Uniform Circuits and Exclusive Read PRAMs*

(Preliminary Version, January 1991)

Inga Niepel Peter Rossmanith

Technische Universität München, Institut für Informatik,
Arcisstr. 21, D-8000 München 2

Abstract

CRCW-PRAMs can be characterized in terms of unbounded fan-in circuits. We introduce the notion of SELECT-gates. Combining this with the concept of an unambiguous circuit we are able to give a circuit equivalent of EREW-PRAMs, thus answering an open question of [SV84]. Moreover, circuits with SELECT-gates characterize CRCW-, CREW-, ERCW-, and EREW-PRAMs in a uniform manner.

Introduction

Parallel random access machines (PRAMs) and uniform circuit families are very important models in parallel complexity theory. Other important models are alternating Turing machines and auxiliary pushdown automata (see e.g. [Ruz80, Coo85]). NC, the class of efficiently parallel solvable problems, can be characterized in terms of all these models.

For the design of efficient parallel algorithms, PRAMs are the most favorite model, while circuits are often used to achieve more theoretical results, such as lower bounds (see e.g. [Has86]). NC is in fact a hierarchy of classes that contains in particular AC^k, classes that are again characterizable by several parallel models including circuits and PRAMs.

There are, however, different possibilities to define PRAMs. AC^k was characterized by CRCW-PRAMs, i.e., PRAMs that allow both simultaneous read and write access to memory cells [SV84]. On the other hand AC^k is also characterized in terms of unbounded fan-in circuits and alternation bounded alternating Turing machines. The most favorite PRAM models are CREW-PRAMs (where concurrent writing is forbidden) and EREW-PRAMs (where concurrent writing and reading is forbidden) [GR89, KR90]. Since no characterizations of these PRAMs in terms of other models were known, Stockmeyer and

*This research was partially supported by the Deutsche Forschungsgemeinschaft, SFB 342, Teilprojekt A4 “KLARA”

Vishkin pointed out that it is an interesting problem to look for such characterizations and, furthermore, that a lack of such characterizations would imply that CRCW-PRAMs are the only reasonable model after all.

Recently, a circuit characterization of CREW-PRAMs could be found by Lange [Lan90]. These circuits are called *unambiguous circuits* and are based on the idea of *vulnerable gates*. Unambiguous circuits are composed of ordinary AND-, and OR-gates of bounded fan-in and additionally of vulnerable gates of unbounded fan-in. For vulnerable OR-gates multiple 1-input is forbidden, while vulnerable AND-gates are not allowed to receive multiple 0-input. The concept of unambiguous circuits seems to be fundamental, because it proved to be useful in the area of semi-unbounded fan-in parallelism, where it could be used to find a circuit characterization for unambiguous auxiliary pushdown automata (see [LR90]). Unfortunately, EREW-PRAMs remained isolated, in the sense that no characterizations for them were found.

In order to find a circuit characterization for EREW-PRAMs it is pointed out in this paper that the simulation of unambiguous circuits of [Lan90] produces concurrent reads only, if the circuits have unbounded fan-out. For this reason we investigate bounded fan-out circuits which turn out to be very restricted in their power, since they can be simulated by logarithmically space bounded Turing machines, if their depth is $O(\log n)$. In order to increase the power of bounded fan-out circuits we invent a new gate that computes the functions $(x_i \wedge y_j)$ for two separate input vectors $(x_i)_{i=1}^{p(n)}$ and $(x_i)_{i=1}^{p(n)}$, where p is some polynomial. With other words such a gate computes the matrix product $x \cdot y^T$ of the vector x and the transposed vector y.

$$(x_1, \ldots, x_m) \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} z_{11} & \cdots & z_{1m} \\ \vdots & & \vdots \\ z_{1m} & \cdots & z_{mm} \end{pmatrix},\ m = p(n),\ z_{ij} = x_i \wedge y_j$$

Our main result is that unambiguous, bounded fan-out circuits which are composed of this new type of gate and AND-, OR-, and NOT-gates exactly characterize EREW-PRAMs, thereby answering the open question of [SV84].

We also show that circuits composed of these gates can characterize PRAMs with all possible combinations of concurrent and exclusive access restrictions if a uniform manner.

Preliminaries

In this paper we consider two different models of parallel computation: PRAMs and uniform circuit families. PRAMs were introduced by Fortune and Wyllie [FW78] as a parallel analogon of random access machines. A PRAM consists of infinitely many RAMs that have access to a shared global memory and work synchronously. Many variants of this model have been considered in literature (see e.g. [SV84, Gol82, Sni82]). The main difference between these models is the restriction of simultaneous read and write accesses to global memory cells.

According to Snir a PRAM that allows both simultaneous read and write access to a memory cell is called CRCW-PRAM (concurrent-read, concurrent-write PRAM). There are several possibilities to solve write conflicts: The COMMON-model requires all values written simultaneously into some memory cell to be equal, whereas in the PRIORITY-model always the lowest numbered processor wins (see [KR90] and [CDHR89] for these and other submodels of CRCW-PRAMs). In CREW-PRAMs each memory cell must receive a new value from at most one processor at each time step, while simultaneous read accesses are still allowed. Finally there are EREW-PRAMs that forbid both simultaneous read and write access.

In addition to these classical models even more restrictive memory access mechanisms were considered: Dymond and Ruzzo introduced CROW-PRAMs, a restriction of CREW-PRAMs [DR86]. In CROW-PRAMs each memory cell is mapped to one processor, its owner. Only this special processor is allowed to write into this memory cell. Analogously to this owner-write PRAM, also a owner-read restriction can be defined which leads to OROW-PRAMs that were investigated in [Ros91]. In OROW-PRAMs each memory cell has a write-owner and a (possibly different) read-owner. Only the write-owner is allowed to write into and only the read-owner is allowed to read from a memory cell.

We will use a definition of PRAMs similar to [SV84]: Each PRAM has an unbounded global memory and infinitely many processors. Each processor has an unbounded local memory and an instruction counter. The program for each processor must be the same and may consist of finitely many instructions. The available instruction set contains add, subtract, division by two, jump, jump if zero, jump if greater than zero, assignment of a constant to a local memory cell, indirect reads and writes from (resp. into) the global memory, and halt. Additionally each processor can read its number. Initially all global and local memory cells contain zero except the first n global memory cells that contain each one bit of the input string and the first local memory cell of each processor, which contains the input length n. The first $p(n)$ processors are started simultaneously at the beginning of their programs, where $p(n)$ is some polynomial. The definition we use is not critical. Though there are many PRAM definitions that differ in details (e.g. [FW78, Sch80, Gol82, Sni82, Vis83, FRW84, EG88]), the accepted classes of languages are the same for resource bounds considered in this paper.*

Subsequently we describe some of the instructions of a PRAM. Herein, L_i stands for a local memory cell and G_a for a global memory cell.

$L_i := const$	Assignment of a constant to local memory cell i
$L_i := L_{i'} + L_{i''}$	Addition of contents of local memory cells i and i', and assignment of the result to local memory cell a
$L_i := L_{i'} - L_{i''}$	Subtraction
$L_i := L_{i'} \wedge L_{i''}$	bitwise AND

*Of course, this is only true, if differences in the restrictions of read and write access are not taken into account.

$L_i := L_{i'} \vee L_{i''}$	bitwise OR
goto k	Continue next step in program line k
goto k if $L_i > 0$	Continue next step in program line k, if local memory cell i contains a positive value
$G_{L_i} := L_{i'}$	Indirect memory write: The contents of local memory cell i' is copied to the global memory cell whose address is in local memory cell i.
$L_i := G_{L_{i'}}$	Indirect memory read: The contents of global memory cell whose address is contained in local memory cell i' is copied into local memory cell i.

Though this is not a minimal instruction set, it is only a small subset of instruction sets normally chosen. However, all further, reasonable instructions can be simulated without timeloss using this subset except the HALT instruction. Instead of using a HALT instruction, we define that all processors halt after a fixed period of running time, which again preserves generality. An input word is accepted, if the first global memory cell contains a non-zero value.

Since we want to use the unit-cost criterion for counting running times (see [Par87]), all values in global and local memory cells must be bounded by some polynomial (a PRAM model fulfilling this restriction is called unit-cost model in [Hag90]). This is no restriction for $O(\log n)$ time bounded PRAMs, but it will be crucial for the simulation of PRAMs with $\omega(\log n)$ running time by circuits. We denote the class of languages accepted by XRYW-PRAMs, $X, Y \in \{C, E, O\}$, in time $O(\log^k n)$ using polynomially many processors by $XRYW^k$.

The second parallel model we are particularly interested in are uniform circuit families. We assume the reader to be familiar with traditional circuit classes, in particular NC^k and AC^k, with or without oracle nodes (see [Coo85] for a survey on these classes). Throughout the paper we assume all circuits to be log-space uniform, the uniformity condition which is called U_{BC}-uniform in [Ruz81].

Our main concern in this paper is to find a circuit characterization of EREW-PRAMs. The first mutual simulation between PRAMs and circuits was given by Stockmeyer and Vishkin in [SV84]. They showed equivalence of size and depth bounded circuits of unbounded fan-in and CRCW-PRAMs:

Theorem 1 [SV84] $AC^k = \mathrm{CRCW}^k$.

At the same time, Stockmeyer and Vishkin posed the question whether there also exist circuit characterizations of CREW-, and EREW-PRAMs. A lack of such characterizations would imply that CRCW-PRAMs might be the only reasonable PRAM model. Recently, a circuit characterization of CREW-PRAMs could be found by Lange [Lan90]. These circuits are called *unambiguous*. Unambiguous circuits have restrictions similar to the exclusive-write restriction of CREW-PRAMs.

Definition 2 $UnambAC^k$ consists of all languages which are accepted by some unbounded fan-in circuit C of polynomial size that fulfills the following restrictions for all possible inputs: No unbounded fan-in OR-gate receives multiple 1-input from its predecessors and no unbounded fan-in AND-gate receives more than one 0. Note that this restriction does not apply to gates with fan-in two.

This new class of circuits characterizes exactly CREW-PRAMs.

Theorem 3 *$UnambAC^k = \mathrm{CREW}^k$.*

In fact, only $UnambAC^1 = \mathrm{CREW}^1$ was shown in [Lan90]. It is possible to extend this result to $UnambAC^k = \mathrm{CREW}^k$, if a PRAM model is used which obeys the reasonable restriction that all values are polynomially bounded. The result of Stockmeyer and Vishkin in [SV84] may be interpreted that this is no restriction for CRCW-PRAMs. Unfortunately, the method relies on the possibility of concurrent memory access and is not extendable to PRAMs with exclusive-access restrictions.

Also we assume all circuits to be leveled throughout this paper. A circuit is called leveled (or layered) if all gates can be arranged in levels such that all gates of level i are only connected to gates in level $i-1$. This restriction is necessary for the proof of Theorem 3 in [Lan90].

Since CREW-PRAMs can be characterized by circuits in which every gate must receive at most one "active value", we can try to use a similar idea to model exclusive read restrictions of PRAMs with circuits. A first idea is to use only circuits of *bounded fan-out*. In such circuits the output of a gate is forwarded to a constant number of other gates. Unambiguous circuits of bounded fan-out can indeed be simulated by CREW-PRAMs since a careful inspection of the proof of Theorem 3 shows that concurrent reads occur only in the case of unbounded fan-out: Processors are assigned to wires of a circuit and these processors read from the memory cell "at the beginning of the wire". In order to avoid concurrent reads these reads can be sequentialized what is only possible if the number of wires is bounded by some constant. Without loss of generality we will assume that bounded fan-out circuit actually have a fan-out of only two.

Definition 4 The classes NC_b^k, $UnambAC_b^k$, and AC_b^k contain all languages recognized by NC^k, $UnambAC^k$, and AC^k circuits, respectively, which have bounded fan-out.

One question arising in this context is whether circuits of unbounded fan-out and circuits of bounded fan-out are equivalent. This could only be shown for NC^k-circuits by now.

Proposition 5 [HKP84] *$NC_b^k = NC^k$.*

It seems very unlikely that Proposition 5 also holds for unbounded fan-in circuits, since we can show that AC_b^k is actually contained in $LOGSPACE$. If AC_b^k and AC^k coincide, this implies that in fact AC^1 equals $LOGSPACE$. This follows immediately from the next proposition and the fact that $LOGSPACE$ is contained in AC^1 (cf. [Sud78])..

Proposition 6 *$AC^1_b \subseteq LOGSPACE$.*

Proof. A path from the output gate of an AC^1_b-circuit to some gate g can be represented by $O(\log n)$ bits. This representation consists of a description of the gate g itself and a string $w \in \{L, R\}^*$ of length $O(\log n)$. This string describes how to follow the path from g to the root by telling whether the next gate on the path is the left or right predecessor. It suffices to distinguish between left and right predecessor because the circuit has fan-out two.

To store a whole path on its work tape enables a log space bounded Turing machine to do a depth-first search through the circuit beginning at its output gate. During this traversal it can be determined whether the circuit accepts. □

But probably, the technique of [HKP84] applies to show $AC^k_b = NC^k$.

The Main Result

We cannot find a circuit characterization for EREW-PRAMs by simply restricting all gates to have *bounded fan-out* due to Proposition 6. But this is not surprising, since the exclusive write restriction could neither be modeled by claiming *bounded fan-in*. To characterize CREW-PRAMs, fan-in was allowed to stay unbounded and only the number of ones an OR-gate receives, was restricted. To characterize EREW-PRAMs we have to find a similar restriction for the output of a gate. A value being emitted by an output of a gate reaches all gates that are connected to this output. This number has to be arbitrary high, since we want many *potential* recipients, just as it is the case for an EREW-PRAM. On the other hand, only one gate must *actually* receive the output. We can modeled this behavior by destroying an asymmetry between inputs and outputs of gates: There is only one output, but many inputs. If we allow a gate to have many outputs and claim that only one of them may emit a 1, we have potentially many recipients, but only one that actually receives this 1. Of course, each output must individually have bounded fan-out. In contrast to EREW-PRAMs, where only *one* processor may read from a memory cell, we actually allow *constantly many* recipients and therefor constant fan-out. This is reasonable because a circuit where and gates have fan-out one cannot distribute any information at all.

We claimed only one active output for each gate, but did not say how it is determined *which one* will be active. In an EREW-PRAM the machine itself computed which processor reads from each memory cell. Similarly, in $UnambAC^k$ circuits, which are equivalent to CREW-PRAMs, the circuit itself has determined for each input of an OR-gate, if it receives a one. Our multi-output gates behave similarly, if we equip them with additional inputs. These inputs receive the number of the active output. Consequently we need gates with a group of outputs and *two groups of inputs*, one group of inputs receives the actual input from which the *output value* is computed, while the other group receives information *on which output* this value appears.

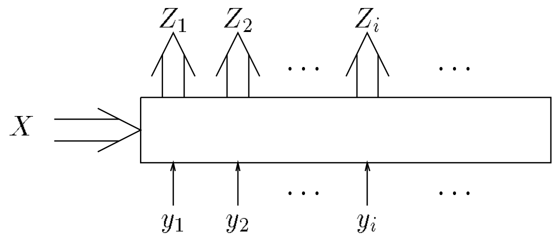

Figure 1: Design of new gate that distributes information

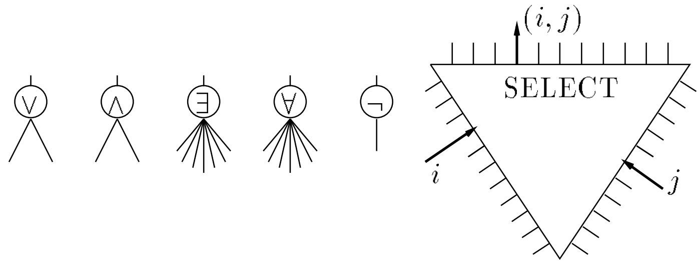

Figure 2: Available gates

The circuit characterization of EREW-PRAMs will be obtained by $UnambAC_b^k$-circuits that additionally contain such multi-output gates. Since a lot of computation can already be performed by $UnambAC_b^k$-circuits itself, it suffices to add only one special type of these new gates. We will design a new gate that is able to distribute information to one of many possible recipients. The "information" is coded unary as a word in $1\,0^* = \{1, 10, 100, \ldots\}$ and is received by a group of inputs X and forwarded to exactly one group Z_i of outputs (see Figure 1). The output group that receives X is selected by additional inputs y_i. If y_i receives a 1, Z_i is the group of outputs that receives X. All other output groups $Z_k, k \neg= i$ will be set to zeroes. Note that X consists in fact of many inputs x_j and Z_i of many outputs z_{ij}. Actually there is no real difference between inputs x_j and y_i since a simple analysis shows $z_{ij} = 1 \iff x_j = 1 \wedge y_i = 1$. Subsequently we will respect this symmetry also in figures that illustrate circuit constructions by a symmetric drawing in contrast to Figure 1. While AND-, OR-, and NOT-gates are drawn as circles containing appropriate symbols (we distinguish gates of bounded (resp. unbounded) fan-in by using $\wedge$, $\vee$, and $\neg$ (resp. $\forall$ and $\exists$), the new gates are drawn as triangles with outputs on top and the two groups of inputs on the left and right sides (see Figure 2).

These gates are called SELECT-gates and have two groups of inputs $x_1, \ldots, x_m$, resp. $y_1, \ldots, y_m$, and one group of outputs. The outputs are denoted by (x_i, y_j), $1 \leq i, j \leq m$. A SELECT-gate computes functions $(x_i, y_i) = x_i \wedge y_j$. Since we mainly consider unambiguous circuits, we require that for every SELECT-gate $x_i = 1$ (resp. $y_j = 1$) holds for at most one i (resp. one j). This automatically guarantees that only one output carries a one.

Note that the two groups of inputs are independently regarded when defining unambiguity. Since each group of inputs may individually receive a one, a SELECT-gate may actually receive two ones in an unambiguous circuit.

The number of inputs m for a SELECT-gate is a polynomial in n, the number of inputs to the circuit, so SELECT-gates build up a family of circuits. Formally SELECT-gates in a circuit are handled as oracle-nodes as defined in [Coo85]. The function computed by SELECT-gates is called SEL.

Definition 7 The class $UnambAC_b^k(SEL)$ consists of all languages which are recognized by some unambiguous unbounded fan-in, bounded fan-out uniform circuit family which contains additional SELECT-gates.

Lemma 8 $UnambAC_b^k(SEL) \subseteq \mathrm{OREW}^k$.

Proof. The basic idea of the simulation is the same as in [SV84]. Using the uniformity of a given circuit, we first assign one processor to each wire and one memory cell to each OR-, AND-, and NOT-gate. SELECT-gates will be treated differently. The memory cells that are assigned to gates initially have all value 0, except the memory cells assigned to AND-gates which are preassigned to one. Additionally, there is one memory cell for each bit of the input of the circuit. A processor assigned to a wire leading from gate g_1 to g_2 does the following. First, it waits for some time proportionally to the distance between gate g_1 and the inputs of the circuit. This distance is easily determined since we claimed all circuits to be leveled. After this time has elapsed, the processor can be sure that the memory cell assigned to g_1 contains the correct value for g_1., Now the processor reads that value and proceeds according to the type of g_2. If g_2 is a NOT-gate, the value is negated and written into the memory cell assigned to g_2. If g_2 is some OR-gate or AND-gate, the value is simple copied into g_2's memory cell, however, in the case of an OR-gate only if the value is 1 and in the case of an AND-gate only if the value is 0. The simulating PRAM is clearly exclusive-write since AND-gates receive at most one 0 and OR-gates at most one 1. It is also owner-read. Each memory cell has *two* read-owners due to fan-out two of the circuit, but this can be easy changed to only one read-owner (see also [Ros91]).

The more interesting part of the simulation is the handling of SELECT-gates. Here again, we assign a memory cell *to each output* of the SELECT-gate and, additionally, one *to each input*. There are two additional memory cells L and R. Memory cell L will contain the number of the input on the left side which receives a 1, and R will contain the same number for the right side. For each gate we also need additional processors $l_1, \ldots, l_m$, $r_1, \ldots, r_m$, and w. Processor l_i reads from the i^{th} left-hand input. If it reads a 1, it subsequently writes the number i into memory cell L. Processors $r_1, \ldots, r_m$ do the same for the right inputs and memory cell R. After both L and R contain the numbers

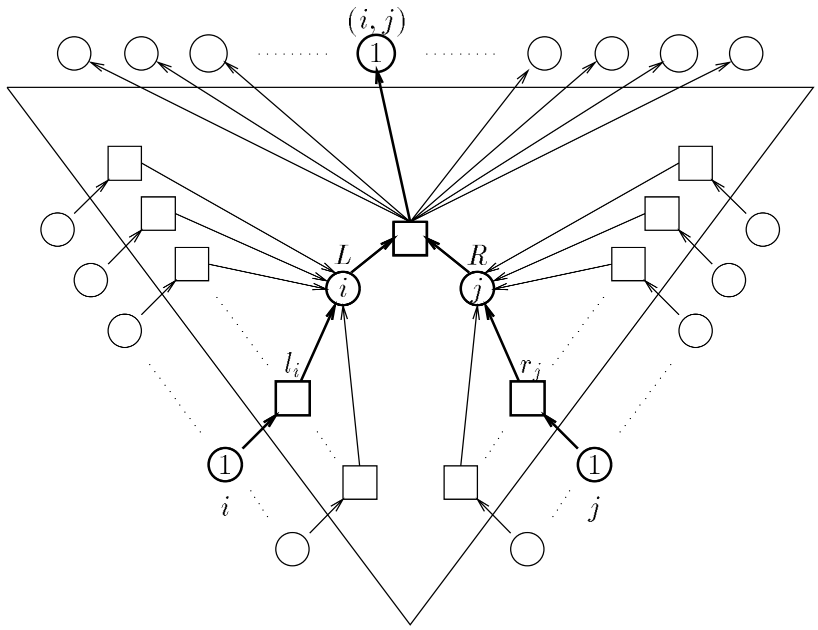

Figure 3: Simulation of SELECT-gates

of active inputs, processor w reads these numbers from L and R and computes which output has to become 1. To compute the address of the memory cell, where it has to write a 1, the processor has to do a multiplication. We can assume that an OREW-PRAM can do multiplication of *polynomially bounded values* without loss of generality. One possibility is to use table look-up. In Figure 3 the above simulation is illustrated using circles for memory cells and squares for processors.

Since the left and right side of each SELECT-gate receive at most one 1 each, it is again clear that this simulation is OREW. Finally, we have to state how the simulation of AND-, OR-, and NOT-gates works together with the simulation of SELECT-gates: A processor assigned to a wire that leads to an input of a SELECT-gate simply has to copy the value it reads to the memory cell assigned to the input of the SELECT-gate. $\square$

Lemma 9 $\mathrm{OREW}^k \subseteq UnambAC_b^k(SEL)$.

Proof. Dymond and Ruzzo described the behavior of a CROW-PRAM by mutually recursive functions in [DR86]. These functions can be evaluated by a deterministic auxiliary pushdown automaton. A similar technique was used in [Lan90] to describe CREW-PRAMs. In contrast to [DR86], here predicates describing the behavior of a CREW-PRAM are evaluated by $UnambAC$-circuits rather than AuxPDAs. In case of EREW-PRAMs we can again define an appropriate set of mutually recursive predicates:

$\langle GLOBAL(t,a), b\rangle$ Global memory cell a contains value b in step t

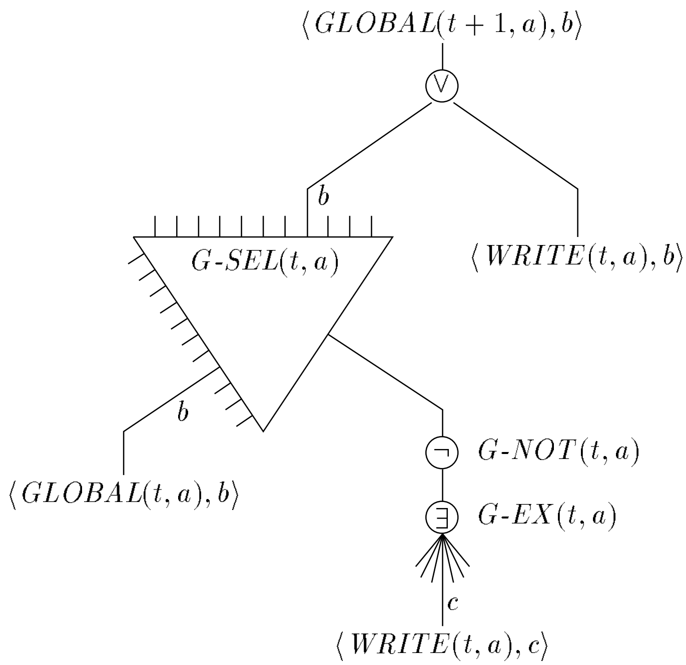

Figure 4: Construction of $\langle GLOBAL(t+1,a),b\rangle$

$\langle LOCAL(p,t,i),a\rangle$	Local memory cell i of processor p contains value a in step t
$\langle ADD(p,t,i),a\rangle$	Processor p performs an add instruction in step i resulting in value a, which is written into local memory cell i
$\langle STATE(p,t),j\rangle$	Processor p performs instruction of program line j in step t
$\langle READ(p,t,i),a\rangle$	Processor p reads value a from global memory and writes it into local memory cell i
$\langle WRITE(t,a),b\rangle$	Value b is written into global memory cell a in step t

To make the subsequent construction as transparent as possible, we will use the following denotations: The letter t always denotes a time step, p denotes a processor number, i, i', and i'' denote local memory addresses, j and k are program lines, and a, b, and c denote arithmetic values or global memory addresses. Since we have to do arithmetic computations with global memory addresses, it is not possible to distinguish between the latter two.

Given an EREW-PRAM that uses polynomially many processors and $\log^k n$ time, we can determine whether it accepts or rejects some input word w, if we know whether the predicate $\langle GLOBAL(\log^k n, 0), 1\rangle$ is true. We will construct an $UnambAC_b^k(SEL)$-circuit that computes exactly this predicate. The circuit contains gates for each possible predicate given above and some additional gates to recursively compute their values. The interconnection of gate $\langle GLOBAL(t+1,a),b\rangle$ is shown in Figure 4. The gate itself is an OR-gate with two inputs. This corresponds to two reasons why global memory cell a contains value b in step $t+1$: One possibility is, that some processor wrote value b into cell a in step t. This corresponds to gate $\langle \mathit{WRITE}(t,a), b\rangle$, which is one input of $\langle \mathit{GLOBAL}(t+1,a), b\rangle$. The other possibility is, that nobody wrote into memory cell a in step t and cell a already contained value b in step t. Therefore the second input of $\langle \mathit{GLOBAL}(t+1,a), b\rangle$ is connected to the b^{th} output of $\mathit{G\text{-}SEL}(t,a)$. Since $\mathit{G\text{-}SEL}(t,a)$ is a select gate, its b^{th} output is one, if and only if $\langle \mathit{GLOBAL}(t,a), b\rangle$ and $\langle \mathit{G\text{-}NOT}(t,a)\rangle$ are one. $\langle \mathit{GLOBAL}(t,a), b\rangle$ means that memory cell a already contained value b in step t, and $\mathit{G\text{-}NOT}(t,a)$ means that *nobody* wrote into global memory cell a in step t. $\mathit{G\text{-}NOT}(t,a)$ computes the complement of $\mathit{G\text{-}EX}(t,a)$, so $\mathit{G\text{-}EX}(t,a)$ has to compute whether *somebody* wrote some value c into cell a in step t. If this happens, then $\langle \mathit{WRITE}(t,a), c\rangle$ is true and $\mathit{G\text{-}EX}(t,a)$ receives a one from it.

By now, we only proved the correct behavior of the $\langle \mathit{GLOBAL}(t+1,a), b\rangle$-subcircuit. There still has to be shown unambiguity and bounded fan-out of involved gates: Unambiguity has only to be checked for two gates having unbounded fan-in. At first there is $\mathit{G\text{-}EX}(t,a)$, which is an OR-gate and receives values from $\langle \mathit{WRITE}(t,a), c\rangle$ for all possible c. Since in an EREW-PRAM only one processor is allowed to write into memory cell a in step t, consequently there exists at most one value c that is written into it and unambiguity of $\mathit{G\text{-}EX}(t,a)$ is assured. Second, there are the left-hand inputs of $\mathit{G\text{-}SEL}(t,a)$. Here unambiguity can be shown very easily, too. At its left side $\mathit{G\text{-}SEL}(t,a)$ receives inputs from $\langle \mathit{GLOBAL}(t,a), b\rangle$ for all b, but since memory cell a can contain only one value b, at most one of these inputs can carry a one. At last, we observe that the claim for bounded fan-out is also fulfilled. One has only to check whether the outputs of all gates involved in Figure 4 lead only to constantly many inputs of other gates. In the case of $\langle \mathit{GLOBAL}(t+1,a), b\rangle$ we have only fan-out two by now ($\langle \mathit{WRITE}(t,a), b\rangle$ leads to both $\langle \mathit{GLOBAL}(t+1,a), b\rangle$ and to b^{th} input of $\mathit{G\text{-}EX}(t,a)$, all other gates have only fan-out one in Figure 4). Of course, some of these gates also appear in following constructions and consequently fan-outs will sum up. Since there is a fixed number of constructions, the actual fan-out is still constant.

In the case of $\langle \mathit{READ}(p,t,i), a\rangle$ the construction will be significantly more complicated. Its interconnection is shown in Figure 5. $\langle \mathit{READ}(p,t,i), a\rangle$ computes whether processor p reads value a from global memory into its local memory cell i in step t. This can only be the case, if processor p carries out a read instruction in step t. $\langle \mathit{READ}(p,t,i), a\rangle$ is itself an OR-gate whose inputs correspond to program lines of the fixed PRAM program. So $\langle \mathit{READ}(p,t,i), a\rangle$ receives a 1 from its j^{th} input, if p reads a into local memory cell i while executing the instruction in program line j. This is only possible if program line j contains a read instruction of the form $L_i := G_{L_{i'}}$, with some arbitrary i'. If j is not of this shape, the jth input of $\langle \mathit{READ}(p,t,i), a\rangle$ is simply left out. In Figure 5 this is illustrated by "[j: $L_i := G_{L_{i'}}$]". This means: "Include this wire only, if program line j contains $L_i := G_{L_{i'}}$. In general "[...]" next to a wire means "include this wire only, if ... holds".

Let us assume program line j contains in fact an instruction $L_i := G_{L_{i'}}$. Then the j^{th} input of $\langle \mathit{READ}(p,t,i), a\rangle$ is connected to a^{th} output of $\mathit{R1\text{-}SEL}(p,t,j)$. Since $\mathit{R1\text{-}SEL}(p,t,j)$ is an SELECT-gate and its a^{th} left-hand input is connected

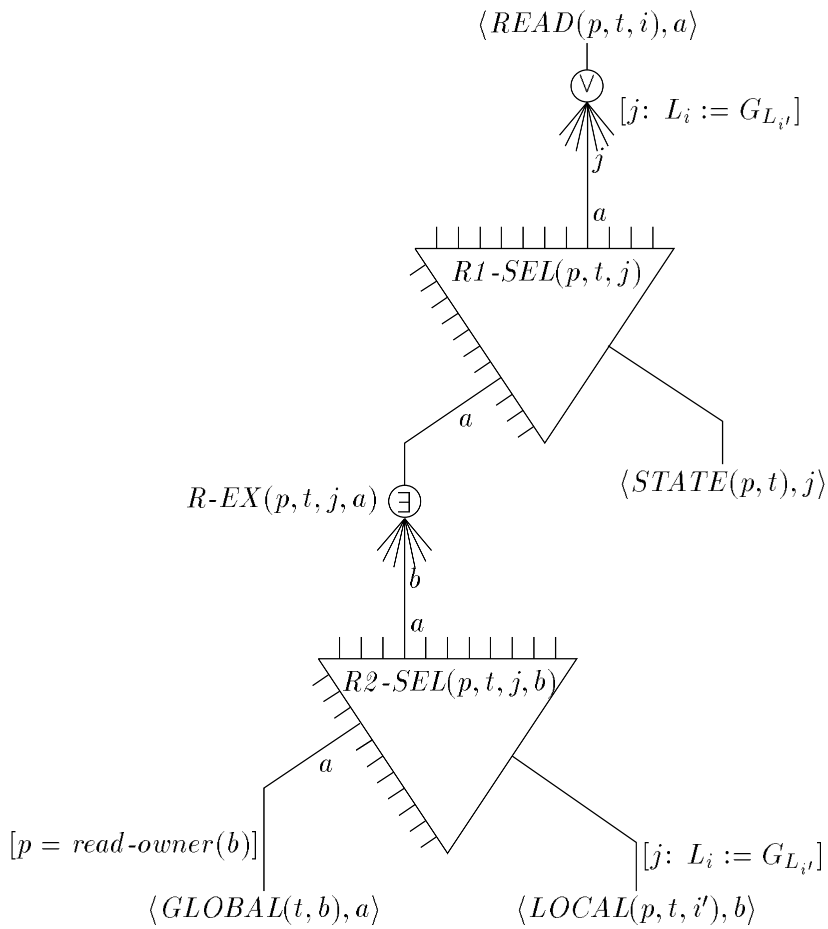

Figure 5: Construction of $\langle READ(p,t,i),a\rangle$

to $R\text{-}EX(p,t,j,a)$, the a^{th} output of $R1\text{-}SEL(p,t,j)$ is in fact a conjunction of $R\text{-}EX(p,t,j,a)$ and $\langle STATE(p,t),j\rangle$. $R\text{-}EX(p,t,j,a)$ means processor p reads value a from global memory and writes it into some local memory cell, if j is the program line being executed. $\langle STATE(p,t),j\rangle$ means "j is the program line being executed" by processor p in step t. $R\text{-}EX(p,t,j,a)$ is an unbounded fan-in OR-gate whose inputs correspond to global memory cells from where the value a is fetched. So the b^{th} input of $R\text{-}EX(p,t,j,a)$, which is connected to a^{th} output of $R2\text{-}SEL(p,t,j,b)$, manages the case of reading value a from global memory cell b and writing it into local memory cell i by the instruction $L_i := G_{L_{i'}}$ in program line j. For this, two conditions have to be met: Local memory cell i' must contain value b and global memory cell b must contain value a. This is exactly what output a of $R2\text{-}SEL(p,t,j,b)$ computes, which is effectively a conjunction of $\langle LOCAL(p,t,i'),b\rangle$ and $\langle GLOBAL(t,b),a\rangle$.

Again we have to check, whether all involved gates have bounded fan-out. For most gates, this is as easy as for the $\langle GLOBAL(t+1,a),b\rangle$ construction, but it is a little more complicated for $\langle GLOBAL(t,b),a\rangle$ and $\langle LOCAL(p,t,i'),b\rangle$. $\langle GLOBAL(t,b),a\rangle$ is connected to a^{th} left input of all *R2-SEL*(p,t,j,b), for all j and all p, with $p =$ *read-owner*(b). Since there is only one p with $p =$ *read-owner*(b), $\langle GLOBAL(t,b),a\rangle$ is connected to as many *R2-SEL*(p,t,j,b) gates as program lines j exist. This number is constant, since the program size is always constant. Also, fan-out of $\langle LOCAL(p,t,i'),b\rangle$ is bounded by the program length for the same reasons.

A circuit is only correct, if only one wire leads to each input. We must be very careful to fulfill this claim everywhere in our construction, too. However, in most cases it is obvious that this property holds. In Figure 5 it is not so obvious for the right input of *R2-SEL*(p,t,j,b), since this input is connected to $\langle LOCAL(p,t,i'),b\rangle$ with arbitrary i'.. But this connection does exist only, if program line j is of shape $L_i := G_{L_{i'}}$, with some arbitrary i. So i' is a function of j in this case and actually only one wire reaches the right input of *R2-SEL*(p,t,j,b). Unambiguity is clear for *R2-SEL*(p,t,j,b) and *R-EX*(p,t,j,a). *R1-SEL*(p,t,j) is also unambiguous, since processor p can read at most one value a from global memory in step t, if we assume it executes some fixed program line j.

The next construction we examine, is the interconnection of $\langle WRITE(t,a),b\rangle$, which is depicted in Figure 6. $\langle WRITE(t,a),b\rangle$ means value b is written into global memory cell a in step t. This is the case if and only if some processor p does this write access. $\langle WRITE(t,a),b\rangle$ is an unbounded fan-in OR-gate, whose inputs correspond to processors in charge for this write access. Input p of $\langle WRITE(t,a),b\rangle$ is connected to *W-OR*(p,t,a,b). *W-OR*(p,t,a,b) means processor p writes value b into cell a in step t, its j^{th} input is connected to $(a,b)^{\text{th}}$ output of *W1-SEL*(p,t,j). This output computes the conjunction of $\langle STATE(p,t),j\rangle$ and $(a,b)^{\text{th}}$ output of *W2-SEL*(p,t,j). The former checks whether processor p really executes program line j in step t, while the latter verifies whether local memory cell i' contains b and local memory cell i contains a. Since this computation must only be valid, when program line j is executed and this program line contains the instruction $G_{L_i} := L_{i'}$, the $(a,b)^{\text{th}}$ output of *W2-SEL*(p,t,j) behaves correctly. A careful inspection shows that all gates in Figure 6 are unambiguous and have bounded fan-out.

The interconnection of $\langle ADD(p,t,i),a\rangle$ is shown in Figure 7. $\langle ADD(p,t,i),a\rangle$ computes, whether processor p writes value a into local memory cell i as a result of an add instruction executed in step t. This happens if and only if an instruction $L_i := L_{i'} + L_{i''}$ is executed and local memory cells i' and i'' belonging to this instruction contain values that yield a when added. For an instruction j: $L_i := L_{i'} + L_{i''}$ gate *A-EX*(p,t,j,a) computes, whether values of cells i' and i'' sum up to a. This is done by a disjunction of all possible combinations of (b,c) with $b+c=a$. Output (b,c) of *A2-SEL*(p,t,j) means cell i' contains b and cell i'' contains c. At last, it has to be checked, whether this program line j is really executed in step t, what is done by *A1-SEL*(p,t,j) together with $\langle STATE(p,t),j\rangle$. All we have finally to do, is to combine this for all possible program lines by an OR-gate to get $\langle ADD(p,t,i),a\rangle$.

There are two predicates left for which we have to give circuit constructions: $LOCAL(p,t,i),a>$ and $\langle STATE(p,t),j\rangle$. The construction of $\langle LOCAL(p,t,i),a\rangle$ is

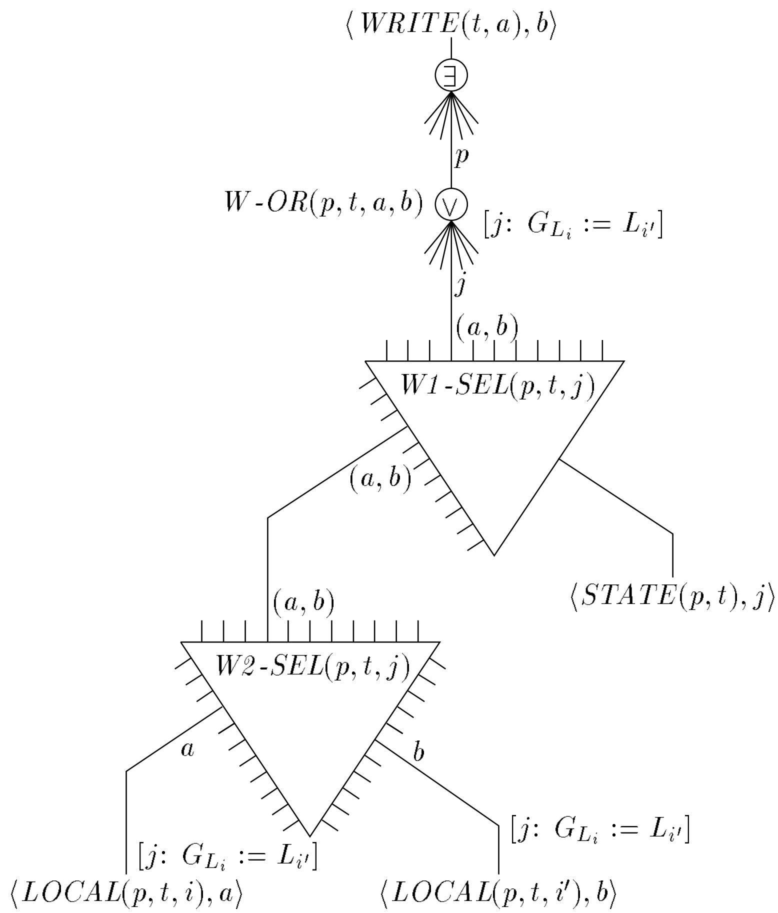

Figure 6: Construction of $\langle WRITE(t,a),b\rangle$

illustrated in Figure 8, it works just like the construction of $\langle GLOBAL(t,a),b\rangle$.

The construction of $\langle STATE(p,t),j\rangle$ is given in Figure 9. $\langle STATE(p,t),j\rangle$ means that processor p executes the j^{th} instruction of its program in step $t+1$ or alternatively that its instruction counter contains the value j. Obviously there are four possibilities for this to happen:

1. In step t instruction $j-1$ was executed which is no jump instruction,
2. In step t instruction $j-1$ was executed which *is* a conditional jump instruction, but the jump is not performed since the appropriate condition is not met,
3. In step t an instruction "goto j" in program line k was executed,
4. In step t an instruction "goto j if $L_i \geq 0$" in line k was executed and the condition $L_i \geq 0$ was met.

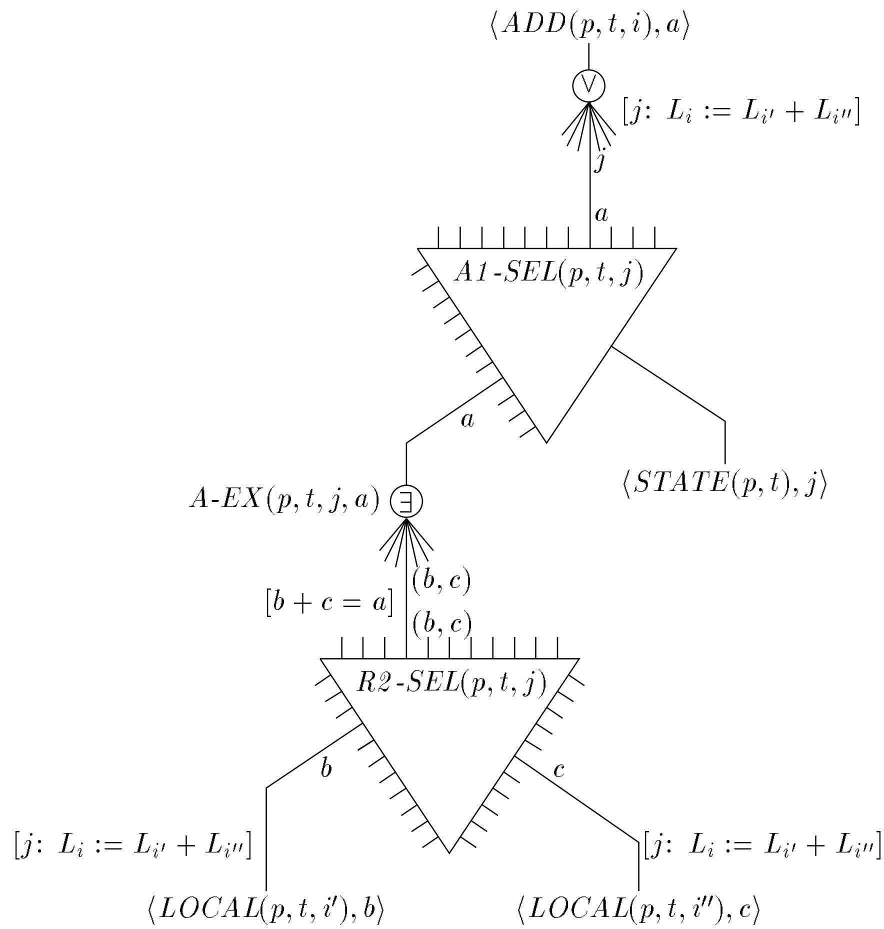

Figure 7: Construction of $\langle ADD(p,t,i), a\rangle$

These four possibilities are computed by the four gates $\langle STATE(p,t), j\rangle$, $S1\text{-}EX(p,t,j)$, $S1\text{-}OR(p,t,j)$, and $S2\text{-}OR(p,t,j)$ which are disjuncted in order to construct $\langle STATE(p,t), j-1\rangle$. $\square$

To prove our main result we need only one more proposition, the equivalence of EREW- and OREW-PRAMs.

Proposition 10 [Ros91] $\text{EREW}^k = \text{OREW}^k$.

Proof. An exclusive read can be simulated by owner read and exclusive write as follows: We assign one extra processor to each memory cell. This processor becomes the read-owner of that memory cell. If some processor wants to know the contents of a memory cell it sends simply a question to the read-owner of that cell. The read-owner reads the value and forwards it to the processor who asked for it. $\square$

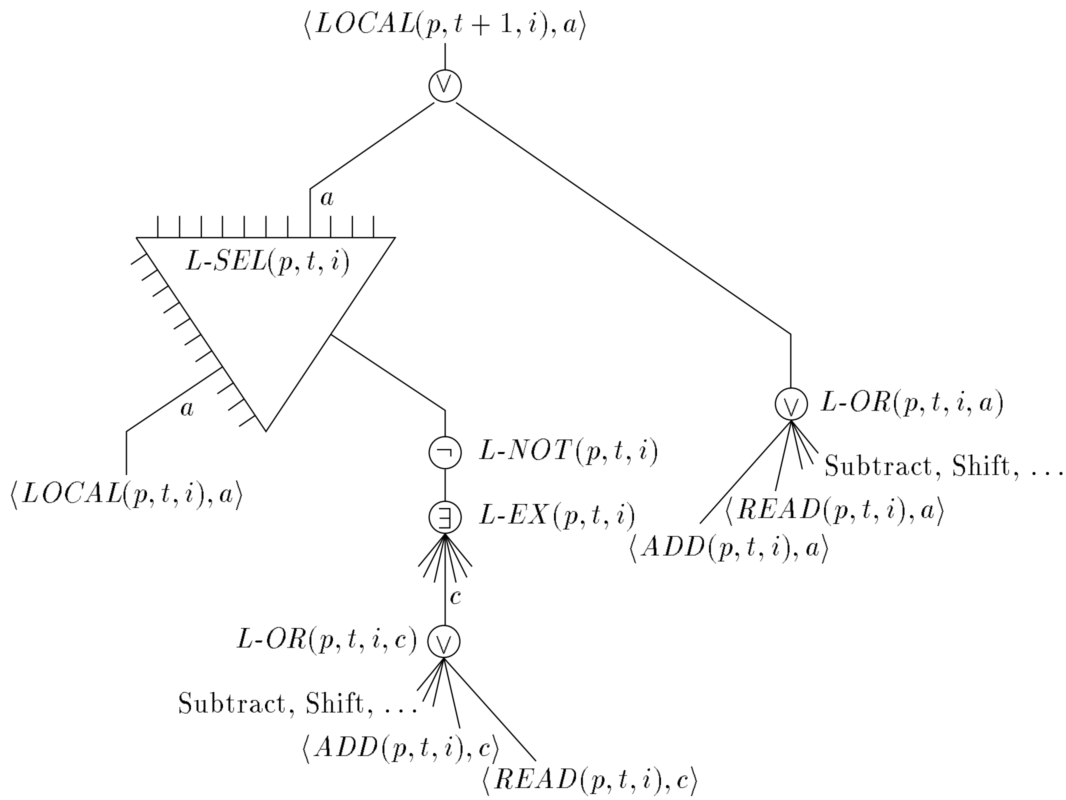

Figure 8: Construction of $\langle LOCAL(p,t,i),a\rangle$

From Lemma 9 and 8 and Proposition 10 our main result follows:

Theorem 11 $UnambAC_b^k(SEL) = \mathrm{EREW}^k$.

Other Applications of SELECT-Circuits

The presence of SELECT-gates makes a great difference for AC_b^k-circuits. While plain AC_b^1 is contained in $LOGSPACE$, $AC_b^1(SEL)$ contains $UnambAC_b^1(SEL)$ which is equivalent to EREW^1. Each SELECT-gate is indeed a special NC^0-circuit. However, the next proposition makes clear that a general NC^0 oracle is a lot more powerful than ordinary SEL oracle nodes.

Proposition 12 $AC_b^k(NC^0) = AC^k$,
$UnambAC_b^k(NC^0) = UnambAC^k$.

Proof. NC^0 oracles can give any circuit arbitrary fan-out. □

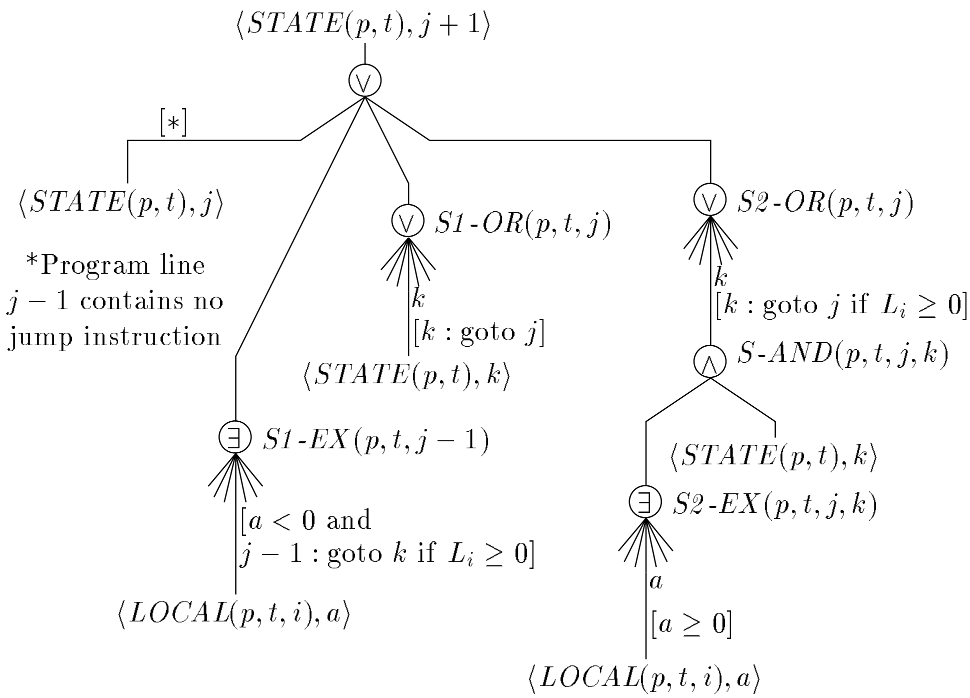

Figure 9: Construction of $\langle STATE(p,t), j+1\rangle$

Of course, neither SELECT-gates nor NC^0-oracles are of any use for circuits that already may have unbounded fan-out.

Proposition 13 $AC^k = AC^k(SEL) = AC^k(NC^0)$,
$UnambAC^k = UnambAC^k(SEL) = UnambAC^k(NC^0)$.

Corollary 14 $UnambAC^k_b(SEL) = \text{CREW}^k$.

Proof. This corollary immediately follows from Theorem 3 and Proposition 13. An alternative proof arises by using the simulations of Lemma 8 and 9 with the slight modification that the restriction $[p = \textit{read-owner}(b)]$ is dropped in the construction of $\langle READ(p,t,i), a\rangle$ (see Figure 5). □

Though we just considered the class $AC^k(SEL)$, it is not clear what a SELECT-gate computes in a circuit that is not unambiguous. In such a circuit it is allowed that SELECT-gates receive an arbitrary high number of ones. Nevertheless, we claim that only one output becomes one. In contrast to the unambiguous case, where this was guaranteed by the fact that each input group receives at most one 1, it is more difficult to decide which output becomes one. This is a similar difficulty as concurrent writing in PRAMs. We adopt the three major models of solution of write conflicts in CRCW-PRAMs for SELECT-gates: PRIORITY, ARBITRARY, and COMMON. In the case of

$AC_b^k(\mathit{PRIORITY\text{-}SEL})$ the output of some SELECT-gate is determined as follows: Output (i, j) becomes one if and only if i is the lowest numbered left hand input that receives a one and j is the lowest numbered right hand input that receives a one, or, with other words, the lexicographical smallest (i, j) wins.. In the case of $AC_b^k(\mathit{ARBITRARY\text{-}SEL})$-circuits, some arbitrary output (i, j) becomes one for which holds that the i^{th} left and j^{th} right inputs of an SELECT-gate receive ones. As for ARBITRARY CRCW-PRAMs the acceptance or rejection of all inputs w must not depend on the selection of the output of any SELECT-gate. Finally, for $AC_b^k(\mathit{COMMON\text{-}SEL})$-circuits we claim the same restriction for SELECT-gates as in $\mathit{UnambAC}_b^k(\mathit{SEL})$, i.e., at most one left and one right input must receive a one. This means that $AC_b^k(\mathit{ARBITRARY\text{-}SEL})$- and $\mathit{UnambAC}_b^k(\mathit{SEL})$-circuits differ only for AND- and OR-gates. While in $\mathit{UnambAC}_b^k(\mathit{SEL})$-circuits all gates are required to be vulnerable, in $AC_b^k(\mathit{COMMON\text{-}SEL})$-circuits this restriction only applies for SELECT-gates.

If we consider CRCW-PRAMs with polynomially many processors and $O(\log^k n)$ time bound, the three models PRIORITY, COMMON, and ARBITRARY accept the same classes of languages. However, this does not seem to be the case for ERCW-PRAMs, since both concurrent reads and concurrent writes play a crucial role in the simulations between these three models (see [KR90]). Also, the three different models do not seem to coincide for $AC_b^1(\mathit{SEL})$-circuits and indeed we can prove equalities between the corresponding circuit and PRAM classes:

Theorem 15

$$\begin{aligned} AC_b^k(\mathit{PRIORITY\text{-}SEL}) &= \mathit{PRIORITY}\text{-ERCW}^k \\ AC_b^k(\mathit{ARBITRARY\text{-}SEL}) &= \mathit{ARBITRARY}\text{-ERCW}^k \\ AC_b^k(\mathit{COMMON\text{-}SEL}) &= \mathit{COMMON}\text{-ERCW}^k \end{aligned}$$

Proof. "$\subseteq$" We can use the same method as it was used to prove Lemma 8. AND-, OR-, and NOT-gates can be simulated using COMMON-ERCW PRAMs in all cases, since all processors involved in the simulation of AND-gates only zeros are written.

A careful inspection shows that we can also adopt the simulation of SELECT-gates without changes. In the PRIORITY case we just have to guarantee that the priority of processors l_i and r_j grows monotonically (see Figure 3).

"$\supseteq$" We will deal separately with the three cases PRIORITY, ARBITRARY, and COMMON. If we use the construction of Lemma 9 to simulate a COMMON-ERCW PRAM, $\langle \mathit{WRITE}(t, a), b\rangle$ are the only gates the resulting circuit is clearly a $AC_b^k(\mathit{COMMON\text{-}SEL})$ circuit. The case of ARBITRARY-ERCW PRAM is more complicated. Now $\mathit{G\text{-}EX}(t, a)$ is not vulnerable, since different values c can be simultaneously written into global memory cell a (see Figure 4). Again this does not matter because OR-gates are not required to be vulnerable. A greater problem is that the construction does not work correctly any more. If two different values b and c are written into memory cell a in step t, both gates $\langle \mathit{GLOBAL}(t+1, a), b\rangle$ and $\langle \mathit{GLOBAL}(t+1, a), c\rangle$ evaluate to one. This is not correct since memory cell a can contain only one value. We can solve this problem by using a small modification of the construction of $\langle \mathit{WRITE}(t, a), b\rangle$. If

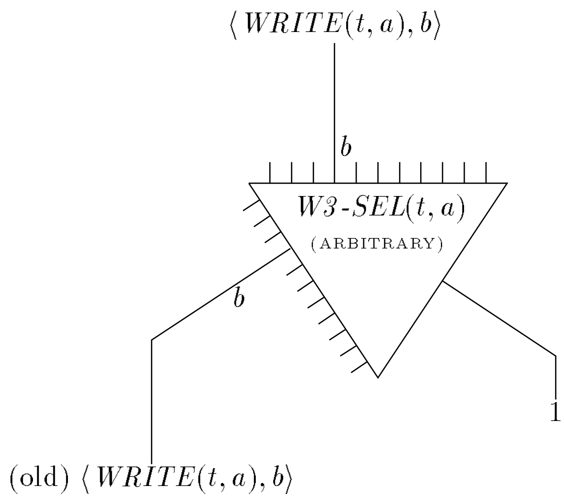

Figure 10: Revised construction of $\langle \mathit{WRITE}(t,a), b\rangle$ for ARBITRARY-ERCW PRAMs

multiple values b are written into memory cell a, $\mathit{W3\text{-}SEL}(t,a)$ selects arbitrarily one of them.

In the case of PRIORITY-ERCW PRAMs this new construction does not help, since we could only select the smallest value written into cell a, but we have to select that value written by the processor with the smallest number. Again this can be achieved by a revised construction of $\langle \mathit{WRITE}(t,a), b\rangle$. Gate $\mathit{W\text{-}OR}(p,t,a,b)$ computes whether processor p tries to write value b into global memory cell a in step t (see Figure 6). We use an additional SELECT-gate $\mathit{W3\text{-}SEL}(t,a)$ to select for fixed t and a the pair (p,b) with following properties:

1. $\mathit{W\text{-}OR}(p,t,a,b)$ evaluates to one, i.e., processor p actually tries to write value b,
2. (p,t) is the pair with minimal p that fulfills claim 1.

The resulting revised construction of $\langle \mathit{WRITE}(t,a), b\rangle$ is illustrated in Figure 11. □

All circuit constructions considered by now contained SELECT-gates where only one input of the right inputs was used. In the proof of Lemma 9 general SELECT-gates, that need all their inputs, were used only at two places: In the construction of $\langle \mathit{WRITE}(t,a), b\rangle$ (cf. Figure 6) and in the construction of $\langle \mathit{ADD}(p,t,i), a\rangle$ (cf. Figure 7). In this context the question arises whether simple SELECT-gates (i.e., SELECT-gates that have only one input at their right side) are sufficient for the simulation of Lemma 9, or if general SELECT-gates are necessary. The next theorem shows that general SELECT-gates are indeed necessary for the simulation, unless $\mathrm{EREW}^1 = \mathit{LOGSPACE}$. We denote the class of languages recognized by AC_b^k-circuits that contain additionally only such simple SELECT-gates by $AC_b^k(\mathit{SEL}_1)$.

Theorem 16 *$AC_b^1(\mathit{SEL}_1) \subseteq \mathit{LOGSPACE}$.*

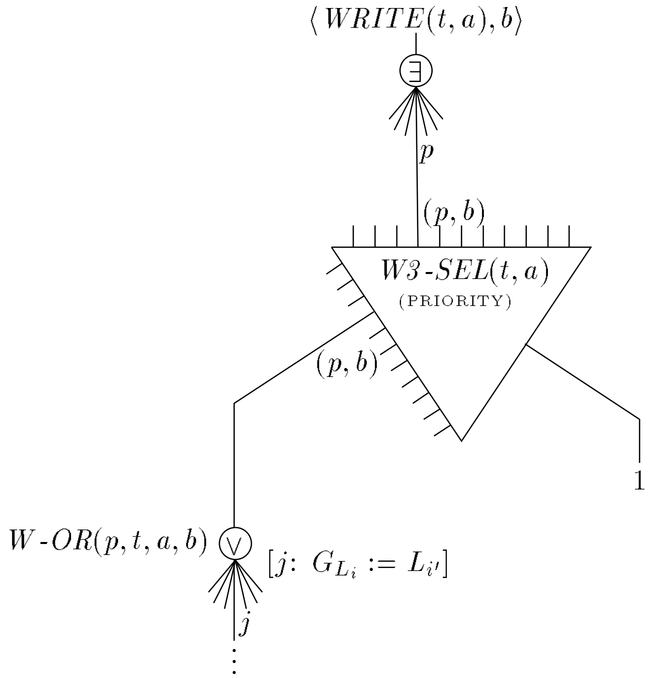

Figure 11: Revised construction of $\langle WRITE(t,a),b\rangle$ for PRIORITY-ERCW PRAMs

Proof. We perform again a depth-first traversal through the circuit. In contrast to the proof of Lemma 6, it is not possible to describe paths in the circuit in the same way. However, it *it* possible if the path does not contain an edge that leads to the right hand input of some SEL_1-gate. Again the description consists of some gate g (including one output number case of SEL_1-gates) and a string $w \in \{L,R\}^*$. The path begins at gate g and is continued by choosing the left or right parent of the current node according to the symbols in w. For instance, if some edge in the path leads to the i^{th} left hand input of some SEL_1-gate h and the corresponding symbol in w is L, then the next edge is the left edge of i^{th} output of h.

By now we have no description of paths that lead through right hand inputs of SEL_1-gates. Such a description does not seem to be possible with $O(\log n)$ space. However, it is sufficient to describe only a subset of all possible paths that contain right inputs of SEL_1-gates: If some edge on the path leads to the right hand input of some SEL_1-gate h, the next edge on the path is the left or right edge (according to w) of the i^{th} output of h, where i is the (uniquely determined) output of h that evaluates to one. An example of such a description is contained in Figure 12. Note that these descriptions need only $O(\log n)$ space.

A log space machine can do a special traversal through $AC^1_b(SEL_1)$-circuits which has two important properties:

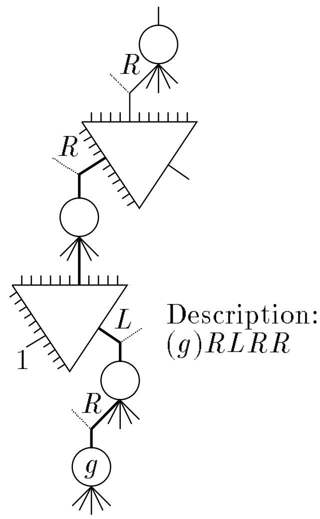

Figure 12:

1. The traversal starts from the root and goes along paths for which short descriptions exist,

2. During the traversal it can be computed whether the circuit accepts.

The traversal is recursively defined: The value of AND-, OR-, and NOT-gates is determined by computing recursively the values of as many predecessors as necessary. The value of the $i^{\rm th}$ output of a SEL_1-gate h is computed as follows. First, the value of the $i^{\rm th}$ output of h is computed. If this value is zero, then the $i^{\rm th}$ output of h is also zero.

If, however, the $i^{\rm th}$ input of h is one, then the value x of the right input of h is recursively computed. Once the value x is known, we know the value of the $i^{\rm th}$ output, too, since it is also x. However, since there is not enough space to store the number i, we have to recompute the number i by checking all left hand inputs of h until we find that the $i^{\rm th}$ left input receives a one. While doing this, the log space machine is able to remember x, since this requires only constant space. $\Box$

Acknowledgment We like to thank Birgit Jenner and Klaus-Jörn Lange for many fruitful discussions.

References

[CDHR89] B. S. Chlebus, K. Diks, T. Hagerup, and T. Radzig. New simulations between CRCW PRAMs. In *Fundamentals of Computation Theory*, pages 95–105. Springer, August 1989.

[Coo85] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[DR86] P. Dymond and W. L. Ruzzo. Parallel RAMs with owned global memory and deterministic language recognition. In *Proc. of 13th International Colloquium on Automata, Languages and Programming*, number 226 in Lecture Notes in Computer Science, pages 95–104. Springer, 1986.

[EG88] D. Eppstein and Z. Galil. Parallel algorithmic techniques for combinatorial computation. *Ann. Rev. Comput. Sci.*, 3:233–283, 1988.

[FRW84] F. E. Fich, P. Ragde, and A. Wigderson. Relationships between concurrent write models of parallel computation. In *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 179–189, 1984.

[FW78] S. Fortune and J. Willie. Parallelism in random access machines. In *Proc. of 10th ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[Gol82] L. M. Goldschlager. A universal interconnection pattern for parallel computers. *Journal of the ACM*, 29(3):1073–1086, 1982.

[GR89] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms*. Cambridge University Press, 1989.

[Hag90] T. Hagerup. Optimal parallel algorithms on planar graphs. *Information and Computation*, 84:71–96, 1990.

[Has86] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proc. of 18th ACM Symposium on Theory of Computing*, pages 6–20, May 1986.

[HKP84] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding fan-out in logical networks. *Journal of the ACM*, 31:13–18, 1984.

[KR90] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 17, pages 869–932. Elsevier, 1990.

[Lan90] K.-J. Lange. Unambiguity of circuits. In *Proc. of 5th Structure in Complexity Conf.*, pages 130–137, 1990.

[LR90] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in LNCS, pages 399–406. Springer, 1990.

[Par87] I. Parberry. *Parallel Complexity Theory*. Pitman, 1987.

[Ros91] P. Rossmanith. The owner concept for PRAMs. In *Proc. of 8th Symposium on Theoretical Aspects of Computer Science*, number 480 in Lecture Notes in Computer Science, pages 172–183. Springer, 1991.

[Ruz80] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235, 1980.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[Sch80] J. T. Schwarz. Ultracomputers. *ACM TOPLAS*, 2(4):484–521, 1980.

[Sni82] M. Snir. On parallel searching. In *Proc. SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 242–253, Ottawa, Canada, 1982.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

[SV84] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, May 1984.

[Vis83] U. Vishkin. Implementation of simultaneous memory address access in models that forbid it. *Journal of Algorithms*, 4:45–50, 1983.