Scalable Computing: Practice and Experience
Volume 10, Number 2, pp. 163–172. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

RELATIONS BETWEEN SEVERAL PARALLEL COMPUTATIONAL MODELS[*]

STEFAN D. BRUDA[†] AND YUANQIAO ZHANG[†]

Abstract. We investigate the relative computational power of parallel models with shared memory. Based on feasibility considerations present in the literature, we split these models into "lightweight" and "heavyweight," and then find that the heavyweight class is strictly more powerful than the lightweight class, as expected. On the other hand, we contradict the long held belief that the heavyweight models (namely, the Combining CRCW PRAM and the BSR) form a hierarchy, showing that they are identical in computational power with each other. We thus introduce the BSR into the family of practically meaningful massively parallel models. We also investigate the power of concurrent-write on models with reconfigurable buses, finding that it does not add computational power over exclusive-write under certain reasonable assumptions. Overall, the Combining CRCW PRAM and the CREW models with directed reconfigurable buses are found to be the simplest of the heavyweight models, which now also include the BSR and all the models with directed reconfigurable buses. These results also have significant implications in the area of real-time computations.

Key words: parallel computation, shared memory parallel models, reconfigurable buses, parallel random access machine, broadcast with selective reduction, reconfigurable multiple bus machine, reconfigurable network, concurrent-read concurrent-write conflict resolution rules, real-time

1. Introduction.

The concurrent-read concurrent-write parallel random access machine (CRCW PRAM) is the most convenient model of parallel computation and so it is used extensively in analyzing parallel solutions to various problems. Lower bounds on the PRAM are in particular very strong. The Priority CRCW PRAM is sometimes considered [19] to be at the upper level of feasible parallel models. The broadcast with selective reduction (BSR) on the other hand is at present the most powerful model of parallel computation, with the Combining CRCW PRAM falling somewhere in between. By logical extension of the Priority CRCW PRAM being at the upper end of the feasibility chain [19], the Combining CRCW PRAM and the BSR should not be considered feasible; however, efficient implementations have been proposed for both [3].

It is widely believed (though to our knowledge not proven) that Priority CRCW PRAM is strictly less powerful that the Combining CRCW PRAM, which is in turn strictly less powerful than the BSR. In all, one can identify two "categories" of models of parallel computation: we thus call the Priority CRCW PRAM and the models below it in terms of computational power *lightweight* models, with the *heavyweight* models represented by the Combining CRCW PRAM and the BSR.

The first purpose of this paper is then to analyze the relation between these two categories in terms of computational power, as well as the relations between the models within the heavyweight class. Although not necessarily explicit, we always have in mind real-time computations, so we are using a strong notion of relationship: we say that model A is (not necessarily strictly) more powerful than model B only if $t(n)$ computational steps of model B using polynomial resources can be simulated in $O(t(n))$ steps of model A using polynomial resources. We often accomplish this by showing how one model is capable of simulating one computational step of the second model in constant time (and the other way around whenever we want to prove an equality).

As expected, we find that the class of heavyweight models is strictly more powerful than the class of lightweight models. Surprisingly, we also find that all the heavyweight models are however equivalent with each other, despite the perceived high computational power (and low feasibility) of the BSR.

We also enhance the usability of models with reconfigurable buses. Such models (namely the reconfigurable multiple bus machine or RMBM for short, and the reconfigurable network or RN) have been studied in the past [5, 21], being recognized as realistic models for VLSI design and other parallel machines [6, 15]. While the more complex variants of the PRAM embrace the combining of values written into memory, such a combination is justly disregarded (in as much as little to no work exists on the matter) on such distributed resources as buses. That such complex combinations are not necessary (and that Collision is sufficient for any computation) is known [7]. We further find that under a reasonable assumption not even Collision is necessary, that concurrent-write does not add anything over exclusive-write.

[*]This research was supported by the Natural Sciences and Engineering Research Council of Canada. Part of this research was also supported by Bishop's University.

[†]Department of Computer Science, Bishop's University, 2600 College St, Sherbrooke, Quebec J1M 1Z7, Canada, (stefan@bruda.ca, yqzhang@cs.ubishops.ca)

Our results are rather significant, as we essentially free the analysis of parallel algorithms and problems from a number of restrictions. On shared memory models, whether using the powerful Broadcast instruction of the BSR diminishes the practicality of the analysis (specifically, we are not aware of any real shared memory machine that implements Broadcast, yet as a consequence of our result such an instruction can be used with no consequences in the design of algorithms for such machines) becomes immaterial. On reconfigurable buses, we allow the use of any form of concurrent-write without penalty, given that any concurrent-write can be simulated in constant time by an exclusive-write machine. At the same time, we also offer a strict delimitation between the two classes of heavyweight and lightweight models of parallel computation.

Furthermore, these results become particularly significant in conjunction with previous work on real-time parallel computations [8] and the characterization of models with reconfigurable buses [9]. We show in effect that the Combining CRCW PRAM is as powerful as reconfigurable buses (thus being useful for the design and analysis of VLSI circuits). We also show that the Combining CRCW PRAM and the exclusive-write on directed reconfigurable buses are the simplest models that can solve exactly all the problems solvable under no matter how tight real-time constraints. In effect, we "simplify" the domain of real time, thus strengthening previous results on real-time computations.

We proceed as follows: The next section presents the necessary preliminaries. The shared memory hierarchy is the subject of Section 3, followed by the discussion on exclusive-write being universal on reconfigurable buses, that can be found in Section 4. The global hierarchy of lightweight and heavyweight models (with shared memory or reconfigurable buses) is summarized in Section 5. Real-time considerations are addressed in Section 6. We conclude in Section 7. Results proved elsewhere are henceforth introduced as Propositions, whereas results proved in this work are introduced as Theorems. Intermediate results are all Lemmata.

2. Preliminaries.

PARITY$_n$ denote the following problem: given n integer data x_1, x_2, …, x_n, compute the function $\text{PARITY}_n(x_1, x_2, \ldots, x_n) = (\sum_{i=1}^{n} x_i) \mod 2$.

2.1. The PRAM and the BSR.

Shared memory models of parallel computation are represented by the *parallel random access machine* (or PRAM for short) and by its extension, the *broadcast with selective reduction* (or BSR).

A PRAM [2, 19] consists in a number of processors that share a common random-access memory. The processors execute the instructions of a parallel algorithm synchronously. The shared memory stores intermediate data and results, and also serves as communication medium for the processors. The model is further specified by defining the memory access mode; we thus obtain exclusive-read exclusive-write (EREW), concurrent-read exclusive-write (CREW) and concurrent-read concurrent-write (CRCW) PRAM. While reading concurrently from the shared memory is defined straightforwardly, writing concurrently into the shared memory requires the introduction of a conflict resolution rule (for the case in which two or more processors write into the same memory location). Four such conflict resolution rules are in use: Common (the processors writing simultaneously in the same memory location must write the same value or else the algorithm fails), Collision (multiple processors writing into the same memory location garble the result so that a special "collision" marker ends up written at that location instead of any processor-provided data), Priority (processors are statically numbered and the memory location receives the value written by the lowest numbered processor), and Combining (where a binary, associative reduction operation is performed on all the values sent by all the processors to the same memory location and the result is stored in that memory location).

Given the obviously increased computational power as well as the straightforward implementation of concurrent-read machines, we will not consider exclusive-read variants. For similar reasons, exclusive-write machines will receive a spotty consideration if any.

The BSR model [2, 4] is an extension of the Combining CRCW PRAM. All the read and write operations of the Combining CRCW PRAM can also be performed by the BSR. In addition, all the BSR processor can write simultaneously into all the memory locations (the Broadcast instruction). Every Broadcast instruction consists in three steps: In the *broadcasting step*, all the n participating processors produce a datum d_i and a tag g_i, $1 \leq i \leq n$, destined to all the m memory locations. In the *selection step* each of the m memory locations uses a limit l_j, $1 \leq j \leq m$ and a selection rule $\sim \in \{<, \leq, =, \geq, >, \neq\}$ to test the received data; the datum d_i is selected for the next step if and only if $g_i \sim l_j$. Finally, the *reduction step* combines all the data d_i destined for memory location j, $1 \leq j \leq m$ and selected in the previous step using a binary, associative operator $\mathcal{R}$, and then writes the result into memory location j. The Broadcast instruction is performed simultaneously for all the processors and all the memory locations.

Typically, the reduction operator $\mathcal{R}$ of the BSR as well as the Combining operator of the Combining CRCW PRAM can be any of the following operations: Σ (sum), Π (product), $\bigwedge$ (logical conjunction) $\bigvee$ (logical disjunction), $\bigoplus$ (logical exclusive disjunction), max (maximum), and min (minimum).

We denote by X CRCW PRAM$(p(n), t(n))$ the class of problems of size n that are solvable in time $t(n)$ on a CRCW PRAM that uses $p(n)$ processors and X as collision resolution rule, $X \in \{\text{Common}, \text{Collision}, \text{Priority}, \text{Combining}\}$. Similarly, we denote by BSR$(p(n), t(n))$ the class of problems of size n that are solvable in time $t(n)$ on a BSR that uses $p(n)$ processors.

The notion of uniform families of PRAM or BSR machines exists [18]. However, we do not need such a notion in this paper. Still, we assume as customary that one location in the shared memory has $O(\log n)$ size for an input of size n, and that the number of memory locations are upper bounded by a polynomial in the number of processors used. Again as usual, we assume that every BSR or PRAM processor has a constant number of internal registers, each of size $O(\log n)$. Finally, we assume that every PRAM processor knows its number as well as the total number of processors in the system. The following result regarding the PRAM will be used later:

Proposition 2.1. [12] PARITY$_n \notin$ Priority CRCW PRAM$(\text{poly}(n), O(1))$.

2.2. The RMBM and the RN. An RMBM [21, 22] consists of a set of p *processors* and b *buses*. For each processor i and bus j 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 buses i_1, i_2, $\ldots$, 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).

For CRCW RMBM, 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 do not need to consider other conflict resolution rules such as Common, Priority, and even Combining, since all of these rules are in fact equivalent to the seemingly less powerful Collision rule (or even exclusive-write, see Theorem 4.2).

An RMBM (DRMBM, F-DRMBM, etc.) *family* $R = (R_n)_{n \geq 1}$ is a set containing one RMBM (DRMBM, etc.) construction with $p(n)$ processors and $b(n)$ buses for each $n > 0$. A family R solves a problem π if R_n solves all inputs of π of size n. We say that some RMBM family R is *uniform* if there exists an NL Turing machine M that, given n, produces the description of R_n using $O(\log(p(n)b(n)))$ work tape cells. We henceforth drop the "uniform" qualifier, with the understanding that any RMBM family described here is uniform. If some family $R = (R_n)$ solves a problem π, and each R_n, $n > 0$, uses $p(n)$ processors, $b(n)$ buses, X as write conflict resolution rule ($X \in \{\text{CREW}, \text{Common CRCW}, \text{Collision CRCW}, \text{Priority CRCW}, \text{Combining CRCW}\}$), and runs in $t(n)$ time, then we say that $\pi \in X$ RMBM$(p(n), b(n), t(n))$ (or $\pi \in X$ F-DRMBM$(p(n), b(n), t(n))$, etc.), and that R has *size complexity* $p(n)b(n)$ and *time complexity* $t(n)$. Whenever we state a property that holds for any conflict resolution rule we drop X (thus writing $\pi \in$ RMBM$(p(n), b(n), t(n))$, etc.).

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

An RN [5, 22] is a network of processors forming a connected graph whose vertices are the processors and whose edges represent fixed connections. Each edge incident to a processor corresponds to a (bidirectional) port of the processor. A processor can partition its ports such that all the ports in the same block of the partition are electrically connected (fused) together. The edges that are thus connected together form a bus. CREW, Common CRCW, Collision CRCW, etc. are defined as for the the RMBM model. The edges are directed in a *directed* RN (DRN). The concept of (uniform) RN family is identical to the RMBM concept. For some write conflict resolution rule X ($X \in \{\text{CREW}, \text{Common CRCW}, \text{Collision CRCW}, \text{Priority CRCW}, \text{Combining CRCW}\}$), X RN$(p(n), t(n))$ [X 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 using the conflict resolution rule X. We drop X when the property refers to any conflict resolution rule.

As in the RMBM case, we note that for any undirected RN there exists a directed RN with the same size complexity and the same running time that simulates it (the directed RN provides a couple of edges of opposite directionality for every edge in the undirected RN).

3. The Shared Memory Hierarchy. Recall that we called Priority CRCW PRAM and all the models of less computational power lightweight, while the Combining CRCW PRAM and the BSR were called heavyweight. We show now that this terminology is introduced for good reason. Specifically, we show that all the heavyweight models have the same computational power, and that they are strictly more powerful than the lightweight models. The first result of this paper is thus stated as follows:

THEOREM 3.1. *For $X \in \{\text{Collision}, \text{Common}\}$ and for any $n \in \mathbb{N}$,*

$$\begin{aligned} X \text{ CRCW PRAM}(\text{poly}(n), O(t(n))) &\subseteq \\ \text{Priority CRCW PRAM}(\text{poly}(n), O(t(n))) &\subsetneq \\ \text{Combining CRCW PRAM}(\text{poly}(n), O(t(n))) &= \\ \text{BSR}(\text{poly}(n), O(t(n))). \end{aligned}$$

Proof. Theorem 3.1 is a direct consequence of Lemmata 3.2 to 3.5 below. Specifically, the inclusions shown in the theorem are proven in the mentioned lemmata one by one. □

We complete all the proofs below by showing how the model on the right hand side of the inclusion simulates in constant time one computational step of the model on the left hand side. Once this is shown, the inclusion that needs to be proved becomes immediate.

We note that some of the hierarchy analyzed here has also been investigated earlier [14]. In particular, Lemma 3.2 has been established in a stronger version [14] (namely, that the two models are equivalent). For completeness however we include here our proof of this result.

LEMMA 3.2.

$$\forall n \in \mathbb{N} : \text{Collision CRCW PRAM}(\text{poly}(n), O(t(n))) \subseteq \text{Priority CRCWPRAM}(\text{poly}(n), O(t(n)))$$

Proof. A Collision CRCW PRAM with k processors p_i, $1 \le i \le k$ and m memory locations u_j, $1 \le j \le m$ is readily simulated by a Priority CRCW PRAM with $2k+m$ processors denoted by $p_i^{\downarrow}$ $(1 \le i \le k)$, $p_i^{\uparrow}$ $(1 \le i \le k)$, and p_j^m $(1 \le j \le m)$. (Note however that the processor group p_j^m and the processor group $p_i^{\downarrow}$ plus $p_i^{\uparrow}$ take turns in the simulation, so the actual number of processors required is $\max(2k, m)$; however, the explicit differentiation eases the presentation.)

In addition to the original memory locations u_j, we use two more "banks" of the same size $u_j^{\downarrow}$ and $u_j^{\uparrow}$, $1 \le j \le m$. A Collision CRCW PRAM step (read, compute, write) is then simulated as follows:

1. For every $1 \le i \le k$, both the processors $p_i^{\downarrow}$ and $p_{k+1-i}^{\uparrow}$ perform the same read, compute, and write cycle as the original p_i, with the following addition: Whenever processor $p_i^{\downarrow}$ writes into memory location u_j, it also writes its number j into memory location $u_j^{\downarrow}$; similarly, whenever processor $p_{k+1-i}^{\uparrow}$ writes into memory location u_j, it also writes its number $k+1-i$ into memory location $u_j^{\uparrow}$.
2. Every processor p_i^m writes the collision value into memory location u_j if and only if $u_j^{\downarrow} \ne u_j^{\uparrow}$.

That the above simulation takes constant time is immediate. Note further that after Step 1 of the simulation the location $u_j^{\downarrow}$ $[u_j^{\uparrow}]$ contains the index of the lowest [highest] ranked processor that modified the memory location u_j (indeed, we operate on a Priority CRCW model, so only the lowest numbered processor succeeds in writing into a given memory location; we then chose the processors numbers in appropriate manner for the desired property to happen). Then, whenever $u_j^{\downarrow} \ne u_j^{\uparrow}$, more than one processor wrote into the given memory location. A collision occurred, so Step 2 places a collision marker accordingly. The simulation is complete. □

LEMMA 3.3.

$$\forall n \in \mathbb{N} : \text{Common CRCW PRAM}(\text{poly}(n), O(t(n))) \subseteq \text{Priority CRCWPRAM}(\text{poly}(n), O(t(n)))$$

Proof. We simulate now a computational step of a Common CRCW PRAM with k processors and m memory locations in constant time using a Priority CRCW PRAM. The simulation will use the same number k of processors (we denote them by p_i, $1 \le i \le k$) and the same memory space (denoted by u_j, $\le j \le m$). The simulation proceeds as follows:

1. All the processors p_i carry on the computational step prescribed by the Common CRCW PRAM algorithm, including the operation of writing into the shared memory (recall however that we are now using the Priority conflict resolution rule).
2. Each processor p_i that wrote a value v_i into memory location u_j in Step 1 also remembers v_i by storing it into an otherwise unused internal register ρ.
3. Every p_i that wrote a value into location u_j in Step 1 read the content of u_j and compares it with the content of its register ρ.
   (a) If the contents of u_j and ρ are the same then either (a) p_i is the sole processor which wrote into u_j, (b) p_i is the highest priority processor which wrote into u_j, or (c) p_i and the highest priority processor agree on the value written into u_j. None of these cases violate the Common resolution rule, so p_i does not do anything.
   (b) If on the other hand u_j and ρ contain different values, then the value written into u_j by p_i disagrees with the value written in the same location by some other processor, which in turn violates the Common resolution rule. So p_i aborts the algorithm and reports the failure.

Note that in effect we chose *one* of the processors writing concurrently into a memory location as representative for all the others (given that we have a Priority machine at our disposal, that representative turned out to be the processor with the highest priority; however the way we chose a representative is immaterial). Every processor which wants to write a value in some memory location compares now its value with the value already written by its representative; if the value is different, then the Common conflict resolution rule is violated; otherwise all is good and the overall algorithm continues with the next step.

The above proof uses the usual definition of Common, as presented in Section 2. Still, we note that sometimes this definition is termed "Fail-safe Common," case in which the "Fail Common" or "Tolerant" variant is also defined [2, 13]. In such a variant, any computational step that violates the Common resolution rule is discarded completely (that is, for all the processors in the system) and the algorithm continues with the next step (instead of aborting the computation). A proof for this modified variant of Common is readily possible. Indeed, all we need is "backup" copies of the memory locations used by the algorithm, plus one memory location used to signal any violation to everybody. The processors now use the backup memory to perform all the simulation described above, except that they set the violation flag instead of aborting the algorithm whenever a violation of the Common resolution rule occurs (since is is just a flag, using Priority as conflict resolution rule will do just as well as almost any other rule). At the end of the computational step, all the processors inspect the flag and write again the values they wanted to write in the first place (this time in the main memory, not the backup) only if the flag is not set; otherwise they all proceed to the next step immediately, thus leaving the main working memory unchanged. □

Lemma 3.4.

$$\forall n \in \mathbb{N} : \text{Priority CRCW PRAM}(\text{poly}(n), O(t(n))) \subsetneq \text{CombiningCRCW PRAM}(\text{poly}(n), O(t(n)))$$

Proof. A Priority CRCW PRAM with k processors p_i, $1 \leq i \leq k$ and m memory locations u_j, $1 \leq j \leq m$ is readily simulated by a Combining CRCW PRAM with the same number of processors (denoted by p'_i) and $2m$ memory locations (denoted by u_i and u'_i):

1. Each processor p'_i performs the same read and compute operations as p_i. Instead of writing (to memory location u_j), p'_i however performs a "dry run" by writing its number into memory location u'_j using a Combining CRCW write with min as combining operation.
2. Each processor p'_i performs now the real write operation: It writes into the memory location u_j in which it wanted to write to begin with, but does so only if its number matches the value stored in u'_j. The min as combining operation performed over the locations u'_j in the previous step ensures that a matching occurs only for the lowest numbered processor, as desired.

That Combining CRCW PRAM$(\text{poly}(n), O(t(n))) \neq$ Priority CRCW PRAM$(\text{poly}(n), O(t(n)))$ is an immediate extension of Proposition 2.1. Indeed, by Proposition 2.1 PARITY$_n$ is not solvable in constant time on a Priority CRCW PRAM with polynomial number of processors. On the other hand, PARITY$_n$ is trivially solvable in constant time using a Combining CRCW PRAM with n processors: Every processor p_i of such a machine, $1 \leq i \leq n$ holds one input datum x_i and writes it into some designated memory location μ by performing a Combining CRCW using Σ as combining operation; then p_1 performs a modulo operation on μ and thus μ contains the output of PARITY$_n$ for the given instance, as desired. □

Lemma 3.5. $\forall n \in \mathbb{N}$: Combining CRCW PRAM$(\text{poly}(n), t(n)) = \text{BSR}(\text{poly}(n), O(t(n)))$.

Proof. That Combining CRCW PRAM$(\text{poly}(n), t(n)) \subseteq \text{BSR}(\text{poly}(n), O(t(n)))$ is immediate from the definition of the BSR. Surprisingly enough, the reverse inclusion is also true. We show now this reverse inclusion. Specifically, we show how one BSR computational step is simulated by a Combining CRCW PRAM in constant time.

Consider a BSR with k processors and m memory locations. Every BSR processor p_i is simulated by a set of m PRAM processors r_{ij}, $1 \leq j \leq m$. The PRAM memory is doubled, every BSR memory location u_j will be simulated by two PRAM memory locations u_j^d and u_j^l, $1 \leq j \leq m$. Finally, the PRAM uses extra processors p_j^u, $1 \leq j \leq m$ (once more processors p_j^u and r_{ij} actually take turns in the simulation so we differentiate them for convenience only; the actual number of processors used is in fact smaller as these two groups can overlap with each other). The PRAM simulation of a BSR Broadcast step (read, compute, Broadcast) then proceeds as follows:

- *Read and Compute:* For $1 \leq i \leq n$, all the processors r_{ij}, $1 \leq j \leq m$ perform the reading and the computation prescribed for p_i. Every time some processor wants to read the value of u_j it will read the value of u_j^d instead. Processors r_{ij} will then *all* hold the values of the datum d_i and the tag g_i originally computed by p_i.
  Note that there are n groups of processors; all the m processors in the same group perform the same computation. This may appear wasteful but is intentional and will be used to simulate the Broadcast instruction.
- *Selection limits:* Every processor p_j^u computes the limit l_j associated with u_j in the selection phase of the BSR step, and stores it in the memory location u_j^l.
- *Broadcast instruction:* r_{ij} will be responsible for the data written by the BSR processor p_i into memory location r_j:
  1. r_{ij} reads l_j from memory location u_j^l so that it holds d_i, g_i, and l_j;
  2. r_{ij} then computes the selection criterion $g_i \sim l_j$ as prescribed by the BSR algorithm;
  3. r_{ij} writes d_j into memory location u_j^d if and only if $g_i \sim l_j =$ True, using a Combining CRCW write with the combining operator prescribed by the BSR algorithm.

  Note that for some fixed i all the processors r_{ij}, $1 \leq j \leq m$ contain identical data, so p_i's replacement covers all the memory locations, thus realizing the desired broadcast.

In effect, we use one PRAM processor for every pair processor–memory location in the BSR algorithm. This allows for an immediate simulation of the broadcast phase of a Broadcast instruction: Instead of broadcasting, every PRAM processor is now responsible for writing to one memory location only; but then since we have as many processors as memory locations, we nonetheless write to all the memory locations at once, as desired. The correctness of the rest of the simulation is immediate, and so is the overall constant running time. ☐

4. On the Power of Concurrent-Write on Reconfigurable Buses.

It has been shown [7] that Collision is universal on reconfigurable buses, meaning that all the other conflict resolution rules can be simulated by Collision with constant overhead. Under certain assumptions, this result can be strengthened. Indeed, one can conceivably identify two variants of concurrent-write on reconfigurable buses:

Definition 4.1. *Concurrent-write (or CW for short) on reconfigurable buses is defined as follows:*

Strong CW *Any two signals arriving simultaneously at the same bus are considered concurrent-write.*

Weak CW *Two signals from two different processors arriving simultaneously at the same bus are considered concurrent-write. However, a signal that is split and arrives two times at some bus is not considered concurrent-write.*

Strong CW is implied in the earlier work that established the universality of Collision [7]. Weak CW, however appears just as realistic, for indeed a bunch of fused buses form an electrical, longer bus; then it makes no sense to consider a signal that travels on two different paths, for the signal is simply placed on the bus and propagates along it according to the physical laws of electricity.

As it turns out, the definition of CW makes a significant difference in terms of universality of Collision: under weak CW, Collision is unnecessary on reconfigurable buses; instead, exclusive-write is all we need. This is all put together as follows:

Theorem 4.2.

1. *Under strong CW,* Collision *is universal on reconfigurable buses (directed or undirected), meaning that* Collision *can simulate all the other conflict resolution rules with constant time overhead and with polynomial resources.*

2. *Under weak CW, exclusive-write is universal on reconfigurable buses (directed or undirected), meaning that exclusive-write can simulate all the forms of concurrent-write with constant time overhead and with polynomial resources.*

Proof. The strong CW case has been established for the directed case earlier [7, 8]. The undirected case is easily derived from the proofs supporting the directed case (sketched below): one only need to replace the graph accessibility problem with its undirected variant and nondeterministic logarithmic space with deterministic logarithmic space as done (in different contexts) earlier [5, 6].

The weak CW case has been established for undirected buses elsewhere [6]. For the directed case, we can easily modify the strong CW proof [7] as follows:

The simulation of all the conflict resolution rules is accomplished via a nondeterministic logarithmic space-bounded Turing machine. This machine computes first the content of all the buses, unfused. Successive computations of the graph accessibility problem (which is complete for nondeterministic logarithmic space [20]) then determine which buses are fused. Once the fused buses are determined, the final content of the fused buses is computed. This Turing machine is in turn simulated by a DRMBM which constructs the graph of possible configurations of the machine and then solves the graph accessibility problem to determine the outcome of the computation of the machine. The construction of the graph does not involve any concurrent-write. Overall, the only computation involving concurrent-write is the computation of the graph accessibility problem; all the other computations do not involve concurrent-write at all.

The graph accessibility problem is solved by a DRMBM that works as follows [8]: Finding whether vertex t is accessible from vertex s in a directed graph with n vertices given as an incidence matrix I can be computed in constant time by a DRMBM with $(n^2-n)/2$ processors and n buses. Denote each processor by p_{ij}, $1 \leq i < j \leq n$, and let p_{ij} know the value of both I_{ij} and I_{ji}. Then, each p_{ij}, $1 \leq i < j \leq n$ directionally fuses buses i and j if and only if $I_{ij} = True$; simultaneously, p_{ij} fuses buses j and i if and only if $I_{ji} = True$. It is easily proven that, for any s, t, $1 \leq s, t \leq n$, a signal placed on bus s reaches bus t if and only if vertex t is accessible from vertex s; the content of the emitted signal is immaterial. Indeed, note that the electrical connections between buses actually form the original graph. Bus t will then receive the signal placed on bus s if and only if there exists a path that connects the two vertices. If there are multiple such paths, then the signal will arrive on bus t multiple times; however, there is only one signal in the whole system, so under weak CW there is simply no possibility for a collision to happen! Under the weak CW model the graph accessibility problem is thus computed by an exclusive-write machine. As already mentioned, none of the other computations that simulate all the conflict resolution rules on a DRMBM involve any concurrent-write, so we conclude that exclusive-write is universal and so we complete the proof for DRMBM.

The simulation of a DRN by a Turing machine is similar to the simulation of a DRMBM, except for some supplementary issues related to initial data distribution. These issues however do not change the fact that concurrent-write occurs only in the computation of the graph accessibility problem, which proceeds in the same manner as in the DRMBM case [7]. □

5. Relations Between Several Parallel Computational Models.

5. Relations Between Several Parallel Computational Models. The results established in Section 3 can be combined with earlier results that establish the BSR as being as powerful as the models with directed reconfigurable buses [9]. They can be further combined with the universality of Collision or exclusive-write, as discussed in Section 4. We can then put everything together as follows:

Corollary 5.1. *For any $n \in \mathbb{N}$ and for any $X \in \{\text{Collision}, \text{Common}\}$,*

$$\begin{gathered}X \text{ CRCW PRAM}(\text{poly}(n), O(t(n))) \subseteq \text{Priority CRCW PRAM}(\text{poly}(n), O(t(n))) \subsetneq \\ \text{Combining CRCW PRAM}(\text{poly}(n), O(t(n))) = \text{BSR}(\text{poly}(n), O(t(n))) = \\ \text{DRMBM}(\text{poly}(n), \text{poly}(n), O(t(n))) = \text{DRN}(\text{poly}(n), O(t(n))) = \\ Y \text{ DRMBM}(\text{poly}(n), \text{poly}(n), O(t(n))) = Y \text{ DRN}(\text{poly}(n), O(t(n)))\end{gathered}$$

where Y = CREW under the weak CW assumption and Y = Combining CRCW under the strong CW assumption.

Proof. The first three relations are established in Theorem 3.1. The next two relations are established elsewhere [9]. The last two relations are given by Theorem 4.2. □

6. Real-Time Implications. The class of problems in NL (problems solvable in nondeterministic logarithmic space) with the addition of (any kind of) real-time constraints is denoted by NL/rt [8]. Let rt-PROC$^M(f)$

denote the class of those problems solvable in real time by the parallel model of computation M that uses $f(n)$ processors (and $f(n)$ buses if applicable) for any input of size n. The following strongly supported conjecture is then established.

Proposition 6.1. [8] rt-PROC$^{\mathrm{DRMBM}}(\mathrm{poly}(n)) = \mathsf{NL}/rt$.

We can further define [9] the following classes of models of parallel computation:

$\mathbb{M}_{<GAP}$ contains exactly all the models that cannot compute the graph accessibility problem (GAP) in constant time, and cannot compute in constant time any problem outside NL.

$\mathbb{M}_{\equiv GAP}$ contains exactly all the models that can compute GAP in constant time, but cannot compute in constant time any problem outside NL.

$\mathbb{M}_{>GAP}$ 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 shown (or is even believed) to belong to this class.

Proposition 6.1 is then extended as follows:

Proposition 6.2. [9] *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*

$$\begin{aligned} \text{rt-PROC}^{M_1}(\mathrm{poly}(n)) &\subseteq \mathsf{NL}/rt \\ \text{rt-PROC}^{M_2}(\mathrm{poly}(n)) &= \mathsf{NL}/rt \\ \text{rt-PROC}^{M_3}(\mathrm{poly}(n)) &\supset \mathsf{NL}/rt \end{aligned}$$

The class $\mathbb{M}_{\equiv GAP}$ is established by Proposition 6.2 as the class of complete models for real time. Indeed, exactly all the models in this class can solve all the real-time problems in the presence of no matter how tight timing constraints. This class has been previously populated with the BSR, the DRMBM, and the DRN.

Now Theorem 3.1 adds to $\mathbb{M}_{\equiv GAP}$ the Combining CRCW PRAM. In effect, we are simplifying the real-time domain, showing that the power of reconfiguration as well as the power of the Broadcast instructions are not only not needed to solve real-time problems, but they are also unnecessary. The PRAM thus becomes the most convenient tool for characterizing computations from a real-time perspective, being the simplest of the complete models for real time.

The real-time domain is also simplified with respect to reconfigurable buses, at least under the weak CW model. Indeed, Theorem 4.2 adds the CREW variant of directed reconfigurable buses to $\mathbb{M}_{\equiv GAP}$. Therefore, for directed reconfigurable buses running in real time concurrent-write is not necessary, as exclusive-write is universal.

Our previous characterization of real-time computations (Proposition 6.1) is strengthened by Proposition 6.2 only to the degree that the class $\mathbb{M}_{\equiv GAP}$ is populated; in this paper we therefore further strengthened this result (by adding simpler models to this class).

7. Conclusions.

It has been widely believed that the all-powerful Combining conflict resolution rule does add computational power to the PRAM model, and that the BSR's Broadcast instruction adds further power. We are to our knowledge the first to establish formally a hierarchy of the PRAM variants that both confirms and contradicts the mentioned belief. Indeed, we showed that Combining does add computational power over "lesser" rules. However, we also showed that surprisingly enough the Broadcast instruction does not add computational power over Combining. In fact we established an intriguing collapse of the hierarchy of parallel models at the top of the food chain, where the Combining CRCW PRAM and the BSR turn out to have identical computational power.

This result offers substantial and unexpected aid to the analysis of real-life shared memory massively parallel machines. Indeed, the power of BSR's Broadcast instruction has attracted attention in various areas of parallel algorithms; algorithms with running time as fast as constant have been developed for various problems, most notably in the areas of geometric and graph computations [16, 17]. Of course, constant time algorithms for such practically meaningful problems are very attractive, and so is the power of the BSR. Nevertheless, the model tends to be frowned upon given its apparent implementation complexity, even if very efficient (optimal) implementations have been proposed [3]. We now showed that whether the BSR is feasible or not is irrelevant, as a Combining PRAM with the same performance and with all the attractive properties of the BSR is automatically available—in essence, one can freely choose between these two models, depending on no matter what issues (ranging from practical feasibility to convenience to mere taste) with the formally supported knowledge that the results are portable to the other model.

We are not aware of any massively parallel machine that implements the BSR's Broadcast instruction, but many such machines implement the PRAM model [1, 10, 11, 23]. Our result shows that such an instruction—and thus the full power of the BSR model—can be used in algorithm design and analysis while keeping the analysis or the design practically pertinent.

We also note that most of our proofs are constructive, so we also set the basis for automatic conversion back and forth between models. True, we did not have efficiency in mind so our constructions are likely not to be optimal; historically however suboptimal algorithms have always been optimized sooner or later, so we believe that our—however sub-optimal—implicit conversion algorithms are nonetheless an additional significant contribution (beside establishing the actual equivalence).

This all being said, we note that simulating the Broadcast instruction on a Combining CRCW PRAM implies a (polynomial) blow-up in the number of processors used. This does not change complexity-theoretical results, but does have practical implications. Unfortunately, we believe that such a tight simulation (with constant time slowdown) of the Broadcast instruction is not possible without a blow-up in the number of processors.

As a second significant result we found that exclusive-write can simulate all the forms of concurrent-write on reconfigurable buses under the reasonable weak CW assumption. Concurrent-write has always been regarded as problematic on distributed resources such as buses. We also believe that weak CW is a realistic assumption, given the physical (electrical) realization of reconfigurable buses. VLSI design uses reconfigurable buses extensively. The universality of Collision or exclusive-write (depending on whether we choose the strong or weak CW rule) is significant for VLSI design, as Collision and especially exclusive-write are easily implemented in silicon. More work is necessary for the refinement of the process (of converting a general machine into an exclusive-write machine) before they become useful in practice, but the most important step (or showing that they are possible) is done here.

Beside the significance of our results by themselves (of bringing the BSR into the area of feasible models of parallel computation for massively parallel systems and of showing that exclusive-write is universal on reconfigurable buses), these results become particularly significant in conjunction with previous work on real-time parallel computations [8] and the rest of the characterization of models with reconfigurable buses [9].

Indeed, we showed previously [9] that models with reconfigurable buses have the same computational power as the BSR. Theorem 3.1 then extends this equivalency to the Combining CRCW PRAM. One consequence is that the Combining CRCW PRAM turns out to be just as powerful as reconfiguration. Reconfiguration is however one of the realistic models for VLSI design [15, 22]. We then expect that bringing the Combining CRCW PRAM to the table has the potential of inspiring new design and analysis techniques for VLSI algorithms. That is, our results are not significant only to the area of massively parallel systems, but also to the area of VLSI circuits.

As a consequence of Theorem 3.1, the Combining CRCW PRAM also joins the family of models that can solve all the problems known to be solvable under no matter how tight real-time constraints, and so does the exclusive-write models with directed reconfigurable buses. This strengthens out previous results on real-time computations by establishing the PRAM and the exclusive-write DRMBM or DRN as the simplest complete models of computation for real time.

REFERENCES

[1] F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, and D. Scbeerer, *On the physical design of PRAMs*, Computer, 36 (1993), pp. 756–762.

[2] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[3] S. G. Akl and L. Fava Lindon, *An optimal implementation of broadcasting with selective reduction*, IEEE Transactions on Parallel and Distributed Systems, 4 (1993), pp. 256–269.

[4] S. G. Akl and G. R. Guenther, *Broadcasting with selective reduction*, in Proceedings of the IFIP 11th World Congress, G. X. Ritter, ed., San Francisco, CA, 1989, North-Holland, Amsterdam, pp. 515–520.

[5] 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.

[6] Y. Ben-Asher, D. Peleg, and A. Schuster, *The power of reconfiguration*, Journal of Parallel and Distributed Computing, 13 (1991), pp. 139–153.

[7] S. D. Bruda, *The graph accessibility problem and the universality of the Collision CRCW conflict resolution rule*, WSEAS Transactions on Computers, 10 (2006), pp. 2380–2387.

[8] S. D. Bruda and S. G. Akl, *Size matters: Logarithmic space is real time*, International Journal of Computers and Applications, 29 (2007), pp. 327–336.

[9] S. D. Bruda and Y. Zhang, *Why shared memory matters to VLSI design: The BSR is as powerful as reconfiguration*, in Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Miami, FL, Apr. 2008.

[10] M. Forsell, *A scalable high-performance computing solution for network on chips*, IEEE Micro, 22 (2002), pp. 46–55.

[11] ———, *On the performance and cost of some PRAM models on CMP hardware*, in Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Miami, FL, Apr. 2008.

[12] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Mathematical Systems Theory, 17 (1984), pp. 13–27.

[13] T. Hagerup and T. Radzik, *Every robust CRCW PRAM can efficiently simulate a PRIORITY PRAM*, in Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, 1990, pp. 117–124.

[14] J. JaJa, *An Introduction to Parallel Algorithms*, Addison Wesley, 1992.

[15] R. Miller, V. L. Prasanna-Kumar, D. I. Reisis, and Q. F. Stout, *Parallel computations on reconfigurable meshes*, IEEE Transactions on Computers, 42 (1993), pp. 678–692.

[16] J.-F. Myoupo and D. Seme, *Work-efficient BSR-based parallel algorithms for some fundamental problems in graph theory*, The Journal of Supercomputing, 38 (2006), pp. 83–107.

[17] J.-F. Myoupo, D. Seme, and I. Stojmenovic, *Optimal BSR solutions to several convex polygon problems*, The Journal of Supercomputing, 21 (2002), pp. 77–90.

[18] I. Parberry, *Parallel Complexity Theory*, John Wiley & Sons, New York, NY, 1987.

[19] L. Stockmeyer and U. Vishkin, *Simulation of parallel access machines by circuits*, SIAM Journal on Computing, 13 (1984), pp. 409–422.

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

[21] 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.

[22] R. Vaidyanathan and J. L. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*, Springer, 2004.

[23] X. Wen and U. Vishkin, *PRAM-on-chip: first commitment to silicon*, in Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, 2007, pp. 301–302.

Edited by: Marek Tudruj, Dana Petcu
Received: February 7th, 2009
Accepted: June 28th, 2009