# On the Power of Reading and Writing Simultaneously in Parallel Computations*

Rolf Niedermeier and Peter Rossmanith**

Fakultät für Informatik, Technische Universität München
Arcisstr. 21, 80290 München, Fed. Rep. of Germany

**Abstract.** In the standard model of Cook, Dwork, and Reischuk [1] for the derivation of lower bounds, one computation step of a CREW-PRAM consists of a read phase, internal computation, and a write phase. We investigate the case when one step comprises only internal computation and *one* communication phase instead of two, but we have *either* reading *or* writing and internal computation, and thus allow the mixed occurrence of reading and writing (of different memory cells) at the same time. We show that simultaneous reading and writing saves communication phases. In detail, we show that the computation of the OR-function requires exactly $\log n$ "single communication" steps instead of $0.72 \log n$ "double communication" steps on the standard model. We provide a general lower bound of $\log(\deg(f))$ in terms of the degree of a function $f$. We obtain a lower bound of $0.76 \log n$ for the computation of critical functions and present a critical function that can be computed in $0.90 \log n$ steps. Finally we demonstrate a tight correspondence between CROW-PRAM's of the modified form and the decision tree complexity.

## 1 Introduction

The derivation of upper bounds has seen immense progress. The matter with lower bounds, however, seems to be more intricate. In general, only little is known about the time it takes at minimum to solve a given problem on a specific computational model. In this paper we continue and complement work concerning upper and lower bounds for concurrent read, exclusive write parallel random access machines (CREW-PRAM's), which Cook, Dwork, and Reischuk [1] initiated in their seminal paper. The most expensive aspect in parallel computation is communication. Often the number of necessary communcations completely dominates the time required by a parallel algorithm, while the costs of the internal computations play only a minor rôle.

In Cook, Dwork, and Reischuk's setting [1], a computation step of a CREW-PRAM consists of, first, a pure read instruction of all processors, then some internal computation of each processor, and finally a pure write instruction of all processors. A CREW-PRAM is made up of an unlimited number of processors and shared memory cells. A computation consists of $T$ synchronous steps that are performed by all processors parallel in lock-step. Kutyłowski [8] showed that it takes exactly $\phi(n)$

* Research supported by DFG-SFB 0342 TP A4 "KLARA."
** Email: niedermr/rossmani@informatik.tu-muenchen.de

steps on a CREW-PRAM to compute the OR of $n$ variables. The function $\phi(n)$ is defined as $\phi(n) = \max\{\, t \mid F_{2t+1} \leq n \,\} \approx 0.72 \log n$***, where $F_k$ is the $k$th Fibonacci number. Since each step comprises a pure read *and* a pure write phase, this means that actually $1.44 \log n$ communication phases are necessary. As Kutyłowski showed, this cannot be improved. One crucial property of a CREW-PRAM is that all processors read and write in alternating manner. Here the question arises what happens if a computation step consists of only one communication phase that, however, may be *either* one write *or* one read instruction. Subsequently we will refer to this model as a PRAM *with simultaneous reading and writing,* because a write may take place exactly at the same time (i.e., simultaneously) as a read. Clearly, counting the number of computation steps as defined above, our model is at least half as fast as a traditional CREW-PRAM and it never is faster. If we count the two communication phases of a computation step in the standard model as two time units, because in our model a computation step only needs one time unit, our model appears to be faster.

In Section 3 we show that our variant of the CREW-PRAM can compute the OR of $n$ variables much faster than in $1.44 \log n$ steps. Actually, if we denote by $\overline{\mathrm{CREW}}(f)$ the number of steps needed to compute a Boolean function $f$ by a CREW-PRAM with simultaneous reading and writing, $\overline{\mathrm{CREW}}(\mathrm{OR}) = \lceil \log n \rceil + 1$ holds for $n > 1$. This means that a CREW-PRAM can take advantage of the ability to read and write at the same time. Recently, Dietzfelbinger, Kutyłowski, and Reischuk [2] came up with a more general method (compared to Kutyłowski [8]) that delivers lower bounds for Boolean functions in terms of their *degree* [2, 13]. Their main result is $\mathrm{CREW}(f) \geq \phi(\deg f)$. Since almost all functions have degree $n$ and every Boolean function can be computed in $\phi(n) + 1$ steps [2, 12], nearly all functions have time complexity $\phi(n)$ plus an *additive* constant of at most one. We show the corresponding result $\overline{\mathrm{CREW}}(f) \geq \lceil \log(\deg f) \rceil + 1$ by a similar proof technique.

*Critical functions* [1] (that are those functions for which there exists an input such that flipping any single input bit changes the output bit) represent a class of Boolean functions for which up to now in general no matching lower and upper bounds are known. Cook, Dwork, and Reischuk [1] gave a lower bound of $0.44 \log n$ and presented a critical function that can be computed in $0.64 \log n$ steps. Parberry and Yan [11] improved these results. Their lower bound is $0.5 \log n$ and they gave a critical function that can be computed in $0.57 \log n$ steps. In Section 4 a careful adaption of Parberry and Yan's methods yields a lower bound of $0.76 \log n$ for simultaneous reading and writing and an upper bound of $0.90 \log n$. Again the upper bound refers to some particular function. We summarize lower and upper bounds up to constant, additive terms in Table 1.

Whereas it is open whether there exists a general relationship between the computational power of CREW-PRAM's with and without simultaneous reading and writing, such a relation exists for CROW-PRAM's [3]. Fich and Wigderson [5] noted that $\mathrm{CROW}(f) = \log D(f)$, where $D$ denotes the decision tree complexity of $f$. In Section 5 we show that $\overline{\mathrm{CROW}}(f) \approx 1.44 \log D(f)$, yielding the general relationship $\overline{\mathrm{CROW}}(f) \approx 1.44\, \mathrm{CROW}(f)$.

We define the speedup $S_{EW}(f) := 2\,\mathrm{CREW}(f)/\overline{\mathrm{CREW}}(f)$ (resp. $S_{OW}(f) :=$

*** All logarithms in this paper are taken to base 2 if not stated otherwise.

| Type of functions | Lower bounds CREW | $\overline{\text{CREW}}$ | Upper bounds CREW | $\overline{\text{CREW}}$ |
|---|---|---|---|---|
| OR | $0.72 \log n$ [8] | $\log n$ | $0.72 \log n$[1] | $\log n$ |
| critical (easiest) | $0.5 \log n$ [11] | $0.76 \log n$ | $0.57 \log n$ [11] | $0.90 \log n$ |
| Boolean (general) | $0.72 \log(\deg(f))$ [2] | $\log(\deg(f))$ | $0.72 \log n$ [1] | $\log n$ |

**Table 1.** Lower and upper bounds for CREW-PRAM's

$2\,\mathrm{CROW}(f)/\overline{\mathrm{CROW}}(f))$ as the factor how much simultaneous read and write access speeds up the computation of $f$ in terms of communication phases. We have $S_{EW}(\mathrm{OR}) \approx 1.44$ and our results also show that the speedup $S_{EW}(f)$ for critical functions $f$ is at most 1.89. We even can derive from our results that for almost all Boolean functions we have $S_{EW} \approx 1.44$. Eventually, our results suggest the conjecture $1.39\,\mathrm{CREW}(f) = \overline{\mathrm{CREW}}(f)$ for all functions $f$, which will probably be very hard to prove. If, however, the conjecture holds, there will be some consequences: By exploiting some specialties of simultaneous read and write access we can go beyond the possibilities of Parberry and Yan's lower bound proof and obtain $\overline{\mathrm{CREW}}(f) \geq 0.76 \log n$ for each critical function $f$. A general relationship $1.39\,\mathrm{CREW}(f) = \overline{\mathrm{CREW}}(f)$ would imply $\mathrm{CREW}(f) \geq 0.55 \log n$ for all critical functions $f$ and this tentative lower bound nearly coincides with the upper bound $0.57 \log n$ of Parberry and Yan [11]. For CROW-PRAM's life is easier: $S_{OW}(f) \approx 1.39$ for all functions $f$.

Due to the lack of space several proofs had to be omitted. They appear in the full paper.

## 2 Preliminaries

In this section we outline some of the necessary background. For a more general introduction to PRAM's (and also lower bounds for them) we refer to JáJá [6] and Karp and Ramachandran [7]. Let $\mathbf{B}_n$ denote the set of all Boolean functions mapping $\{0,1\}^n$ to $\{0,1\}$ and let $f \in \mathbf{B}_n$. We abbreviate the minimal number of steps necessary to compute $f$ on a CREW-PRAM as $\mathrm{CREW}(f)$. Here one step consists of reading *and* writing. If one step consists *either* of reading *or* writing, we call the minimal number of steps $\overline{\mathrm{CREW}}(f)$. Any number of processors can read from the same memory cell simultaneously, but at most one processor must write into a memory cell in each step. Furthermore, if one processor writes into a memory cell, then no processor is allowed to read from it at the same time.

There is an arbitrary number of different states for each processor. In the beginning of a computation, each processor is in its initial state. The memory cells can hold arbitrary values. Initially, the first $n$ cells contain the $n$ input bits. In the standard CREW-PRAM model [1] each step consists of three parts. First, a processor reads from a memory cell that is determined by the current state of the processor. Then the processor changes its state according to the value read and its old state. Finally, the processor writes into a memory cell according to its new state. See the work of Cook, Dwork, and Reischuk [1] for more details. We study a model whose computation steps only consist of two parts: *either* read *or* write and a state transition.

There are lower bounds in terms of the *degree* of a Boolean function [2], a generalization of Kutylowski's [8] tight bounds for the computation of the OR-function. Smolensky [13] expressed any Boolean function $f(\vec{x})$ as a polynomial of $\vec{x} = (x_1, \ldots, x_n)$ over the reals as follows: Let $S \subseteq \{1, \ldots, n\}$ and $m_S := \prod_{i \in S} x_i$. Every function $f$ is a unique linear combination of monomials:

$$f(\vec{x}) = \sum_{S \subseteq \{1,\ldots,n\}} \alpha_S(f)\, m_S(\vec{x}),$$

where $\alpha_S(f)$ is a positive integer. We define the degree of $f$ as

$$\deg(f) := \max\{\, |S| \mid \alpha_S(f) \neq 0 \,\}.$$

Dietzfelbinger, Kutyłowski, and Reischuk [2] proved $\mathrm{CREW}(f) \geq \phi(\deg(f))$.

## 3 Lower bounds in terms of the degree and the complexity of the OR-function

A naïve algorithm for the OR-function needs $\log n$ steps. Cook, Dwork, and Reischuk [1] found a quicker solution. In this section we show $\overline{\mathrm{CREW}}(\mathrm{OR}) = \lceil \log n \rceil + 1$. This may be a hint for the naturalness of simultaneous reading and writing. We start with a lower bound in terms of the degree of a function similar to Dietzfelbinger, Kutylowski, and Reischuk [2]. We also represent Boolean functions as polynomials, but instead of using partitions of the input space and the "full information" assumption (i.e., each processor remembers everything it has read so far and when it writes, the symbol written includes all information the processor has accumulated so far as well as the processor identification number and the current time step), we use sets that contain the characteristic functions of memory cells or processors at a given point of time.

**Theorem 1.** $\overline{\mathrm{CREW}}(f) \geq \lceil \log \deg(f) \rceil + 1$ *for* $f \in \mathbf{B}_n$ *and* $\deg(f) > 1$.

*Proof.* For $t \geq 0$ let $S_t$ and $M_t$ be sets of Boolean functions defined via the system of recurrences

$$\begin{aligned} S_0 &= \{0, 1\}, \\ M_0 &= \{0, 1, x_1, \ldots, x_n, 1 - x_1, \ldots, 1 - x_n\}, \\ S_{t+1} &= \sum M_t S_t \cap \mathbf{B}_n, \\ M_{t+1} &= S_t + M_t(1 - S_t) \cap \mathbf{B}_n. \end{aligned}$$

In the preceding equations the intersections with $\mathbf{B}_n$ guarantee that we do not leave the set of Boolean functions. By $M_t S_t$ we understand the set $\{\, g \cdot h \mid g \in M_t,\ h \in S_t \,\}$ and by $\sum M_t S_t$ the set of all sums consisting of elements of $M_t S_t$. By induction we get $\deg(M_t), \deg(S_t) \leq 2^{t-1}$ for $t > 0$. So any function $f \in M_t$ fulfills $\lceil \log \deg(f) \rceil + 1 \leq t$. In the following we show that any Boolean function that can be computed in $t$ steps is contained in $M_t$. From this the result follows.

We use the following notation. The Boolean function $s_{p,t,q}(\vec{x})$ has value 1 iff processor $p$ is in state $q$ after step $t$. Similarly, $m_{k,t,a}(\vec{x})$ has value 1 iff global memory cell $k$ contains value $a$ after step $t$. Thus $s_{p,t,q}$ and $m_{k,t,a}$ denote the characteristic functions of processor $p$ and memory cell $k$ on input $\vec{x}$. We prove $s_{p,t,q} \in S_t$ for all $p$ and $q$ and $m_{k,t,a} \in M_t$ for all $k$ and $a$. The new state of a processor depends on its old state and the value read during the step. So $s_{p,t,q}(\vec{x}) = 1$ if there exist $q'$, $k$, and $a$ such that $s_{p,t-1,q'}(\vec{x}) = 1$ and $m_{k,t-1,a}(\vec{x}) = 1$, and processor $p$ reads from memory cell $k$ if in state $q'$ and changes to state $q$ if the value read is $a$. So we can express $s_{p,t,q}$ as

$$s_{p,t,q} = \sum_{(q',k,a)} s_{p,t-1,q'} \cdot m_{k,t-1,a},$$

where the sum is taken over appropriate $(q', k, a)$ as described above. Clearly, $s_{p,t,q}$ is in $\mathbf{B}_n$ and $s_{p,t-1,q'} \in S_{t-1}$ and $m_{k,t-1,a} \in M_{t-1}$ follow from the induction hypothesis. By definition, the left part of the equation is in $S_t$.

There are two reasons why memory cell $k$ contains $a$ after step $t$: (i) Some processor writes the value in step $t$. This happens if there exists a processor $p$ that is in state $q$ and $q$ is a state that makes $p$ write into $k$. A Boolean function $f_1 = \sum_{(p,q)} s_{p,t-1,q}$, where we sum over appropriate $(p, q)$, computes whether this is the case. The set $S_{t-1}$ contains $f_1$, because $S_{t-1}$ is closed under summation of functions with disjoint support sets and each $s_{p,t-1,q} \in S_{t-1}$. (ii) No processor writes into cell $k$ and $k$ already contained $a$ after step $t-1$. The function $f_2 = 1 - \sum_{(q',p)} s_{p,t-1,q'}$ computes the first condition; here $q'$ is a state causing some processor $p$ to write into $k$. Function $f_3 = m_{k,t-1,a}$ computes the second condition. Again we know $1 - f_2 \in S_{t-1}$ and $f_3 \in M_{t-1}$. Combining (i) and (ii) we get $m_{k,t,a} = f_1 + f_2 f_3$, which is contained in $M_t$. We can add since the cases (i) and (ii) are disjoint. □

**Corollary 2.** $\overline{\mathrm{CREW}}(\mathrm{OR}) \geq \lceil \log n \rceil + 1$.

*Proof.* This is a direct consequence of $\deg(\mathrm{OR}) = n$. □

We now give an algorithm for the OR-function to provide a matching upper bound.

**Lemma 3.** $\overline{\mathrm{CREW}}(\mathrm{OR}) \leq \lceil \log n \rceil + 1$.

*Proof.* To compute the OR of $n$ values, we use two procedures to compute the OR of some parts of the input. Procedure $M[i,j]$ computes $x_i \vee \cdots \vee x_j$ in the $i$th memory cell. Procedure $P[i,j]$ computes $x_i \vee \cdots \vee x_j$ such that the $i$th processor knows the result. For $M[i,i]$ there is nothing to do because the $i$th memory cell already contains the result $x_i$. The computation of $P[i,i]$ needs one step. Processor $i$ simply reads $x_i$ from the $i$th memory cell. Now we state how to perform $M[i,j]$ and $P[i,j]$ in general. Let $k = \lfloor (i+j)/2 \rfloor$. To do $M[i,j]$, first do in parallel $M[i,k]$ and $P[k+1,j]$. Now the $i$th memory cell contains $x_i \vee \cdots \vee x_k$ and the $(k+1)$st processor knows $y := x_{k+1} \vee \cdots \vee x_j$. In the next step processor $k+1$ writes 1 into the $i$th memory cell if $y = 1$, and does nothing otherwise. To compute $P[i,j]$, first perform $P[i,k]$ and $M[k+1,j]$ in parallel. Then processor $i$ reads from memory cell $k+1$ and thus knows $x_i \vee \cdots \vee x_j$. A short analysis shows that $M[1,n]$ takes exactly $\lceil \log n \rceil + 1$ steps. □

Since the upper bound of Lemma 3 exactly matches the lower bound from Corollary 2, we have determined the time complexity of the OR-function precisely.

**Corollary 4.** $\overline{\mathrm{CREW}}(\mathrm{OR}) = \lceil \log n \rceil + 1$ *for* $n > 1$.

In the same way we can compute the AND-function. Using the conjunctive normal form for the representation of Boolean functions, we can compute *any* Boolean function with just two more steps than the AND-function itself.

**Theorem 5.** $\overline{\mathrm{CREW}}(f) \leq \lceil \log n \rceil + 3$ *for all* $f \in \mathbf{B}_n$.

*Proof.* (Sketch) For every possible input $\vec{w}$, there will be one processor that knows after $\lceil \log n \rceil + 2$ steps whether $\vec{x} = \vec{w}$. This processor writes the result in step $\lceil \log n \rceil + 3$ into the first memory cell. □

Almost all Boolean functions have degree $n$ [2, 12]. We conclude that $\overline{\mathrm{CREW}}(f) = \log n + c$ for some $c \in \{1, 2, 3\}$ and for almost all Boolean functions $f$.

## 4 Lower and upper bounds for critical functions

A *critical* function is a Boolean function for which an input $I$ exists such that flipping any single bit of $I$ changes the output. The OR-function is a well-known critical function whose *critical input* is the all-zero word. For traditional CREW-PRAM's Parberry and Yan [11] proved a lower bound of $0.5 \log n$ for all critical functions. They also presented a particular critical function and showed how to compute it in $0.57 \log n$ steps. In the following we give a corresponding lower bound of $0.76 \log n$ for simultaneous reading and writing. For this model, we also exhibit a particular critical function and demonstrate how to compute it in $0.90 \log n$ steps.

**Definition 6.** [1]

1. Input index $i$ *affects* processor $p$ *at time* $t$ *with* $I$ if the state of $p$ at time $t$ with input $I$ differs from the state of $p$ at $t$ with input $I(i)$. Input index $i$ *affects* memory cell $k$ *at time* $t$ *with* $I$ if the contents of $k$ at time $t$ with input $I$ differs from the contents of $k$ at $t$ with input $I(i)$.
2. The set of indices that affect processor $p$ at time $t$ on input $I$ is denoted by $P(p,t,I)$. The set of indices that affect memory cell $k$ at time $t$ on input $I$ is denoted by $M(p,t,I)$.
3. $P_t := \max_{p,I} |P(p,t,I)|, \quad M_t := \max_{c,I} |M(c,t,I)|$.

We will subsequently figure out the values of $P_t$ and $M_t$. Note that we can't compute a function with a critical input of length $n$ in time $T$ unless $M_T \geq n$ [1, 11].

As Cook, Dwork, and Reischuk [1] did, we start with an investigation of *oblivious* PRAM's. A PRAM is oblivious if the points of time and addresses of all writes depend only on the input length, but not on the input itself.

**Theorem 7.** *If an oblivious CREW-PRAM with simultaneous reading and writing computes a critical function in $T$ steps, then $n \leq F_T$.*

To prove Theorem 7, we derive the recurrence $P_0 = 0$, $M_0 = 1$, $P_1 = 1$, $M_1 = 1$, $P_{t+1} = P_t + M_t$, $M_{t+1} = P_t$, whose solution is $M_t = F_t$. In contrast to oblivious PRAM's, for *semi-oblivious* PRAM's the decision whether or not it is written into a cell may depend on the input. Again Cook, Dwork, and Reisschuk's [1] techniques yield the following theorem.

**Theorem 8.** *A semi-oblivious CREW-PRAM with simultaneous reading and writing needs at least $\lceil \log n \rceil + 1$ steps to compute a critical function. This lower bound cannot be improved.*

The semi-oblivious algorithm of Lemma 3 computes the OR-function in $\lceil \log n \rceil + 1$ steps. Since the OR-function is critical, the above lower bound $\lceil \log n \rceil + 1$ is optimal. Finally, we come to critical functions on general CREW-PRAM's with simultaneous reads and writes. We have to rework the intricate proof of Parberry and Yan [11] to fit our model.

**Theorem 9.** *Every CREW-PRAM with simultaneous reading and writing requires at least $0.76 \log n$ steps to compute a critical function.*

*Proof.* (Sketch) A careful analysis of Parberry and Yan's proof [11, Theorem 4.7] yields the following recurrence: $P_0 = 0$, $P_1 = 1$, $P_2 = 2$, $M_0 = 1$, $M_1 = 1$, $M_2 = 2$, $P_{t+1} = P_t + M_t$, and $M_{t+1} = 2P_t + P_{t-2} + M_t$. The lower bound resulting from this recurrence is $\log_\alpha n$, where $\alpha \approx 2.47$. □

**Theorem 10.** *There is a critical function $f \in \mathbf{B}_n$ that can be computed by a CREW-PRAM with simultaneous reading and writing in less than $0.90 \log n$ steps.*

*Proof.* Let us call a pair consisting of a memory cell and a processor simply a *pair*. We say that a pair computes a Boolean function $f(\vec{x})$ in step $t$, if after step $t$ the processor is in a distinguished state iff $f(\vec{x}) = 1$ and the memory cell contains $f(\vec{x})$. A pair may compute the OR of two bits in two steps. Starting from this point, we compute bigger and bigger critical functions by pairs and by plain memory cells. If $M_t$ is the size of a critical function computed by a memory cell in $t$ steps and $P_t$ is the corresponding size for a pair, then we already know $M_2 = P_2 = 2$. Now let us assume we have three pairs $(p_1, m_1)$, $(p_2, m_2)$, and $(p_3, m_3)$, each of them computing functions of size $P_t$ after step $t$. Step $t + 1$ consists of the following actions: Processor $p_1$ reads from $m_2$, $p_2$ reads from $m_3$, and $p_3$ reads from $m_1$. In step $t + 2$ each of these three processors writes 1 into a memory cell $m_4$, if the processor computed itself 1, but has read 0. Cell $m_4$ computes a function of size $M_{t+1}$ after step $t + 1$, so $m_4$ computes a function of size $M_{t+1} + 3P_t$ after step $t + 2$. This gives us the first recurrence we need: $M_{t+2} = M_{t+1} + 3P_t$, for $t > 1$.

The second recurrence reads $P_{t+1} = P_t + M_t$ and is established in the following way: Assume that processors $p$ and $p'$ compute the same function. Processor $p$ reads from a copy of $m'$ and simultaneously $p'$ writes 1 into $m'$, if $p'$ has computed 1. The original pair was, say, $(p, m)$, but now we consider the pair $(p, m')$. It computes a critical function of size $P_t + M_t$, if we assume that $(p, m)$ computed a function of size $P_t$ and $m'$ computed a function of size $M_t$. The solution of the recurrence $M_2 = P_2 = 2$, $M_{t+2} = M_{t+1} + 3P_t$, $P_{t+1} = P_t + M_t$ is

$$M_t = \alpha^t + \beta^t \cos(at + b) \quad \text{for } t \geq 2,$$
$$\text{where } \alpha = \frac{2}{3} + \frac{\sqrt[3]{2}}{3\sqrt[3]{79 + 9\sqrt{77}}} + \frac{\sqrt[3]{79 + 9\sqrt{77}}}{3\sqrt[3]{2}} \text{ and } \beta < \alpha.$$

This means that our algorithm takes $\log_\alpha(n) + \Theta(1) \approx 0.89 \log n$ steps to compute the described critical function of $n$ variables. We do not examine the function any further. It suffices that it has the critical input $\vec{0}$ and that there are no write conflicts. □

## 5 CROW-PRAM's

For CREW-PRAM's we conjectured a speedup of 1.44 for all Boolean functions. We couldn't, however, prove it and the reasons that support the conjecture are rather weak. For CROW-PRAM's we can prove that all Boolean functions have the same speedup. Dymond and Ruzzo [3] introduced CROW-PRAM's (concurrent read, owner write) as a frequently occurring subclass of CREW-PRAM's. Here each global memory cell has a unique "write-owner" that is the only processor allowed to write into it. Nisan [10] showed that CREW- and CROW-PRAM's need the same amount of time up to a constant factor to compute a function on a full domain. Therefore, $\text{CREW}(f) = \Theta(\text{CROW}(f))$. A *decision tree* computes a Boolean function by repeatedly reading input bits until the function can be determined from the bits accessed. The decision of which bit to read at any time may depend on the previous bits read. The complexity measure $D(f)$ is the minimal height of a decision tree that computes $f$ [10, 14]. There is a tight connection between CROW-PRAM's and decicion trees: $\text{CROW}(f) = \log(D(f)) + \Theta(1)$ [4, 5]. In this section we prove a corresponding relation for CROW-PRAM's with simultaneous reading and writing.

**Theorem 11.** $\overline{\text{CROW}}(f) = \lceil \log_\alpha D(f) \rceil - O(1) \approx 1.44 \log D(f)$, *where* $\alpha = \frac{1}{2}(1 + \sqrt{5})$.

*Proof.* "$\geq$": Here we assume that the state of a processor reflects its whole history including all values read and written. The contents of a cell encodes the value and the time of the latest write and the state of the processor that has written. Instead of accepting with a 1 in the first cell of global memory and rejecting with a 0, we agree upon accepting on a value in the first memory cell that is a member of an arbitrary "set of accepting values." An ordinary CROW-PRAM can simulate such a modified accepting mechanism with at most two additional steps. We err in the right direction, hence we can assume this full information assumption without loss of generality. We introduce two types of Boolean functions. First, we have $m_{k,t,i}(\vec{x})$ that evaluates to 1 if on input $\vec{x}$ memory cell $k$ contains value $i$ at time $t$. Second, we have $s_{p,t,q}(\vec{x})$ which evaluates to 1 if on input $\vec{x}$ processor $p$ is in state $q$ at time $t$. Furthermore, let $M_t$ denote the maximum height necessary for decision trees to compute all functions of type $m_{k,t,i}(\vec{x})$ and let $P_t$ be the corresponding maximum height for decision trees to compute all functions $s_{p,t,q}(\vec{x})$. By induction we prove the recurrence $P_0 = 0$, $M_0 = 1$, $M_1 = 1$, $M_t = P_{t-1}$, and $P_t = P_{t-1} + M_{t-1}$. The solution is $M_t = F_{t-1}$, from which $\overline{\text{CROW}}(f) \leq \log_\alpha(D(f))$ follows. The basis is immediate. We start directly with the induction step.

First we show $M_t = P_{t-1}$ by proving $m_{k,t,i}(\vec{x}) \in P_{t-1}$. Due to the full information assumption, we can uniquely determine the point of time $t' \leq t-1$ when the last write took place and the state $q$ of the processor $p$ which has written into $k$. Thus we know that cell $k$ at time $t$ has contents $i$ if and only if processor $p$ at time $t'$ is in state $q$. This implies $m_{k,t,i}(\vec{x}) = s_{p,t',q}(\vec{x}) \in P_{t-1}$. Next, we show $P_t = M_{t-1} + P_{t-1}$. We have to show the existence of a decision tree that computes whether a processor $p$ is in state $q$ at time $t$. The height of this decision tree must be at most $M_{t-1} + P_{t-1}$. From $q$ we determine that processor $p$ read last time the value $a$ from memory cell $k$ at time $t'$. We also conclude from $q$ that processor $p$ was in state $r$ at time $t' - 1$. The reverse is also true: $s_{p,t,q}(\vec{x}) = s_{p,t'-1,r}(\vec{x}) \wedge m_{k,t',a}(\vec{x})$. By induction hypothesis there are decision trees that compute $s_{p,t'-1,r}(\vec{x})$ and $m_{k,t',a}(\vec{x})$ since $t' \leq t-1$. Placing one of these decision trees on top of the other yields a decision tree that computes $s_{p,t,q}(\vec{x})$.

"$\leq$": Now we come to the reverse direction of the proof, where a CROW-PRAM simulates a decision tree. The basic idea is "pointer doubling" on decision trees [10]. Without modificaton this approach leads to an algorithm with $2\log(D(f))$ steps. First, for each node of the decision tree a processor sets up a pointer to the successor determined by the variable's value. Then we have to show how to perform pointer doubling on a list of length $n$ in $1.44\log n$ steps. The standard algorithm for pointer doubling, where the processor reads the successor's sucessor and writes it in its own memory cell, requires $2\log n$ steps. We improve this to $1.44\log n$. We avoid alternating statements of the form read, write, read, write, and so on. In lieu, a processor reads repeatedly followed by a final write. This works as follows. Instead of having only one tree where the computation takes place, we initially make $T \approx 1.44\log n$ copies of that tree. We associate one processor with each element in each tree. We will use the first tree only in the first step, the second tree only in first two steps, and so on. In the $t$th step a processor determines its successor's successor by reading from the approporiate place *in the $t$-th copy*. Unless the processor belongs to the $(t+1)$st copy it does not write the value back, but proceeds reading. The owner write property of the above algorithm is obvious.

We prove the running time of approximately $1.44\log D(f)$ as follows. Let $P_t$ denote the distance a pointer, known to a processor, spans after step $t$. Similarily, $M_t$ denotes the distance a pointer, contained in a memory cell, spans after step $t$. All writes are oblivious, so $M_t = P_{t-1}$. The knowledge of a processor that always reads is composed of the knowledge of the processor at time $t-1$ and the value read at time $t-1$. A processor computes its new pointer by combining its old pointer with a pointer read from a memory cell, so $P_t = P_{t-1} + M_{t-1}$. □

## 6 Conclusion

CREW-PRAM's and CROW-PRAM's with simultaneous reading and writing are a reasonable alternative to the standard model of Cook, Dwork, and Reischuk. In general, simultaneous reading and writing saves time. The speedup lies, of course, always between one and two. For CROW-PRAM's we showed that *all* functions have the same speedup 1.39. Whether CREW-PRAM's have also a fixed speedup for all functions, remains an open question. We conjecture a fixed speedup of 1.44. Almost all functions indeed have a speedup of 1.44, but there still could exist functions with speedups of one or two. A fixed, general speedup for CREW-PRAM's would be nice, because results for simultaneous reading and writing would translate to the traditional model. A proof of our conjecture would immediately improve Parberry and Yan's results for critical functions. We used proof techniques already known for the standard model, but also introduced new techniques.

*Acknowledgement* We thank Klaus-Jörn Lange for many helpful discussions and comments.

## References

1. S. A. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15(1):87–97, 1986.
2. M. Dietzfelbinger, M. Kutyłowski, and R. Reischuk. Exact time bounds for computing boolean functions on PRAMs without simultaneous writes. In *Proc. of 2d SPAA*, pages 125–135, 1990.
3. P. Dymond and W. L. Ruzzo. Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. of 13th ICALP*, number 226 in LNCS, pages 95–104. Springer-Verlag, 1986.
4. F. E. Fich. The complexity of computation on the parallel random access machine. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, chapter 20, pages 843–900. Morgan Kaufmann Publishers, 1993.
5. F. E. Fich and A. Wigderson. Toward understanding exclusive read. *SIAM Journal on Computing*, 19(4):718–727, 1990.
6. J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
7. R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, chapter 17, pages 869–932. Elsevier, 1990.
8. M. Kutyłowski. Time complexity of boolean functions on CREW PRAMs. *SIAM Journal on Computing*, 20(5):824–833, 1991.
9. K.-J. Lange. Unambiguity of circuits. *Theoretical Computer Science*, 107:77–94, 1993.
10. N. Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.
11. I. Parberry and P. Yuan Yan. Improved upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 20(1):88–99, 1991.
12. R. L. Rivest and J. Vuillemin. On recognizing graph properties from adjacency matrices. *TCS*, 3:371–384, 1976.
13. R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proc. of 19th STOC*, pages 77–82, 1987.
14. I. Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner, 1987.