Large Parallel Machines can be Extremely Slow for Small Problems

Vince Grolmusz

Department of Computer Science

Eötvös University, Budapest

Keywords: parallel computation, lower bounds, fault tolerant computation

ABSTRACT

We consider concurrent-write PRAMs with a large number of processors of unlimited computational power and an infinite shared memory. Our adversary chooses a small number of our processors and gives them a 0-1 input sequence (each chosen processor gets a bit, and each bit is given to one processor). The chosen processors are required to compute the *PARITY* of their input, while the others do not take part in the computation. If *at most* q processors are chosen and $q \leq \frac{1}{2}\log\log n$ then we show that computing *PARITY* needs q steps in the worst case. On the other hand, there exists an algorithm which computes PARITY in q steps (for any $q \leq n$) in this model, thus our result is sharp. Surprisingly, if our adversary chooses *exactly* q of our processors, then they can compute PARITY in $[q/2]+2$ steps, and in this case we show that it needs at least $[q/2]$ steps. Our result implies that one cannot construct large parallel machines which are efficient when only a small number of their processors are active. On the other hand, a result of Ajtai and Ben-Or [1] shows that if we have n input bits which contain at most polylog n 1's and polynomially many processors which are all allowed to work, then the *PARITY* can be solved in *constant* time.

Current affiliation: Mathematical Institute of the Hungarian Academy of Sciences; Reáltanoda u. 13-15, H-1053 BUDAPEST HUNGARY

1. INTRODUCTION

The PRIORITY concurrent-read concurrent-write parallel random access machine (PRIORITY PRAM) [9] has been proved to be a useful and a strong model of the parallel computation ([3] [13] [14] [10]). In this model n processors $P_1, P_2, ..., P_n$ are allowed synchronous read/write access to the cells $M_1, M_2, M_3, ...$ of an infinite shared memory. Each step of the computation consists of three phases. In the *compute phase* each processor performs local computation. To get more powerful lower bounds we make no assumptions about the maximal size of the local memory of a processor, or about its instruction set: an arbitrary amount of local computation is allowed in one step. In the *write phase* each processor may write into one cell, and simultaneous writes are allowed. If several processors write simultaneously into the same cell then this cell will contain the value written by the processor of lowest index. In the *read phase* each processor may read from a cell; simultaneous reads from the same cell are permitted.

We remark that this is the strongest model among the commonly used PRAM models ([4], [13], [7], [10]).

It is shown in [5] that if the instruction set of a processor is restricted and the size of the shared memory is bounded by a polynomial in n then PRIORITY PRAMs and unbounded fan-in Boolean circuits are time-depth equivalent (allowing a polynomial blowup in the number of processors or gates). Thus the circuit lower bounds hold for these restricted PRAMs. One method for proving lower bounds for these unrestricted PRAMs uses a Ramsey–theoretic approach which allows simplifying the pattern of interprocessor communication which could otherwise be extremely complex (e.g. [13], [6], [14], [10]). Applying Ramsey–theorem restricts input to great numbers (i.e. for large numbers to be sorted in [13]), and cannot be used for binary inputs.

The method of Beame and Hastad [3] works for binary inputs and proves tight $\Omega(\log n / \log \log n)$ lower bound for computing PARITY of n bits on PRIORITY PRAMs with a polynomial number of processors. The proof is based on the technique of "probabilistic restrictions" ([8], [15], [11]).

It is natural to ask the question: what is the complexity of PARITY if among the n input bits at most q are allowed to be 1 ? Surprisingly, if q is polylogarithmic in n then

- by a result of Ajtai and Ben-Or [1] -there exists a PRAM with a polynomial number of processors, which solves the PARITY in constant time. The proof shows the existence of that PRAM using probabilistic methods. (In fact, in [1] the existence of a Boolean circuit is demonstrated, but this implies the existence of the PRAM by [5]). Thus intuitively a large machine can solve a small problem very fast when all of its processors are working on it.

But how fast can it compute the PARITY of q bits if only q of its processors are allowed to work ? In order to be more precise, we define

SACK OF PROCESSORS $S(n,q)$*:*
$S(n,q)$ is a PRIORITY PRAM with n processors $P_1, P_2, ..., P_n$ and an infinite shared memory with cells $M_1, M_2, M_3, ...$ An input for Sack of Processors $S(n,q)$ consists of $q' \leq q$ bits. We say that $S(n,q)$ solves a problem P in T steps if the following holds: arbitrarily chosen $q' \leq q$ processors from $P_1, P_2, ..P_n$ are given q' input bits (one chosen processor gets exactly one bit), and after T steps the correct answer appears in the output cell(s). The processors which are not chosen are not allowed to take part in the computation.
An EXACT SACK OF PROCESSORS $S_{=}(n,q)$ is defined as a Sack of Processors $S(n,q)$ where q'=q (i.e. exactly q processors are chosen every time).

The chosen processors are called live ones. Intuitively, we imagine $S(n,q)$ as a set of numbered processors, put into a sack. We are allowed to give the processors arbitrarily complex programs. Then our adversary chooses several processors from our sack, gives them an input, and the chosen processors are required to compute a function, while the others are obliged not to take part in the computation.

Note that no live processor knows the identity of the other live processors at the start of the computation.

The concept of *dead* and *live* processors first appeared in [7] (prisoner–type problems). This model is helpful in designing large PRAMs which are supposed to handle several tasks simultaneously without any interaction between the processors computing different

tasks; for example, suppose that the input bits of a PRAM are colored by three colors, red, blue and green. The PRAM is required to compute the PARITY of red, the MAJORITY of blue and the decimal form of green bits. Then the processors, with input bits of the same color can be considered as chosen processors from a sack.

The SACK of PROCESSORS model can also be useful in building certain types of fault-tolerant PRAMs (when some of the processors can completely fail while the others are still supposed to compute some function). In [7] an algorithm in a similar model is used for simulation of the PRIORITY PRAM by COMMON PRAM, with an equal number of processors (in COMMON PRAM [12], several processors may attempt to write into the same cell if only they write the same word – otherwise the entire PRAM stops working).

Intuitively, some problems may be more difficult in this model than in the usual PRAM model, since the live processors *a priori* do not know whom they can work with. If every live processor knew the identity of the other live processors, then the SACK of PROCESSORS model is obviously equivalent with q–processor PRIORITY PRAMs, this implies that computing any function would take only $O(\log q)$ steps [2].

It is easy to see that for any n and q there exists a SACK of PROCESSORS which computes fan-in q' OR in one step. If positive integers can be written into the input cells, then the Element Distinctness problem ([6], [2]) can also be solved in a constant number of steps in this model for arbitrary n and q, while the COMMON PRAM needs non-constant time to solve it even when all the processors are allowed to work [14]. If our processors in sack $S(n,q)$ are only allowed to use the COMMON write-conflict resolution scheme then computing PARITY needs $\Omega(\log n/\log\log n)$ steps, even for $q=2$ [7].

We shall see that the following two problems are difficult in the (PRIORITY) SACK of PROCESSORS model. The first is the well known PARITY function [8] with at most q input bits, and the other is the

RIGHTMOST ONE problem:

input: $b_{i_1}, b_{i_2}, ..., b_{i'_q}$; $q' \leq q$; b_{i_k} *is the bit given to processor* P_{i_k}; *output:* i_k, *iff* $b_{i_{k+1}}, ..., b_{i_{q'}} = 0$, *and* $b_{i_k} = 1$.

The output is required to be written into the first cell of the shared memory. We would like to obtain sharp results without even additive constants, thus the processors are allowed to write some redundant information into the first cell as well, but the output must be well identified.

We note, that computing LEFTMOST ONE needs only 1 step.

The following simple CONFERENCE ALGORITHM solves RIGHTMOST ONE and PARITY in at most q steps for $S(n,q)$ sacks:

CONFERENCE algorithm:

- *Before the start of the algorithm each live processor is in a special state, called "write". Then*

($\star$) *Every live processor in "write" state attempts to write its name (number), its input-bit and a word (which depends on the problem to be solved) into M_1. Then every live processor reads M_1. The processor in "write" state which has read its own name in M_1 ceases to be in "write" state.*

- *repeat ($\star$).*

The processors read and compute the sum of the input bits previously appeared in M_1, and add to this sum their own input bit, modulo 2. This sum is the extra "word", which the processors attempt to write into M_1, when the function to be computed is the PARITY. In the RIGHTMOST ONE problem processors need not write anything but their signature. Then after at most q steps the correct answer will be written into the result cell.

After at most q steps of Conference Algorithm each live processor knows the complete input: if $q' < q$, then after step q' each processor knows the whole input, but they do not know that they know, so one more step is needed, when the content of cell M_1 will not be changed. When $q' = q$ then this additional step is not needed. Thus – since the local computation is costless – they can compute an arbitrary function in the *compute phase* of step $q+1$ and write the result into cell M_1. Thus for every function with at most q variables there exists an $S(n,q)$ which computes it in $q+1$ steps.

At the first glance, this algorithm seems to be extremely bad (it uses only one cell

for the interprocessor communication). However, we shall prove that it is optimal for our problems if q is small enough. Our main result is

THEOREM 1. *Let $S(n,q)$ be a SACK of PROCESSORS which computes PARITY in T steps. If $q < \frac{1}{2}\log\log n$ then $T \geq q$.*

The PRAMs with costless local computation can compute any Boolean function in $O(\log n)$ steps, simply by gathering the input-bits into the local memory of a processor [2]. This also works for SACKS of PROCESSORS: for any function there exists an $S(n,q)$ which computes the function in $O(\log n)$ steps, for any n and $q \leq n$.

Our Theorem 1 holds also for the RIGHTMOST ONE problem (Corollary 1). An algorithm of Chlebus, Diks, Hagerup and Radzik [16] shows that Corollary 1 does not hold for larger q's: their algorithm works in $O(\log\log n)$ steps for all $q \leq n$.

Obviously, the EXACT SACKS of PROCESSORS are not weaker than the SACKS of PROCESSORS. We show in Section 3 that they can compute PARITY (and any other symmetric Boolean function) in $\lceil q/2 \rceil + 2$ steps, and we shall prove there that this is almost optimal:

THEOREM 2. *Let $S_{=}(n,q)$ be an EXACT SACK of PROCESSORS which computes PARITY in T steps. Suppose, that $q \leq \frac{1}{3}\log\log n$. Then $T \geq \lceil q/2 \rceil$.*

We remark, that PARITY can be computed for all $q \leq n$ by SACK of PROCESSORS $S(n,q)$ in expected time $O(\log q)$, when live processors are chosen in uniform distribution [17]. This shows an exponential gap between the expected and the worst case running time of SACK of PROCESSORS– algorithms.

2. PROOF OF THEOREM 1

Suppose that n and q satisfy the conditions of Theorem 1, and let $S(n,q)$ be a SACK of PROCESSORS which computes PARITY in T steps. We show that $T \geq q$ even when all the chosen processors get only "1" bits. The proof is an adversary argument. The strategy will be described with help of partitions of the set of processors. Let R denote the set of the processors in sack $S(n,q)$: $R = \{P_1, P_2, ..., P_n\}$. For $t = 0, 1, ..., q-1$ we shall define partitions of R into 3 sets A_t, B_t, C_t. Let $A_0 = B_0 = \emptyset$, $C_0 = R$. A_{t+1}, B_{t+1}, and C_{t+1} will satisfy $A_t \subset A_{t+1}, B_t \subset B_{t+1}, C_{t+1} \subset C_t$ and $\mid A_t \mid \leq t$. Our adversary chooses $q' \leq q$ processors from R and gives each of them an input-bit. The set of the chosen processors $H \subset R$ is called a *t-vicious choice* if

- $A_t \subset H \subset A_t \cup C_t$,
- *each processor in H gets input-bit "1".*

By the last property, a t-vicious choice describes not only the set of processors which are allowed to work, but the inputs given to them. The possible t-vicious choices are determined by sets A_t, B_t, C_t. Obviously, any (t+1)-vicious choice is also a t-vicious choice. We show that there exists a partition of R into sets $A_{q-1}, B_{q-1}, C_{q-1}$ and a (q-1)-vicious choice such that the result-cell will not contain the correct answer after step q-1, thus $S(n,q)$ needs at least q steps to compute PARITY.

Let P be an element of C_t. We say that processor P *t-affects* a processor $Q \neq P$, $Q \in A_t \cup C_t$, if there exist t-vicious choices H and H' such that $H = H' \cup \{P\}$, $Q \in H'$, and for choices H and H' the state of processor Q is different after step t.

Similarly, we say that processor P *t-affects* cell c if there exist t-vicious choices H and H' such that $H = H' \cup \{P\}$ and for choices H and H' the contents of c differ after step t.

The following two conditions are required to hold for A_t, B_t, C_t for $t = 0, 1, ..., q-1$.

Condition 1. *There exist no processors $P, Q \in A_t \cup C_t$, such that P t-affects Q.*

Condition 2. *Every cell is t-affected by at most one element of C_t.*

Suppose A_{q-1}, B_{q-1} and C_{q-1} satisfy these conditions and $\mid C_{q-1} \mid \geq 2$. Then Condition 2 implies that the result-cell M_1 is also (q-1)-affected by at most one element of C_{q-1}, say P'. Let $P'' \in C_{q-1}$ such that $P'' \neq P'$. Then the choices A_{q-1} and $A_{q-1} \cup \{P''\}$ are (q-1)-vicious, and since P'' does not (q-1)-affect result-cell M_1 the contents of M_1 is the same for choices A_{q-1} and $A_{q-1} \cup \{P''\}$, but the parity of the inputs is different. Thus $S(n,q)$ cannot compute PARITY in $q-1$ steps.

Thus it is enough to find $A_{q-1}, B_{q-1}, C_{q-1}, \mid C_{q-1} \mid \geq 2$ satisfying Conditions 1 and 2. We give a recursive construction. For $A_0 = B_0 = \emptyset$, $C_0 = R$ Conditions 1 and 2 hold. Suppose that for some $t \leq q-2$, for all $t' \leq t$ $A_{t'}, B_{t'}$, and $C_{t'}$ satisfy Conditions 1 and 2. We shall define $A_{t+1}, B_{t+1}, C_{t+1}$. Let $k_t = \mid C_t \mid$. Condition 1 implies that for all t-vicious choice $H, P \in H$, the state of processor P after step t is independent of the special choice of H. We define an equivalence relation $\sim$ on C_t. For $P, Q \in C_t$ $P \sim Q$ if P and Q attempt to write into the same cell "s" in step $t+1$. Equivalence relation $\sim$ determines equivalence classes on C_t. Either there exists a class of size at least $\sqrt{k_t}$ or there are more than $\sqrt{k_t}$ classes.

In the first case let us consider a class C'_{t+1} of size at least $\sqrt{k_t}$. Let W be the processor of smallest index (i.e. the highest priority) in C'_{t+1}. Suppose that W attempts to write into cell c in step $t+1$. If our adversary in the t - vicious choices also choose processor W then the word written into cell c is fixed, since either W succeeds to write into c or because one of the elements of A_t with priority higher than that of W writes there. Let $C''_{t+1} = C'_{t+1} - \{W\}$, and $A_{t+1} = A_t \cup \{W\}$.

In the second case, when there are more than $\sqrt{k_t}$ equivalence classes, let C''_{t+1} be an arbitrary set which contains exactly one common element with each equivalence class, and let $A_{t+1} = A_t$.

In order to get C_{t+1} we define a directed graph G_{t+1} with vertex set $A_{t+1} \cup C''_{t+1}$. (Q, P) is an edge of G_{t+1} if for some cell c processor Q attempts to write into or reads from cell c in step $t+1$, where for some $H \subset C''_{t+1}$, $P \notin H$, c has different contents after step $t+1$ for t-vicious choices $A_{t+1} \cup H \cup \{P\}$ and $A_{t+1} \cup H$.

If cell c has different contents for choices $A_{t+1} \cup H \cup \{P\}$ and $A_{t+1} \cup H$ after step $t+1$, then either P t-affects cell c or P writes into cell c in step $t+1$. By Condition 2 every cell is t-affected by at most one processor, and by the construction of C''_{t+1} at most one additional processor may write into a cell of the shared memory in step $t+1$, thus each processor in $A_{t+1} \cup C''_{t+1}$ has out-degree at most 4 in the graph G_{t+1}. Observe, that processors in A_{t+1} have in-degree 0 in graph G_{t+1}. Let $N(A_{t+1})$ denote the set of neighbors of elements of A_{t+1}. Let us denote $C'''_{t+1} = C''_{t+1} - N(A_{t+1})$. Clearly, $|C'''_{t+1}| \geq \sqrt{k_t} - 4(t+1) - 1$. Now we would like to find a large subset C_{t+1} in C'''_{t+1}, which is independent in graph G_{t+1}. Each vertex in C'''_{t+1} has out-degree at most 4, so, at least half of the vertices in C'''_{t+1} have at most 8 connecting edges. So, if G_{t+1}, restricted to this half, is denoted by G'_{t+1} then G'_{t+1} can be colored with 9 colors. This implies that there exists in C'''_{t+1} an independent vertex set C_{t+1} satisfying

$$|C_{t+1}| \geq \frac{1}{18}|C'''_{t+1}| \geq \frac{1}{18}\left(\sqrt{k_t} - 4(t+1) - 1\right). \tag{1}$$

Obviously, $C_{t+1} \cup A_{t+1}$ is independent in G_{t+1}.

We state that for A_{t+1}, B_{t+1} and C_{t+1} Conditions 1 and 2 hold. If P (t+1)-affects but does not t-affect Q then Q reads a cell which is (t+1)-affected by P in step t+1. In this case (Q, P) is an edge of G_{t+1}, and this is a contradiction, since P and Q are elements of an independent set of vertices of G_{t+1}. Suppose that cell c is (t+1)-affected, but not t-affected by two elements, P and Q, of C_{t+1}. Then either (P, Q) or (Q, P) or both are edges of G_{t+1}, and because of this contradiction Condition 2 holds.
Because of inequality (1), we cannot get stuck if $t \leq \frac{1}{2}\log\log n$, thus Theorem 1 is proved.
■

COROLLARY 1. *Let $S(n, q)$ be a SACK of PROCESSORS which computes RIGHT-MOST ONE in step T for $q \leq \frac{1}{2}\log\log n$. Then $T \geq q$.*

Proof. The proof of the previous theorem works for this corollary, too. Note that the largest index of the elements of A_t is less than the smallest index of the processors of

C_t. Since C_{q-1} contains at least 2 processors, $q-1$ steps is not enough to compute the RIGHTMOST ONE live processor. ∎

3. FURTHER RESULTS

In this section we examine the EXACT SACKS of PROCESSORS. Intuitively, knowing that our adversary will choose exactly q of our processors may help in designing algorithm for SACKS of PROCESSORS. Our results show that knowing the number of processors to be chosen halves the time which is necessary and sufficient to compute PARITY for small enough q's. The following ES (EXACT SACK) algorithm computes PARITY (and any other symmetric Boolean function) in $[q/2]+2$ steps.

ES algorithm:

- *Before the start of the computation each live processor with input bit i is in state i-write, for $i = 0, 1$.*

$(*)$ *Every live processor in i-write state attempts to write its name into cell M_{i+1}. Then every processor with input-bit i reads cell M_{i+1}. The processors in i-write state, which have read their own name in M_{i+1}, cease to be in i-write state.*

- *Each live processor repeats $(*)$ until in two consecutive steps the contents of M_1 or M_2 does not get changed.*

If processor P has input-bit i and the contents of M_{i+1} remains unchanged in step $t+1$ then all the processors with input-bit i have written their name into M_{i+1}, thus P has read the names of all the processors, which have input i. So P can compute the parity of the input since it knows the number of the $1-i$ bits, and writes it into the result-cell.

Clearly, in $[q/2]+1$ steps a repetition will occur either in M_1 or in M_2, since either the number of 1's or the number of 0's is less or equal to $[q/2]$. One more step is needed to write the correct answer into the result-cell. Thus in $[q/2]+2$ steps PARITY (and any other symmetric Boolean function) can be computed with the ES algorithm on an EXACT

SACK of PROCESSORS.

However, knowing that the adversary will choose exactly q processors does not help in finding the RIGHTMOST ONE 1-bit.

COROLLARY 2. *Any EXACT SACK of PROCESSORS $S_=(n,q)$, where $q \leq \frac{1}{2}\log\log n$, needs at least q steps to solve the RIGHTMOST ONE problem.*

PROOF. The proof of Corollary 1 can easily be modified such that the t-vicious choices have size q. In C_{q-1} at least $q - |A_{q-1}| + 1$ processors are needed. ∎

Proof of Theorem 2. The proof is similar to the proof of Theorem 1, but it is a little bit more complicated, and some definitions should be changed.

Let $S_=(n,q)$, $q \leq \frac{1}{3}\log\log n$, be an EXACT SACK of PROCESSORS which solves PARITY in T steps, Let R denote the set of n processors of $S_=(n,q)$. For $t = 0, 1, 2, ..., [q/2]-1$ we define partitions of R into 4 sets A_t^0, A_t^1, B_t, C_t. The following relations will be satisfied: $A_t^0 \subset A_{t+1}^0,\ A_t^1 \subset A_{t+1}^1,\ B_t \subset B_{t+1}, C_{t+1} \subset C_t,\ |A_t^1| \leq t,\ |A_t^0| \leq t$. The adversary makes a *t-vicious choice* $H \subset R$ if

- $A_t^0 \cup A_t^1 \subset H \subset A_t^0 \cup A_t^1 \cup C_t$,
- $|H| = q$

A *legal input* α for t-vicious choice H is a sequence of input bits given to processors of H such that processors of A_t^0 get bit 0, processors of A_t^1 get 1, and processors of $H \cap C_t$ may get any bit.

We say that processor $P \in C_t$ *t-affects* processor $Q \in A_t^0 \cup A_t^1 \cup C_t$, $Q \neq P$ if there exists a t-vicious choice H such that $P, Q \in H$, and there exist

- either legal inputs α and β such that α and β differ only in bit given to processor P, and the state of Q for inputs α and β is different after step t;
- or legal input α such that the state of Q after step t is different in case of choice H and input α and choice $H - \{P\}$ and input α', where α' is defined such that the input of processors of $H - \{P\}$ is the same as in α.

Remark. The choice $H - \{P\}$ is illegal in the previous definition. Our extremely vicious

adversary may make illegal choices, but in this case our EXACT SACK of PROCESSORS $S_{=}(n,q)$ is not required to perform a correct computation. But, if Q has different states in step $t+1$, we may say that P t-affects Q.

Similarly, we say that processor $P \in C_t$ *t-affects* cell c of the shared memory if there exists t-vicious choice H, where $P \in H$ and there exist

- either legal inputs α and β such that α and β differ only in bit given to processor P, and the contents of c is different for inputs α and β after step t;
- or legal input α such that the contents of c after step t is different in case of choice H, input α and in case of choice $H - \{P\}$ and input α', where in α' the input of processors of $H - \{P\}$ is the same as in α.

Conditions 1 and 2 are required to hold for A_t^0, A_t^1, B_t, C_t where $t = 0, 1, 2, ..., [q/2] - 1$.

Condition 1. *There exist no processors P and Q such that P t-affects Q;*

Condition 2. *Every cell is t-affected by at most one element of C_t.*

Suppose $A^0_{[q/2]-1}, A^1_{[q/2]-1}, B_{[q/2]-1}, C_{[q/2]-1}$ satisfy these conditions and $\mid C_{[q/2]-1} \mid \geq q$. Let $P, Q \in C_{[q/2]-1}$, $P \neq Q$, and let H be a ([q/2]-1)-vicious choice such that $P, Q \in H$. By Condition 2, result cell M_1 is $([q/2]-1)$-affected by at most one processor, say Q. Then for legal inputs α and β, where α and β differ only in bit given to processor P, the contents of M_1 will be the same, although the parity of α and β is different. This implies that $S_{=}(n,q)$ needs at least $[q/2]$ steps to compute PARITY.

We give a recursive construction for A_t^0, A_t^1, B_t, C_t. Clearly, $A_0^1 = A_0^0 = B_0 = \emptyset$, $C_0 = R$ satisfy the requirements. Suppose that A_t^0, A_t^1, B_t and C_t satisfy Conditions 1 and 2, and $\mid C_t \mid = k_t$. Then for any t-vicious choice the state of each processor after finishing step t depends only on its input bit. Let $\cong$ be an equivalence relation on C_t such that for $P, Q \in C_t$, $P \cong Q$ if the input bit of P and Q is 1 then P and Q attempt to write into the same cell of the memory in step $t+1$. There exists either an equivalence class of at least $\sqrt{k_t}$ elements, or there are more than $\sqrt{k_t}$ classes.

In the first case let C^*_{t+1} be a class with at least $\sqrt{k_t}$ elements and let W be the processor of the smallest index in C^*_{t+1}. Let $C^{**}_{t+1} = C^*_{t+1} - \{W\}$, and $A^1_{t+1} = A^1_t \cup \{W\}$.

In the second case let C^{**}_{t+1} be an arbitrary set which contains exactly one common element with every equivalence class, and let $A^1_{t+1} = A^1_t$.

Now we define a new equivalence relation $\approx$ on C^{**}_{t+1}. For $P, Q \in C^{**}_{t+1}$ $P \approx Q$ if the input of P and Q is 0 then they attempt to write into the same cell in step $t+1$. Either there exists an equivalence class C'_{t+1} of at least $k_t^{\frac{1}{4}} + 1$ processors, or there are at least $k_t^{\frac{1}{4}} - 1$ equivalence classes.

In the first case let X be the processor of the smallest index in C'_{t+1}. Let $A^0_{t+1} = A^0_t \cup \{X\}$ and $C''_{t+1} = C'_{t+1} - \{X\}$.

In the second case let C''_{t+1} be a set which contains exactly one common element with every equivalence classes, and let $A^0_{t+1} = A^0_t$.

The size of C''_{t+1} is at least $k_t^{\frac{1}{4}} - 1$ in both cases. In order to get C_{t+1} we define a directed graph G_{t+1} on vertex set $V_{t+1} = A^0_{t+1} \cup A^1_{t+1} \cup C''_{t+1}$. For $P, Q \in V_{t+1} P \neq Q$ (Q, P) is an edge of G_{t+1} if there exist

- *a t-vicious choice $H = A^0_{t+1} \cup A^1_{t+1} \cup H'$, where $H' \subset C''_{t+1}$;*
- *and inputs α and β for processors of H, such that the input of processors in A^j_{t+1} is j for $j = 0, 1$, and α and β differ only in bit given to P;*
- *and a cell c, such that for choice H and input α Q attempts to write into or reads from cell c in step $t+1$, where c has different contents in step $t+1$*
  - *a) either for inputs α and β,*
  - *b) or for choice H with input α and choice $H - \{P\}$ with input α', where in α' each processor of $H - \{P\}$ gets the same input as in α.*

By Condition 1, for any t-vicious choice, each processor has two different states in the writing, and just before the reading phase of step $t+1$. These states depend only on the input bit of the processors. Thus each processor may attempt to write into at most two cells and may read from at most two cells of the shared memory in step $t+1$. Thus for

each Q there exists at most 4 different cells c in the definition of G_{t+1}. For any t-vicious choice, any cell is t-affected by at most one processor. Cell c in definition of G_{t+1} may have different contents if either P t-affects cell c, or for some inputs, P writes into c in step $t+1$. If we consider only choices $H = A^0_{t+1} \cup A^1_{t+1} \cup H'$, $H' \subset C''_{t+1}$, then for any legal inputs any cell may be written into by at most two additional processors. Thus in graph G_{t+1} the out-degree of each node is at most 12. We remark that the in-degree of the vertices of $A^0_{t+1} \cup A^1_{t+1}$ is 0.

Let $N(A^i_{t+1})$ denote the set of neighbors of elements of A^i_{t+1} in graph G_{t+1}, for $i = 0, 1$, respectively. Let us denote by $C'''_{t+1} = C''_{t+1} - (N(A^0_{t+1}) \cup N(A^1_{t+1}))$. Clearly, $|C'''_{t+1}| \geq k_t^{\frac{1}{4}} - 24(t+1) - 1$. Now we would like to find a large subset C_{t+1} in C'''_{t+1}, which is independent in graph G_{t+1}. Each vertex in C'''_{t+1} has out-degree at most 12, so, at least half of the vertices in C'''_{t+1} have at most 24 connecting edges. So, if G_{t+1} restricted to this half is denoted by G'_{t+1} then G'_{t+1} can be colored with 25 colors. This implies that in C'''_{t+1} there exists an independent vertex set C_{t+1} satisfying

$$|C_{t+1}| \geq \frac{1}{50}|C'''_{t+1}| \geq \frac{1}{50}\left(k_t^{\frac{1}{4}} - 24(t+1) - 1\right). \tag{2}$$

Let $B_{t+1} = R - (A^0_{t+1} \cup A^1_{t+1} \cup C_{t+1})$. This completes the definition of the four-partition of set R in step $t+1$.

We show now that Conditions 1 and 2 hold. It is easy to see that if processor P (t+1)-affects but does not t-affect processor Q then (Q, P) is an edge of G_{t+1}, and this is not possible when P and Q are elements of a (t+1)-vicious choice. This implies that Condition 1 holds. Similarly, if P and Q are elements of a (t+1)-vicious choice, then they cannot (t+1)-affect any cell c simultaneously, because in this case (P, Q) or (Q, P) were edges of G_{t+1}. Thus Condition 2 also holds. Because of (2), we cannot get stuck until $t \leq \frac{1}{3}\log\log n$. This completes the proof of Theorem 2. ∎

4. OPEN PROBLEMS

We have shown that for small q's the processors of a sack $S(n,q)$ or $S_{=}(n,q)$ cannot communicate effectively and because of this, computing some simple functions needs long time, relative to q. In [16] an algorithm is shown which solves the RIGHTMOST ONE problem for any $q \leq n$ in $O(\log\log n)$ time. However, this algorithm does not work for computing PARITY, in this case we have only the trivial $c \log n$ upper bound. It would be interesting to give a better SACK of PROCESSORS algorithm for PARITY.

An interesting question, related to our result, is the following: can the processors help each other, or, more precisely: given n processors and n colored input-bits. The parity of the bits of each color should be computed and written into the result cells of the appropriate color. There exist n/q colors and q bits of each color. There are n/q different result cells, one for each color. Our adversary gives the processors the colored input bits.

If the processors with input bit of color i have not access to cells, written by processors with input bit of color j, $i \neq j$, then computing PARITY of each color needs $[q/2]$ steps by our Theorem 2. But how fast can they compute PARITY if they have access to cells, written by processors of different color?

Acknowledgements

I thank *Miklós Simonovits* for many stimulating discussions of the material presented here, and *Prabhakar Ragde* for helpful comments.

REFERENCES

[1] Ajtai, M., Ben-Or, M.: A Theorem on Probabilistic Constant Depth Computation. Proc. 16^{th} Annual ACM Symposium on Theory of Computing, 1984, pp. 471-474.

[2] Beame, P.: Lower Bounds in Parallel Machine Computation, Ph. D. Thesis, University of Toronto, 1986.

[3] Beame, P., Hastad, J.: Optimal Bounds for Decision Problems on the CRCW PRAM. Proc. 19^{th} Annual ACM Symposium on Theory of Computing, 1987

[4] Cook, S.A., Dwork, C., Reischuk, R.: Upper and Lower Bounds for Parallel Random Access Machines Without Simultaneous Writes, SIAM J. Computing, vol 15, no. 1, 1986. pp. 87-97.

[5] Stockmeyer, L., Vishkin, U.: Simulation of Parallel Random Access Machines by Circuits, SIAM J. Comput. vol 14, No. 3. pp. 688-708

[6] Fich, F.E., Meyer auf der Heide, F. Wigderson, A.: One, Two, Three,... Infinity: Lower Bounds for Parallel Computation. Proc. 17^{th} Annual ACM Symposium on Theory of Computing, 1985, pp. 48-58.

[7] Fich, F.E., Ragde, P., Wigderson, A.: Simulations Among Concurrent - Write PRAMs, Algorithmica 3, No. 1, 1988, pp. 43-52.

[8] Furst, M., Saxe, J.B., Sipser, M.: Parity, Circuits, and the Polynomial Time Hierarchy. Math. Systems Theory, 17, pp. 13-27, 1984.

[9] Goldschlager, L.: A Unified Approach to Models of Synchronous Parallel Machines. J. ACM vol 29, No. 4, 1982, pp. 1073-1086.

[10] Grolmusz, V., Ragde, P.: Incomparability in Parallel Computation. Proc. 27^{th} Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 89-98.

[11] Hastad, J.: Almost Optimal Lower Bounds for Small Depth Circuits. Proc. 18^{th} Annual ACM Symposium on Theory of Computing, 1986, pp. 6-20.

[12] Kučera, L. Parallel Computation and Conflicts in Memory Access, Information Processing Letters, vol. 14, No. 2, 1982, pp. 93-96

[13] Meyer auf der Heide, F., Wigderson, A.: The Complexity of Parallel Sorting. SIAM Journal of Computing, vol. 16, pp. 100-108.

[14] Ragde, P., Steiger, W., Szemerédi, E., Wigderson, A.: The Parallel Complexity of Element Distinctness is $\Omega(\sqrt{\log n})$, SIAM Journal of Discrete Mathematics, vol 1, No. 3, pp. 399-410

[15] Yao, A.: Separating the Polynomial-Time Hierarchy by Oracles. Proc. 25^{th} Annual IEEE Symposium on Foundations of Computer Science, 1985, 1- 10.

[16] Chlebus, B.S., Diks, K., Hagerup, T., Radzik T.: Efficient Simulations between Concurrent-Write PRAM Models, Mathematical Foundations of Computer Science, 1988.

[17] Grolmusz, V.: On the Complexity of Parallel Algorithms, Ph. D. Thesis, Eötvös University–Hungarian Academy of Sciences, 1988 (in Hungarian).