, it follows from Claim 1 that no problem outside NL can be solved in real time in general, not only as far as RMBM computations are concerned.

5 Conclusions

We found that there exists a very strong similarity between the two models with reconfigurable buses, the RN and the RMBM: Not only they solve the same problems (namely, exactly all the problems in NL), but in both cases (a) the smallest possible bus width is enough for all problems, and (b) the Collision resolution rule is the most powerful (even as powerful as the Combining rule).

Furthermore, we showed that Collision is the most powerful on DRNs and DRMBMs for any running time. According to our results, the discussion regarding the practical feasibility of rules like Priority or Combining on spatially distributed resources such as buses is no longer of interest. Indeed, such rules are not only of questionable feasibility, but also not necessary!

We also noted the central role of the graph accessibility problem (GAP) for the DRN and DRMBM results obtained here and also previously [4]. In fact, the universality of the Collision rule on models with reconfigurable buses was established as a consequence of our more general result that the Collision rule is universal on any model of parallel computation that is able to compute GAP in constant time.

Having found that Collision is universal on all the models from $\mathbb{M}_{\equiv GAP}$ and given that GAP is not computable in constant time on models from $\mathbb{M}_{<GAP}$, we believe that the Collision resolution rule is *not* universal on any model from $\mathbb{M}_{<GAP}$. We also expect that a lesser conflict resolution rule is universal on models in $\mathbb{M}_{>GAP}$ (if any). Showing (or disproving) this is an intriguing open problem.

References:

[1] S. G. AKL AND G. R. GUENTHER, *Broadcasting with selective reduction*, in Proceedings of the IFIP Congress, 1989, pp. 515–520.

[2] Y. BEN-ASHER, K.-J. LANGE, D. PELEG, AND A. SCHUSTER, *The complexity of reconfiguring network models*, Information and Computation, 121 (1995), pp. 41–58.

[3] S. D. BRUDA AND S. G. AKL, *Pursuit and evasion on a ring: An infinite hierarchy for parallel real-time systems*, Theory of Computing Systems, 34 (2001), pp. 565–576. For an extended version see http://turing.ubishops.ca/home/bruda/-papers/pursuit.

[4] ——, *On the relation between parallel real-time computations and logarithmic space*, in Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, Nov. 2002, pp. 102–107. For an extended version see http://turing.ubishops.ca/-home/bruda/papers/nlogspace.

[5] R. GREENLAW, H. J. HOOVER, AND W. L. RUZO, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press, New York, NY, 1995.

[6] A. SZEPIETOWSKI, *Turing Machines with Sublogarithmic Space*, Springer Lecture Notes in Computer Science 843, 1994.

[7] J. L. TRAHAN, R. VAIDYANATHAN, AND R. K. THIRUCHELVAN, *On the power of segmenting and fusing buses*, Journal of Parallel and Distributed Computing, 34 (1996), pp. 82–94.

[8] B.-F. WANG AND G.-H. 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 (1990), pp. 500–507.

[9] ——, *Two-dimensional processor array with a reconfigurable bus system is at least as powerful as CRCW model*, Information Processing Letters, 36 (1990), pp. 31–36.