An $\Omega(n \log^2 n / \log \log n)$ lower bound for Algorithm W in a synchronous fail-stop (no restart) PRAM

Alejandro López-Ortiz *

Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick E3B 4A3
Canada
e-mail: alopezo@unb.ca

Abstract

In [1] Buss et al. propose an algorithm for the *Write-All* problem for a general fail-stop PRAM. In the same paper it is conjectured that the algorithm performs work $\Omega(N \log N \log \log N)$ for the fail-stop with no restart model. In this work it is shown that, using the adversary proposed by Kanellakis and Shvartsman, the amount of work performed by such algorithm is $\Omega(n \log^2 n / \log \log n)$.

1 On the Model of Computation

This model assumes a fail-stop without restart synchronous PRAM. Since an asynchronous PRAM has as a particular behaviour synchronicity, the lower bound in the amount of work performed by this particular algorithm applies as well to an asynchronous model.

*Work done while at Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

2 Sketch of the Algorithm

The algorithm treats cells $1 \dots n$ as arranged in a heap. Processors are assigned to the leaves of the heap which write a 1 and proceed further up the tree. An internal node in the tree may only be written when both children have been initialized to 1. A processor present in an internal node where not both of the children have been initialized descends down the to the uninitialized child or to the one corresponding to its PID if both are uninitialized.

3 Description of the Adversary

The adversary being considered fail-stops $1/\log n$-th processors reaching the uninitialized leaves.

For the purposes of clarity of the argument, we assume that the adversary fails processors in the leftmost leaves of the tree, but this is irrelevant to the correctness of the algorithm.

That is to say, in the first round $n/\log n$ processes are made to fail, in the second round out of the $n(1 - 1/\log n)$ surviving processors $1/\log n$ of them are made to fail, namely those in the $n/\log^2 n$ leftmost leaves of the subtree of undone leaves. (The definition of a round is made formal in the next section)

The adversary withdraws when the number of processors to be failed is below one.

4 Work Performed

To compute the amount of work performed we look at steps within each round namely,

1. Processors are at the leaves, those not failed by the adversary write 1.

2. Functional processors move up the tree, writing ones until they reach a node for which not both children are initialized.

3. Processors go down the tree until they reach a leaf. at which point we are back in step 1.

Let us compute the amount of work performed in step 3 first.

Definition 1 *A k-round no-failures subtree is a perfect subtree of the heap, such that the leaves of the subtree are leaves of the heap and no process assigned to the subtree in round k was killed.*

Definition 2 *A maximal k-round no-failures subtree is the a k-round no-failures subtree which is not contained in any other k-round no-failures subtree.*

It follows from the definition of Algorithm W and the fact that the PRAM is synchronous that all processors belonging to a (maximal) 1st-round no-failures subtree reach the end of step 2 at the same time-step.

Furthermore, the time-step at which they finish step 2 is a constant times the height of the subtree. Which means that shallower subtrees near the failed region finish earlier than bigger subtrees at the rightmost part of the heap.

As an example, after the first round of failures there is exactly one subtree for each of the heights $\log n - 1, \log n - 2, \ldots, \log n - \log\log n$. After the second round the tree heights are $\log n - \log\log n - 1, \log n - \log\log n - 2, \ldots, \log n - 2\log\log n$.

At the end of step 2 processors start trickling down the unfinished failed subtree as indicated in step 3. Those processors which finished earlier (from shallow subtrees) will reach the leaves of the unfinished subtree before processors assigned to deeper trees. Processors which reach a leaf in step 3 and are not failed in step one will proceed upwards the tree in step two **before** other processors finish step three of the **previous** round. These late processors will not reach the leaves since they will encounter memory cells already initialized by the earlier processors. At this point late processors consider step 3 to be finished, skip step 1 of the next round and "catch up" with earlier processors in step 2.

Notice that in each round all processors spend the same amount of time ($\log n - k\log\log n$, for the k-th round) in step 3, although they may be at different times in such step.

This is due to the fact that processors take the same time (within a constant factor) to go downwards in step 3 as it takes to go upwards in step 2. (This may not be true for some variants of the algorithm, in which case the argument is modified to be: Processors spend as much time in step 2 and step 3 as the leftmost, earlier processors, do in step 2 alone).

Notice that the maximum delay is $O(\log\log n)$. Furthermore the earlier processors spend time $O(\log n - k\log\log n)$ in step 3. Implying that all late processors will catch up with the earlier processors **before** the end of step 2 in any given round.

The final point that needs to be made is: How many processors are functional at round k?

Since at each step the adversary fails $1/\log n$ of them we have the recurrence

$$\begin{aligned}
FP(1) &= n && (1)\\
FP(k) &= FP(k-1) - (1/\log n)FP(k-1) && (2)\\
&= \frac{\log n - 1}{\log n}FP(k-1). && (3)
\end{aligned}$$

With solution

$$FP(k) = n\left(\frac{\log n - 1}{\log n}\right)^k.$$

Therefore the amount of work performed is, at least,

$$\sum_{k=1}^{\frac{\log n}{\log\log n}} \underbrace{n\left(\frac{\log n - 1}{\log n}\right)^k}_{\text{number of processors in round k}} \underbrace{(\log n - k\log\log n)}_{\text{work in step 2 by leftmost processors}}$$

The closed form of this summation is not particularly enlightening. Instead we show that such summation is $\Theta(n\log^2 n/\log\log n)$

The key point is to notice that $\left(\frac{\log n-1}{\log n}\right)^{\frac{\log n}{\log\log n}}$ converges to 1 as n goes to infinity.

From this is an exercise of arithmetic to show that

$$1/2 \le \left(\frac{\log n - 1}{\log n}\right)^k \le 1$$

for $n > 4$ and $1 \le k \le \frac{\log n}{\log\log n}$. Which implies,

$$n/2(\log n - k \log\log n) \le n \left(\frac{\log n - 1}{\log n}\right)^k (\log n - k \log\log n) \le n(\log n - k \log\log n)$$

Therefore it follows that,

$$\sum_{k=1}^{\frac{\log n}{\log\log n}} n \left(\frac{\log n - 1}{\log n}\right)^k (\log n - k \log\log n) \in \Theta(n/2 \frac{\log^2 n}{\log\log n} + n \frac{\log n}{\log\log n})$$

As required.

Theorem 1 *Algorithm W performs $\Omega(n \log^2 n / \log\log n)$ work.*

Proof. Follows trivially. $\square$

It is worth mentioning that the failure factor $1/\log n$ is apparently optimal.