Technical Report No. 97-412

On the Power of Some PRAM Models

Selim G. Akl
Department of Computing and Information Science, Queen's University
Kingston, ON K7L 3N6, Canada

Lin Chen
FRL, P. O. Box 18345, Los Angeles, CA 90018, USA

Abstract

The focus here is the power of some underexplored CRCW PRAMs, which are strictly more powerful than exclusive write PRAM but strictly less powerful than BSR. We show that some problems can be solved more efficiently in time and/or processor bounds on these models. For example, we show that n linearly-ranged integers can be sorted in $O(\log n/\log\log n)$ time with optimal linear work on Sum CRCW PRAM. We also show that the maximum gap problem can be solved within the same resource bounds on Maximum CRCW PRAM. Though some models can be shown to be more powerful than others, some of them appear to have incomparable powers.

Keywords: PRAM; BSR; time and processor bounds; simulation; sorting.
Classification Categories: F.1.1, F.1.2, F.2.2

1 Preliminaries

The focus of this work is on some underexplored Concurrent Read Concurrent Write (CRCW) Parallel Random Access Machine (PRAM) models. These PRAM models differ only in the way of resolving write conflicts. Some of them can be shown to be strictly more powerful than others, whereas some of them appear to have incomparable powers. Two related models, namely, Exclusive Read Exclusive Write (EREW) PRAM and Broadcasting with Selective Reduction (BSR), are also

covered here, though to a smaller extent. The powers of the CRCW PRAMs studied in this paper lie between those of the EREW PRAM and the BSR models, and do not significantly differ from them. It can be shown that an algorithm on any of these CRCW PRAM models can be simulated by an algorithm on EREW PRAM using the same number of processors with a slowdown only logarithmic in the number of processors. All of these models have shared memory with multiple processors. Each processor is identified by a unique integer. On each of these models, the usual assumptions include that any arithmetic or logical operation takes constant sequential time and that there is no error in computation. These assumptions are also taken in this work.

In designing our PRAM algorithms, we often use the following result, usually attributed to Brent [10], to obtain the best time and processor bounds.

Theorem 1 *If a problem can be solved in $O(T)$ time with $O(W)$ work on a PRAM, then the problem can also be solved in $O(T + W/P)$ time with P processors in the same PRAM model.*

(This theorem is true for a large class of standard problems; however, it does not hold for many nonconventional computations, as shown in Akl [2].)

There have been already a large volume of publications on Common CRCW PRAM, Arbitrary CRCW PRAM, and Priority CRCW PRAM. In Common CRCW PRAM, multiple processors are not allowed to write into a single memory location unless they write an identical value. In Arbitrary CRCW PRAM, an arbitrary processor succeeds in writing in case of write conflict. In Priority CRCW PRAM, the processor with the highest priority (lowest indexed or highest indexed) succeeds in writing in case of write conflict. It is well known that Common, Arbitrary, and Priority are in the order of increasing power (see, e.g., [19]).

BSR is a relatively new model of parallel computation. The basic BSR was introduced in Akl and Guenther [4] and an optimal implementation has been obtained [21]. Already several algorithms exist for solving various computational problems on this model; see, for example, Akl and Chen [3], Akl and Guenther [5], Akl and Lyons [6], Chen [12], Gewali and Stojmenović [18], Melter and Stojmenović [22], Semé and Myoupo [23], Springsteel and Stojmenović [24], Stojmenović [25], and Xiang and Ushijima [27]. A generalization of BSR to allow for multiple selection criteria is proposed in Akl and Stojmenović [7]. An implementation of this generalization and several algorithms for solving problems on it are described in Akl and Stojmenović [8]. Recently, Akl and Chen [3] introduced advanced BSR that allows multiple simultaneous BROADCAST instructions.

One frequently used procedure in parallel computing is the prefix computation. The procedure computes $\oslash_{i=1}^{j} a_i$, for $0 < j \leq n$, where $a_1, a_2, \ldots, a_n$ are an array of n input elements in a domain,

and $\oslash$ is an associative operation, which realizes a semigroup on the domain. It is already known that the procedure can be done optimally on EREW PRAM [20] and Common CRCW PRAM [14]. If there are multiple arrays, we can apply prefix computation on all the arrays simultaneously. On BSR model, one BROADCAST instruction on each array suffices. The resource requirements are listed in the following theorem.

Theorem 2 *Prefix computation on one or more arrays can be done in $O(1)$ time with a linear number of processors on BSR, or in $O(\log n)$ time with linear work on EREW PRAM, or in $O(\log n/\log\log n)$ time with linear work on Common CRCW PRAM if each element is represented by $O(\log n)$ bits.*

In many parallel computations on arrays, it is often helpful to perform array packing. Given an array, say $a[1:n]$, consisting of two types of elements, the array packing problem is to obtain another array $b[1:k]$ such that $b[j]$ is the jth element in array a of type 1, for $0 < j \leq k$, where k is the number of elements in array a of type 1. The problem can be solved by invoking the procedure for computing the prefix sums, as shown in Akl [1] and Chen [11]. We list the result in the following theorem.

Theorem 3 *Array packing can be done in $O(\log n)$ time with $O(n/\log n)$ processors on EREW PRAM, or in $O(\log n/\log\log n)$ time with $O(n\log\log n/\log n)$ processors on Common CRCW PRAM, or in $O(1)$ time with $O(n)$ processors on BSR.*

2 Powers of various CRCW PRAMs

A CRCW PRAM is called a Maximum (or Minimum) CRCW PRAM if the maximum (or minimum) of the values processors attempt to write is actually written in case of write conflict. It is easy to see that Maximum CRCW PRAM and Minimum CRCW PRAM have the same computational power, since $\max_{i=1}^{n} a_i = -\min_{i=1}^{n}(-a_i)$ and $\min_{i=1}^{n} a_i = -\max_{i=1}^{n}(-a_i)$.

It is already known that a Maximum CRCW PRAM is at least as powerful as a Priority CRCW PRAM [15]. A possible way of simulating a Priority write on a Maximum CRCW PRAM is described below. Assume, without loss of generality, that the larger the processor number is, the higher priority the processor has. Then a statement such as "processor i Priority writes d_i into m_j" is equivalent to the following statements:

1. Processor i Maximum writes i into m_j;

2. If $m_j = i$ then processor i Maximum writes d_i into m_j;

Note that at Line 2, exactly one processor satisfies the condition $m_j = i$ and there can be no write conflict.

By definition, the maximum of n numbers can be computed in $O(1)$ time with n processors on a Maximum CRCW PRAM. But on a Priority CRCW PRAM, Fich, Meyer auf der Heide, and Wigderson [16] showed that the computation requires $\Omega(\log\log n)$ time with n processors. We can now conclude the following.

Theorem 4 *A Maximum (or equivalently, Minimum) CRCW PRAM can simulate a Priority CRCW PRAM within the same time and processor bounds, but not vice versa.*

A CRCW PRAM is called an And (or Or, or Exclusive-Or) CRCW PRAM if the And (or Or, or Exclusive-Or) of the values processors attempt to write is actually written in case of write conflict. It is easy to see that And CRCW PRAM and Or CRCW PRAM have the same computational power, since $\wedge_{i=1}^{n} b_i = \overline{\vee_{i=1}^{n} \overline{b_i}}$ and $\vee_{i=1}^{n} b_i = \overline{\wedge_{i=1}^{n} \overline{b_i}}$. Each of these PRAMs is further divided into two kinds. For example, we have Logical-And CRCW PRAM and Bitwise-And CRCW PRAM. It is easy to see that a Bitwise-kind CRCW PRAM is at least as powerful as the corresponding Logical-kind CRCW PRAM. Also observe that any algorithm on Common CRCW PRAM also works on Bitwise-And (and Bitwise-Or) CRCW PRAM. If each operand represents a set, then Bitwise-And and Bitwise-Or correspond to Set-Intersection and Set-Union, respectively. Below we will show that Common CRCW PRAM is at least as powerful as Logical-And CRCW PRAM. Observe that a statement such as "processor i Logical-And writes d_i into m_j" can be simulated on a Common CRCW PRAM as follows.

1. Processor i Common writes `true` into m_j.
2. If $d_i =$ `false`, then processor i Common writes `false` into m_j.

Thus, we have proved the following theorem.

Theorem 5 *Bitwise-And (or equivalently, Bitwise-Or) CRCW PRAM is at least as powerful as Common CRCW PRAM, which in turn is at least as powerful as Logical-And (or equivalently, Logical-Or) CRCW PRAM.*

Below we study the maximum gap problem on Maximum CRCW PRAM. The input is n numbers, $n > 1$. The problem is to compute the maximum gap between two such numbers that are neighbors when the n numbers are sorted. The procedure works as follows.

1. Compute the maximum max and the minimum min of n input numbers.

2. Divide the range [min, max] into $(n+1)$ equal subranges, and associate with each of the $(n+1)$ subranges a bucket i, for $0 < i \leq n+1$.

3. Place n input numbers into the buckets based on the values of the numbers (if a number lies on the boundary between bucket i and bucket $(i+1)$, then the number is placed into bucket i), compute the maximum $\max_i$ and the minimum $\min_i$ for each bucket i, and mark all non-empty buckets. (Note that at least one bucket is empty, and there exist positive integers i and j such that buckets $(i+1), \ldots, (i+j)$ are all empty and the maximum gap is $(\min_{i+j+1} - \max_i)$.)

4. Obtain a sequence of k buckets by removing all empty buckets. (Note $1 < k \leq n$.)

5. For $0 < i < k$ do in parallel, Maximum write $(\min_{i+1} - \max_i)$ into *MaximumGap*, where $\min_i$ and $\max_i$ are, respectively, the minimum and the maximum of the value(s) in the current bucket i.

All steps except Step 4 can be done in constant time with linear work. Step 4 is the array packing computation, which takes $O(\log n / \log\log n)$ time and linear work on Common CRCW PRAM by Theorem 3. Since Maximum CRCW PRAM is more powerful than Priority CRCW PRAM (recall Theorem 4), which in turn is more powerful than Common CRCW PRAM, we can therefore conclude the following.

Theorem 6 *The maximum gap problem can be solved in $O(\log n / \log\log n)$ time with optimal linear work on a Maximum CRCW PRAM.*

On the powerful BSR model, Akl and Stojmenović [8] have shown that the maximum gap problem can be solved in constant time with a linear number of processors.

A related problem called the uniform gap problem can be solved faster on a weaker computational model. The problem asks, given n input numbers, for an $n > 2$, if the gap between any two neighboring numbers in the sorted order is always the same. Suppose n input numbers are a_0, a_1, $\ldots$, a_{n-1}. Below is the procedure.

1. Compute the maximum max and the minimum min of n input numbers.

2. Initialize b_i as 0, for $0 \leq i < n$.

3. For $0 \leq i < n$, do the following in parallel: compute $l_i := (n-1)(a_i - \min)/(\max - \min)$; if l_i is an integer, then set b_{l_i} to 1.

4. Compute $d := \wedge_{i=0}^{n-1} b_i$.

5. Answer "yes" if and only if $d = 1$.

Suppose the gap between any two neighboring numbers after sorting is the same. Then the gap is $(\max - \min)/(n-1)$, and the numbers in sorted order are $a'_i = \min + i(\max - \min)/(n-1)$, for $0 \leq i < n$. If $a_i = a'_j$, then $j = (n-1)(a_i - \min)/(\max - \min)$ and is an integer. Conversely, if $j = (n-1)(a_i - \min)/(\max - \min)$ and is an integer, then a'_j equals a_i, one of the input numbers. We have a uniform gap for the n numbers if and only if each a'_i , $0 \leq i < n$, appears in the input. We can now see easily that the above procedure works correctly.

Step 1 can be done by invoking the procedure for finding the maximum in Shiloach and Vishkin [26]. The procedure runs in $O(\log\log n)$ time with linear work. In case of concurrent write, the contending processors write the same bit. So the procedure works on a Logical-And (or equivalently, Logical-Or) CRCW PRAM. All other steps can be done in constant time with optimal work. It follows that the uniform gap problem can be solved in $O(\log\log n)$ time with linear work on a Logical-And CRCW PRAM. If the input is n polynomially bounded integers, then the maximum and the minimum can also be computed in $O(1)$ time with $O(n)$ work on a Logical-And CRCW PRAM [15] [17]. So, in this case, the uniform gap problem can be solved in $O(1)$ time with $O(n)$ work on a Logical-And CRCW PRAM. The procedure is obviously time-optimal and work-optimal. Note that finding the maximum in an array of numbers can be accomplished in $O(1)$ time with optimal work on a Maximum CRCW PRAM. Thus, we have the following theorem.

Theorem 7 *The uniform gap problem can be solved in $O(\log\log n)$ time with optimal linear work on Logical-And (or equivalently, Logical-Or) CRCW PRAM, or in $O(1)$ time with a linear number of processors on Maximum CRCW PRAM. If the input is n polynomially bounded integers, then the problem can be solved in $O(1)$ time with $O(n)$ work on Logical-And (or equivalently, Logical-Or) CRCW PRAM.*

Next, we consider Sum CRCW PRAM. A CRCW PRAM is called a Sum CRCW PRAM if the Sum of the values processors attempt to write is actually written in case of write conflict. The first result we will show involving Sum CRCW PRAM is the following: if a problem can be solved in $O(T)$ time with $O(P)$ processors on a Common CRCW PRAM, then the problem can also be solved on a Sum CRCW PRAM within the same time and processor bounds. This is true because a statement such as "processor i Common writes d_i into m_j" can be replaced by the following statements on a Sum CRCW PRAM.

1. Processor i Sum writes 1 into m_j.

2. Processor i Sum writes d_i/m_j into m_j.

Right after Step 1, the value of m_j equals the number of processors that write into m_j simultaneously. When Step 2 is done, the value of m_j is d_i. Therefore, if a problem can be solved in $O(T)$ time with $O(P)$ processors on a Common CRCW PRAM, so can it on a Sum CRCW PRAM. Nevertheless, if a problem can be solved in $O(T)$ time with $O(P)$ processors on a Sum CRCW PRAM, the problem may not be solved on a Common CRCW PRAM within the same time and processor bounds. For example, with $O(n^{O(1)})$ processors, the computation of Exclusive-Or of n bits requires $\Omega(\log n/\log\log n)$ time on a Common CRCW PRAM and even on a Priority CRCW PRAM [9]; however, on a Sum CRCW PRAM, the Exclusive-Or of n bits b_1, b_2, ..., b_n can be computed in constant time with linear work as follows.

1. For $0 < i \leq n$ do in parallel, Sum write b_i into c.

2. Set *ExclusiveOr* to c mod 2.

We can therefore conclude the following.

Theorem 8 *A Sum CRCW PRAM can simulate a Common CRCW PRAM within the same time and processor bounds, but not vice versa.*

Below we will show that n integers in a linear range can be sorted in $O(\log n/\log\log n)$ time with linear work on a Sum CRCW PRAM. For simplicity, assume, without loss of generality, that the range of n input integers a_1, a_2, ..., a_n is $[1, n]$.

1. Set b_i to 0, for $0 \leq i \leq n$.

2. Sum write 1 into b_{a_i}, for $0 < i \leq n$. (Now b_j gives the number of j's in array a.)

3. Apply prefix sum computation on array b and store the result in array c.

4. For $0 < i \leq n$, do the following in parallel: if $c_{i-1} < c_i$, then write i into d_{c_i}, ..., $d_{c_i - b_i + 1}$. (Now array d is sorted.)

Below is a partial trace on a sample input.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a_i	-	7	2	11	6	2	7	5	4	7	5	11	6	6	7	7	5
b_i	0	0	2	0	1	3	3	5	0	0	0	2	0	0	0	0	0
c_i	0	0	2	2	3	6	9	14	14	14	14	16	16	16	16	16	16
d_i	-	2	2	4	5	5	5	6	6	6	7	7	7	7	7	11	11

In the above procedure, $O(1)$ time and $O(n)$ work suffice for all steps except Step 3, which can be done in $O(\log n/\log\log n)$ time with $O(n)$ work. Thus, we can conclude the following.

Theorem 9 *Linearly-ranged integers can be sorted in $O(\log n/\log\log n)$ time with $O(n)$ work on a Sum CRCW PRAM.*

No known work-optimal PRAM algorithms for integer sorting are faster. The preceding procedure is the first deterministic work-optimal NC algorithm for sorting integers in a linear range, though Chen [12] has shown before that linearly-ranged integers can be optimally sorted in parallel for several special cases such as the case where each integer has a constant upper bounded multiplicity.

Below we will show that with $O(n^2)$ processors on a Sum CRCW PRAM, we can perform sorting faster, in only $O(1)$ time, even if the input is not an array of integers in a linear range. For convenience in describing the algorithm, we assume that the n input numbers are $a_0, a_1, \ldots, a_{n-1}$.

1. Set b_i to 0, for $0 \le i < n$.
2. For $0 \le i, j < n$, do the following in parallel: if $a_i < a_j$ or $a_i = a_j \wedge i < j$, then Sum write 1 into b_j. (Now b_j gives the number of elements that precede a_j in the sorted array.)
3. Set c_{b_i} to a_i, for $0 \le i < n$. (Now array c is sorted.)

The following trace on a sample input shows the behavior of the procedure.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a_i	7	2	11	6	2	7	5	4	7	5	11	6	6	7	7	5
b_i	9	0	14	6	1	10	3	2	11	4	15	7	8	12	13	5
c_i	2	2	4	5	5	5	6	6	6	7	7	7	7	7	11	11

Note that the identical numbers preserve their relative order in the resulting array, so the sort is stable. It is now easy to conclude the following.

Theorem 10 *An array of n numbers can be stably sorted in $O(1)$ time with $O(n^2)$ processors on a Sum CRCW PRAM.*

On a Sum CRCW PRAM, we can also perform matrix multiplication, another important operation, in constant time. Suppose two input matrices A and B are of sizes $p \times q$ and $q \times r$, respectively. Below is the procedure for computing $C = AB$.

1. For $0 < i \leq p$, $0 < j \leq r$ and $0 < k \leq q$, do the following in parallel: Sum write $a_{i,k}b_{k,j}$ into $c_{i,j}$.

We summarize the result in the following theorem.

Theorem 11 *Computation of AB, where A and B are matrices of sizes $p \times q$ and $q \times r$, respectively, can be done in $O(1)$ time with $O(pqr)$ processors on a Sum CRCW PRAM.*

A CRCW PRAM is called a Product CRCW PRAM if the product of the values processors attempt to write is actually written in case of write conflict. We can use Product CRCW PRAM to simulate a statement such as "processor i Common writes d_i into m_j" as follows.

1. Processor i Product writes 2 into m_j.
2. Processor i Product writes $d_i^{\log_{m_j} 2}$ into m_j.

Right after Step 1, the value of m_j is 2^k, where k is the number of processors that write into m_j. So $k = \log_2 m_j$. When Step 2 is done, the value of m_j is d_i.

Therefore, if a problem can be solved in $O(T)$ time with $O(P)$ processors on a Common CRCW PRAM, so can it on a Product CRCW PRAM. However, the converse statement is not true, because with $O(n^{O(1)})$ processors, the computation of Exclusive-Or of n bits b_1, b_2, ..., b_n on Common CRCW PRAM requires $\Omega(\log n / \log\log n)$ time whereas on Product CRCW PRAM the computation can be done in constant time with linear work as follows.

1. Set p to 1.
2. For $0 < i \leq n$ do in parallel, if $b_i = 1$ then Product write (-1) into p.
3. Set *ExclusiveOr* to 0 or 1 depending on whether $p = 1$ or not.

We can now conclude the following.

Theorem 12 *A Product CRCW PRAM can simulate a Common CRCW PRAM within the same time and processor bounds, but not vice versa.*

There can be some other ways of resolving write conflict. Note that each logical function can be defined by a truth table. It is now easy to see that there are 16 distinct binary logical (or bit) functions in all. And, Or, and Exclusive-Or are only 3 of them. Though not all of the 16 binary logical functions are associative, some others such as the operation of checking whether the number of 0's is even also realize semigroups. Note that the operation of Exclusive-Or is equivalent to checking whether the number of 1's is odd. The remaining 4 semigroup operations on {0, 1} are defined as follows.

1. $a \oslash b = 0$.
2. $a \oslash b = 1$.
3. $a \oslash b = a$.
4. $a \oslash b = b$.

A CRCW PRAM that resolves write conflict with one of the last 2 operations is regarded as a Priority CRCW PRAM.

There is exactly one fixed way of resolving write conflict in each of the CRCW PRAM models we have covered so far. Those PRAMs are called static PRAMs. In fact, we can also have a CRCW PRAM with multiple policies for resolving write conflict. For a PRAM of such kind, we specify a resolution policy at each point where concurrent write occurs. These PRAMs are called dynamic PRAMs. We can even have a CRCW PRAM that supports all the resolution policies mentioned above. Such a CRCW PRAM is probably more powerful than a CRCW PRAM that is equipped with only one of the mechanisms for resolving write conflict. We note here that this is precisely the definition given for the PRAM in Akl [2], where the PRAM is viewed as a single model with different forms of memory access, all of which can be used at will according to the needs of the algorithms. This model is justified by noting that if a combinational circuit is used to connect processors to memory locations then all forms of memory access (namely, exclusive read and write, as well as concurrent read and write) have the same depth and size bounds. Algorithms on any PRAM we have mentioned in this paper work on this single dynamic PRAM. By limiting certain forms of memory access on this single platform, we can also design algorithms that work on some static, less powerful PRAMs.

3 Relation between CRCW PRAM and some other models

In this section, we will investigate the relation between CRCW PRAM and some other models of parallel computation, namely EREW PRAM and BSR. Though some CRCW PRAMs are more powerful than some others, below we will see that their powers do not differ significantly. Even the strongest CRCW PRAM can be simulated on an EREW PRAM using the same number of processors with a slowdown only logarithmic in the number of processors. Simulating a Priority CRCW PRAM on an EREW PRAM is already well known (see, e.g., [19]). So we only need to consider how to simulate a write access when the conflict resolution policy is Maximum, Sum, or another semigroup operation. Suppose processor i writes d_i into memory location m_j on a CRCW PRAM, for $0 < i \leq P$. Then on EREW PRAM, we write the triple (j, i, d_i) into m'_i, for $0 < i \leq P$. We then sort all the triples in lexicographically increasing order, which can be done in $O(\log P)$ time with P processors [13] (note that (j, i, d_i) precedes $(j', i', d_{i'})$ if and only if $j < j'$ or $j = j' \wedge i < i'$ in this case, so the comparison between two triples can be done in $O(1)$ sequential time). For each maximal subarray of triples with the same first element j, apply parallel prefix computation, with the specified conflict resolution policy as the semigroup operation, on the last element of the triples. This can be done in $O(\log P)$ time with $O(P)$ work without access conflict (recall Theorem 2). We then write the result corresponding to j into m_j, for all j. We can now conclude the following.

Theorem 13 *A problem can be solved in $O(T \log P)$ time with $O(P)$ processors on an EREW PRAM if the problem can be solved in $O(T)$ time with $O(P)$ processors on a dynamic PRAM.*

Note that sorting and prefix computation in the above simulation can both be done in constant time with P processors on BSR (see, e.g., [3]). We can now conclude the following easily.

Theorem 14 *A BSR can simulate any CRCW PRAM within the same time and processor bounds, but not vice versa.*

4 Conclusion

We have investigated several models of parallel computation. While parallel algorithms developed for a weaker model often have the advantage of being easier to implement on real machines, parallel algorithms for a stronger model may also offer some benefits in practice since they are usually more efficient in terms of resource bounds. We note that the best known EREW PRAM algorithms for certain problems are sometimes no better than the ones obtained by simulating CRCW PRAM

algorithms. So in such cases, a more efficient CRCW PRAM algorithm will imply a more efficient EREW PRAM algorithm. Additionally, the research on parallel models can also help decide which software and/or hardware mechanisms should be implemented in parallel systems to best satisfy our needs within the budgetary constraint. Often a good way to compare and evaluate the performance of various computers is to run sample programs on these machines, in which case efficient parallel algorithms on these types of machines are needed.

Though we have shown that some CRCW PRAMs are more powerful than some others, a number of questions are obviously open. For example, is Sum CRCW PRAM more powerful than Maximum CRCW PRAM? Are there any problems that require more time and/or processors on Product CRCW PRAM than on Priority CRCW PRAM? It will be interesting to search for the answers.

References

[1] S. G. Akl. An optimal algorithm for parallel selection. *Information Processing Letters*, 19:47–50, 1984.

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

[3] S. G. Akl and L. Chen. Efficient parallel algorithms on proper circular arc graphs. *IEICE Transactions on Information and Systems*, E79-D(8):1015–1020, August 1996.

[4] S. G. Akl and G. R. Guenther. Broadcasting with selective reduction. In G. X. Ritter, editor, *Proceedings, 11th IFIP World Computer Congress*, pages 515–520, New York, 1989. North-Holland.

[5] S. G. Akl and G. R. Guenther. Applications of broadcasting with selective reduction to the maximal sum subsegment problem. *International Journal of High Speed Computing*, 3:107–119, 1991.

[6] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry.* Prentice-Hall, 1993.

[7] S. G. Akl and I. Stojmenović. Multiple criteria BSR: An implementation and applications to computational geometry problems. *Proceedings of the Hawaii International Conference on System Sciences*, 2: pages 159–168, 1994.

[8] S. G. Akl and I. Stojmenović. Broadcasting with selective reduction: A powerful model of parallel computation. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, pages 192–222. McGraw-Hill, New York, 1996.

[9] P. W. Beame and J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*, 36(3):643–670, July 1989.

[10] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21:201–208, 1974.

[11] L. Chen. Optimal parallel time bounds for the maximum clique problem on intervals. *Information Processing Letters*, 42(4):197–201, June 1992.

[12] L. Chen. Optimal bucket sorting and overlap representations. *Parallel Algorithms and Applications*, 10:249–269, 1997.

[13] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, August 1988. Correction. *ibid.* 22(6):1349, December 1993.

[14] R. Cole and U. Vishkin. Faster optimal parallel prefix sums and list ranking. *Information and Computation*, 81(3):334–352, June 1989.

[15] D. Eppstein and Z. Galil. Parallel algorithmic techniques for combinatorial computation. *Ann. Rev. Comput. Sci.*, 3:233–283, 1988.

[16] F. Fich, F. Meyer auf der Heide, and A. Wigderson. Lower bounds for parallel random access machines with unbounded shared memory. In F. Preparata, editor, *Advances in Computing Research*, volume 4, pages 1–15. JAI Press, 1987.

[17] F. E. Fich, P. L. Ragde, and A. Wigderson. Simulations among concurrent-write PRAMs. *Algorithmica*, 3(1):43–51, 1988.

[18] L. P. Gewali and I. Stojmenović. Computing external watchman routes on PRAM, BSR, and interconnection models of parallel computation. *Parallel Processing Letters*, 4:83–93, 1994.

[19] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 869–941. North Holland, Amsterdam, 1990.

[20] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

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

[22] R. A. Melter and I. Stojmenović. Solving city block metric and digital geometry problems on the BSR model of parallel computation. *Journal of Mathematical Imaging and Vision*, 5:119–127, 1995.

[23] D. Semé and J.-F. Myoupo. A parallel solution of the sequence alignment problem using BSR model. *Proceedings of the International Conference on Parallel and Distributed Computing*, pages 357–362, 1997.

[24] F. Springsteel and I. Stojmenović. Parallel general prefix computations with geometric, algebraic and other applications. *International Journal of Parallel Programming*, 18:485–503, 1989.

[25] I. Stojmenović. Constant time BSR solutions to parenthesis matching, tree decoding, and tree reconstruction from its traversals. *IEEE Transactions on Parallel and Distributed Systems*, 7:218–224, 1996.

[26] Y. Shiloach and U. Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *J. Algorithms*, 2:88–102, 1981.

[27] L. Xiang and K. Ushijima. Decoding and drawing on BSR for a binary tree from its $i - p$ sequence. To appear in *Parallel Processing Letters*.