The parallel complexity of integer prefix summation

Torben Hagerup

Report LSI–92–18–R

The Parallel Complexity of Integer Prefix Summation *

TORBEN HAGERUP

Departament de LSI
Universitat Politècnica de Catalunya
E-08028 Barcelona, Spain

Abstract. The time required by a p-processor CRCW PRAM to add n integers of N bits each or to compute all their prefix sums is shown to be $\Theta\left(\frac{n}{p} + \frac{\log n}{\log\log p} + \log\min\left\{\frac{N}{\log p}, n\right\}\right)$. This strengthens and unifies many previous results. In particular, the new bound shows that the prefix sums of n integers of N bits each can be computed in $O(\log n/\log\log n)$ time with optimal speedup if and only if $N = 2^{O(\log n/\log\log n)}$; previously this was known only for $N = O(\log n)$.

1 Introduction

The problems of adding n integers of N bits each and of computing all their prefix sums on a PRAM are of fundamental importance and have been considered by many authors. A classical simple result states that both problems can be solved for any value of N in $O(\log n)$ time on an EREW PRAM with $O(n/\log n)$ processors, i.e., using $O(n)$ operations. This is optimal for the EREW and CREW PRAMs as regards both time and number of operations, as follows in the case of time from the lower bound of [4], and in the case of number of operations from trivial considerations. In the following we use as our model of computation the CRCW PRAM, on which faster solutions are possible.

Rajasekaran and Reif [8] showed that the prefix sums of n integers of $N = O(\log n)$ bits each can be computed with $p \geq n/\log n$ processors in time $O\left(\frac{\log n}{\log\log(p\log n/n)}\right)$. With this bound on N, the result of [8] is optimal for large values of p, but the algorithm does not exhibit optimal speedup together with a sublogarithmic running time for any value of p. Cole and Vishkin [3] showed that for $N = O(\log n)$, optimal speedup can be achieved with a running time of $O(\log n/\log\log n)$. Ragde [7] combined ideas from both earlier papers and gave a solution for $N = 1$ that uses $O(\log n/\log\log p)$ time with $p \geq n$ processors. Concentrating on cases where constant time is achievable, Parberry and Schnitger [6] showed that for every λ with $0 < \lambda \leq 1/2$, there exists $\mu > 0$ such that n integers of at most $n^{1-\lambda}$ bits each can be added in constant time with $2^{O(n^{1-\mu})}$ processors.

* Supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II).

Complementing this effort to find fast algorithms, a series of papers showed that for certain (large) values of N, adding n N-bit integers with any polynomial number of processors needs $\Omega(\log n)$ time, whereby the necessary magnitude of N was lowered by successive authors. An account of this development was given by Beame [1], who also used a very simple argument to show that the bound of $\Omega(\log n)$ holds even for $N = n$. In addition, it is known from the work of Beame and Hastad [2] that the time needed is $\Omega(\log n/\log\log n)$ with any polynomial number of processors, even for $N = 1$.

In this paper we show that the time needed by a p-processor CRCW PRAM to add n integers of N bits each or to compute all their prefix sums is

$$\Theta\left(\frac{n}{p} + \frac{\log n}{\log\log p} + \log\min\left\{\frac{N}{\log p}, n\right\}\right).$$

All of the results for the CRCW PRAM cited above can be derived from this bound, which strengthens or extends each of them in one or more ways. Specializing the parameters to the setting considered by Cole and Vishkin [3], e.g., it can be seen that the prefix sums of n N-bit integers can be computed in $O(\log n/\log\log n)$ time with optimal speedup not only for $N = O(\log n)$, as described by Cole and Vishkin, but in fact exactly if $N = 2^{O(\log n/\log\log n)}$. As another example, our bound shows that in the result of Parberry and Schnitger [6], we can actually take $\mu = \lambda$ (and this is the best possible).

Our lower bound was essentially shown by Beame [1] and Beame and Hastad [2], although it has not been expressed in this form before. The upper bound is new, but follows from a straightforward combination of known methods. Our main contribution therefore is simply the expedient organization of a number of known facts. The present exposition also pursues a secondary goal. The author considers existing accounts of optimal sublogarithmic prefix summation to be relatively difficult to understand and, in particular, ill suited for classroom teaching. Here we give an argument that is less obscure, or at least different, and that the author prefers by far in the context of teaching. It shows quite transparently why no simple change of parameters can yield a running time better than $O(\log n/\log\log p)$.

2 Preliminaries

A PRAM is a synchronous parallel machine consisting of a finite collection of processors numbered $0, 1, 2, \ldots$ and a global memory accessible to all processors. The PRAM model is one of the most popular models of parallel computation. It comes in three main submodels: The EREW PRAM does not allow concurrent access to the same memory cell by several processors, the CREW PRAM allows concurrent reading, but not concurrent writing, and the CRCW PRAM allows both concurrent reading and concurrent writing, with the effect of concurrent writing fixed in some way (see below).

For the lower bound in this paper we will use the so-called Abstract CRCW PRAM of [1]. Loosely speaking, each processor of an Abstract CRCW PRAM has infinite local computational power. More precisely, in each step of the computation each processor does the following: In the beginning of the

step, the processor is in a certain state. Based on this state, it chooses a global memory cell and reads the contents of this cell. Then, depending on the old state and the value read, it enters a new state. Finally, based on the new state, it chooses a global memory cell and a value and attempts to store the chosen value in the chosen cell. If several processors attempt to write to the same global memory cell in the same step, only the lowest-numbered among them is successful (this is known as the PRIORITY rule). There are no requirements of uniformity and no restrictions on the complexity of the state transitions of processors or on the values that can be stored in global memory cells, which means that the model is very powerful.

Our upper bound applies to all CRCW PRAM variants with the property that the OR of n bits can be computed in constant time using n processors and $O(n)$ space. Variants of the CRCW PRAM with this capability include the standard COMMON, ARBITRARY and PRIORITY PRAMs, but also the weak TOLERANT PRAM of [5]. We assume a local computational power at least corresponding to what [6] calls the minimal instruction set. Essentially, this means the availability of addition, subtraction and shift operations, but not necessarily multiplication and division. As a minimum, we assume that the available operations can be executed in constant time on integer operands of absolute value bounded by $\max\{p, n, m\}$, where p is the number of processors of the machine under consideration, n is the input size, and m is the largest absolute value of an input number.

Throughout the paper and for any integer $k \geq 1$, an integer of k bits (equivalently, a k-bit integer) is an element of the set $\{0, \ldots, 2^k - 1\}$. For the lower bound we consider the problem of adding n integers $a_1, \ldots, a_n$ of N bits each, i.e., of computing $\sum_{i=1}^{n} a_i$. For all positive integers n, p and N, denote by $T(n, p, N)$ the number of time steps needed by a p-processor Abstract CRCW PRAM to solve this problem. For the upper bound we consider the related problem of computing all prefix sums of $a_1, \ldots, a_n$, i.e., the quantities $\sum_{j=1}^{i} a_j$, for $i = 1, \ldots, n$. As follows from our result, computing all prefix sums of a sequence of integers is no more difficult, up to a constant factor, than just adding them. Although we consider the prefix summation of nonnegative integers only, it is easy to extend our result to the prefix summation of arbitrary integers of absolute value bounded by $2^N - 1$: Simply compute the prefix sums of the positive and of the negative input numbers separately, and finally add the two resulting sequences element by element.

We always use $\log x$ to denote $\max\{\log_2 x, 1\}$, for arbitrary $x > 0$. $E_1 = O(E_2)$, where E_1 and E_2 are expressions, means that there is a constant c such that $E_1 \leq cE_2$ for *all* legal values of the parameters occurring in E_1 and E_2. The meaning of $E_1 = \Omega(E_2)$ and $E_1 = \Theta(E_2)$ is defined analogously.

3 The lower bound

This section argues lower bounds of $\Omega(n/p)$, $\Omega(\log n / \log\log p)$ and $\Omega\left(\log \min\left\{\frac{N}{\log p}, n\right\}\right)$ on $T(n, p, N)$, from which the overall lower bound follows immediately.

Lemma 3.1: For all $N \geq 1$, $T(n,p,N) = \Omega(n/p)$.

Proof: The claim is trivial, since p processors need $\Omega(n/p)$ time to produce an output of size n. ∎

Lemma 3.2: For all $N \geq 1$, $T(n,p,N) = \Omega(\log n/\log\log p)$.

Proof: For all $N \geq 1$, computing the parity of n bits clearly reduces to adding n N-bit integers. More precisely, given any Abstract PRAM that adds n N-bit integers using p processors and T time steps, it is easy to derive an Abstract PRAM that computes the parity of n bits using p processors and $T+1$ time steps — the extra step is used to store the parity of the sum of the n input bits in an output cell. But Theorem 4.1 of [2] states that for some integer n_0, any Abstract PRAM that computes the parity of $n \geq n_0$ bits in T time steps has at least $2^{(1/96)n^{1/T}-2}$ processors. Hence for $n \geq n_0$,

$$p \geq 2^{(1/96)n^{1/(T(n,p,N)+1)}-2},$$

from which follows that $T(n,p,N) = \Omega(\log n/\log\log p)$. ∎

Lemma 3.3: $T(n,p,N) = \Omega\left(\log \min\left\{\frac{N}{\log p}, n\right\}\right)$.

Proof: For each integer $b \geq 2$, let $Encode_b$ be the function from $\{0,\ldots,b-1\}^*$ to $\{0,1,\ldots\}$ with $Encode_b(a_1,\ldots,a_n) = \sum_{i=1}^{n} a_i b^{i-1}$; note that if b is a power of 2, then $Encode_b$ simply concatenates the $(\log b)$-bit binary representations of its arguments. Beame [1] showed that for all integers $p \geq 1$ and $n, b \geq 2$, any Abstract p-processor PRAM that computes $Encode_b$ on inputs of size n uses at least

$$\log n + 1 - \log(1 + \log_b(2p))$$

time steps. As observed by Beame, the computation of $Encode_b$ is a special case of addition. More precisely, the following holds for all positive integers n, p, N, T, b and m with $b \geq 2$, $m \leq n$ and $(b-1)b^{m-1} \leq 2^N - 1$: If there is an Abstract p-processor PRAM that adds n N-bit integers in T time steps, then there is an Abstract p-processor PRAM that computes $Encode_b$ on inputs of size m in T time steps — the latter simply interprets its ith input value a_i as $a_i b^{i-1}$, for $i = 1,\ldots,m$, assumes $n-m$ fictitious additional input values equal to zero, and then adds the resulting integers. We use this to prove the lower bound. Assume $n \geq 2$ and consider two cases, depending on the size of N relative to n.

Case 1: $N \leq n$. By the above observation, used with $b = 2$ and $m = N$, $T(n,p,N) \geq \log N + 1 - \log(1 + \log(2p))$ and hence $T(n,p,N) = \Omega\left(\log \frac{N}{\log p}\right)$.

Case 2: $N > n$. By the same observation, now used with $b = 2^{\lfloor N/n \rfloor}$ and $m = n$,

$$T(n,p,N) \geq \log n + 1 - \log\left(1 + \frac{\log(2p)}{\lfloor N/n \rfloor}\right).$$

If $\frac{\log(2p)}{\lfloor N/n \rfloor} \leq 8$, then $T(n,p,N) = \Omega(\log n)$. On the other hand, if $\frac{\log(2p)}{\lfloor N/n \rfloor} > 8$, then

$$\log\left(1 + \frac{\log(2p)}{\lfloor N/n \rfloor}\right) \leq \log\left(\frac{\log(2p)}{\lfloor N/n \rfloor}\right) + 1 \leq \log\left(\frac{\log p}{N/n}\right) + 3$$

and hence

$$T(n,p,N) \geq \log n - \log\left(\frac{\log p}{N/n}\right) - 2 = \log_2 \frac{N}{\log p} - 2. \quad \blacksquare$$

Theorem 3.4: $T(n,p,N) = \Omega\left(\frac{n}{p} + \frac{\log n}{\log\log p} + \log\min\left\{\frac{N}{\log p}, n\right\}\right)$.

Proof: By Lemmas 3.1, 3.2 and 3.3. ∎

4 The upper bound

We begin by showing that prefix sums can be computed in constant time with a large number of processors. This was observed previously by many authors.

Lemma 4.1: Given $2n$ integers $a_1,\ldots,a_n,s_1,\ldots,s_n$, n processors can decide in constant time using $O(n)$ space whether $s_1,\ldots,s_n$ are the prefix sums of $a_1,\ldots,a_n$, i.e., whether $s_i = \sum_{j=1}^{i} a_j$, for $i = 1,\ldots,n$.

Proof: Let the processors check the condition $s_1 = a_1$ and the $n-1$ conditions $s_i = s_{i-1} + a_i$, for $i = 2,\ldots,n$, and then use the evaluation of an n-way OR to accept the input if and only if all n conditions are satisfied. ∎

Lemma 4.2: If k is a power of 2, the prefix sums of k integers of k bits each can be computed in constant time using 2^{k^4} processors and $O(2^{k^4})$ space.

Proof: The sum of k integers of k bits each is at most $k(2^k-1) \leq 2^k(2^k-1) < 2^{2k}$. It therefore suffices to compute the integer $S = Encode_{2^{2k}}(s_1,\ldots,s_k)$, where $s_1,\ldots,s_k$ are the prefix sums of the input numbers. But $0 \leq S < (2^{2k})^k = 2^{2k^2}$. Hence associate a team of k processors with each integer in the set $\{0,\ldots,2^{2k^2}-1\}$ and let each team use the algorithm of Lemma 4.1 to check whether S equals its associated integer, in which case the team can easily produce the desired output. The total number of processors needed is $k\cdot 2^{2k^2}$, which for $k \geq 2$ is bounded by $2^k \cdot 2^{k^3} \leq 2^{k^4}$. The processor allocation is easy: Each processor simply interprets the least significant $2k^2$ bits of its processor number as the integer associated with its team, and the next $\log k$ bits as its number within that team. ∎

Recall that the proof of Lemma 3.3 depended on Beame's lower bound for the *Encode* function. Using the method of the proof of Lemma 4.2, we now give an upper bound for this function.

Lemma 4.3: If b and m are powers of 2, then $Encode_b$ can be computed on inputs of size m using constant time, $m\cdot b^m$ processors and $O(m\cdot b^m)$ space.

Proof: The output to be computed is an integer S in the set $\{0,\ldots,b^m-1\}$. Associate a team of m processors with each integer in this set and let each team check in constant time whether its associated integer is equal to S. ∎

We are now ready to prove an upper bound for prefix summation that matches the lower bound of the previous section. We distinguish between two cases, depending on the size of p relative to n, the case of optimal speedup and the fast case. We begin with the fast case, which is somewhat simpler.

Lemma 4.4: For $p \geq 2^{(4 \log n)^8}$, the prefix sums of n integers of N bits each can be computed using p processors,

$$O\left(\frac{\log n}{\log \log p} + \log \min\left\{\frac{N}{\log p}, n\right\}\right)$$

time and $O(p)$ space.

Proof: We will assume that n and hence p is larger than some unspecified constant. We can also assume that $N/\log p \leq n$, since otherwise the bound is $O(\log n)$ and the problem is trivial; in particular, this means that $N \leq \sqrt{p}$. Let $r = \lceil \log n \rceil$ and note that the sum of n integers of r bits each is an integer of $2r$ bits. Choose k as the smallest power of 2 no smaller than $(\log p)^{1/8}$. Since $r \leq 2 \log n$ and $p \geq 2^{(4 \log n)^8}$, we have $k \geq 2r$. Now consider an ordered tree T with n leaves, numbered $1, \ldots, n$ from left to right, at most n internal nodes, maximum degree at most k and height $h = O(\log n/\log k) = O(\log n/\log \log p)$. For each vertex v of T, let $d(v)$ be the number of children of v, and for $i = 1, \ldots, d(v)$, define the ith *subtree* of v as the maximal subtree of T rooted at the ith child of v. Assume that for some $\delta > 0$, each internal node v of T is a $d(v)$-way prefix sum device (PSD) with delay δ, defined as follows. A PSD is like a gate in the circuit model of computation, except that it has internal memory and therefore needs to be explicitly activated (or "clocked") when it is to operate. An l-way PSD with delay δ has l input lines, numbered $1, \ldots, l$, one output line and $l + 1$ internal registers $S_0, \ldots, S_l$. When an integer a_i of k bits is applied to the ith input line, for $i = 1, \ldots, l$, the PSD can be activated, in which case it computes $s_i = \sum_{j=1}^{i} a_j$ and stores s_i in S_i, for $i = 0, \ldots, l$ (thus $S_0 = 0$ always), and applies s_l to its output line at most δ time units after its activation. For each internal node v of T and for $i = 1, \ldots, d(v)$, identify the ith input line of v, considered as a PSD, with the edge in T between v and its ith child. Finally assume that each leaf node of T is a 1-way PSD with delay δ.

It is easy to see that if we apply an r-bit integer a_i to the input line of the ith leaf of T before some time t_0, for $i = 1, \ldots, n$, and then activate the nodes at level j at time $t_0 + j\delta$, for $j = 0, \ldots, h$, then at time $t_0 + (h + 1)\delta$ the root of T will apply $\sum_{i=1}^{n} a_i$ to its output line; in particular, we have taken care to ensure that intermediate values do not become too large. Furthermore, for $i = 1, \ldots, n$, the prefix sum $\sum_{j=1}^{i} a_j$ can be derived from T in $O(h)$ time by a single processor as follows: Initialize a variable x to the value a_i and let v be the ith leaf of T. Then for each proper ancestor u of v in T, let j be the number of the subtree of u that contains v and add to x the value of the register S_{j-1} of u. As is easy to see, the final value of x is $\sum_{j=1}^{i} a_j$.

In the context of the original prefix summation problem, we can break each input number into *blocks* of r bits each and use the above observation to compute the prefix sums of each group of n blocks in corresponding positions, one from each input number. According to Lemma 4.2, 2^{k^4} processors can

implement a k-way PSD with constant delay, so that the computation can be carried out in time $O(h)$, as desired. Since there are at most $2n$ nodes in T and at most N groups of n blocks each, the total number of processors needed is at most

$$2^{k^4} \cdot 2n \cdot N \leq 2^{2^4\sqrt{\log p}} \cdot 2p^{1/4} \cdot \sqrt{p} \leq p.$$

At this point each of the n output values to be computed is represented by a collection of $\lceil N/r \rceil$ "pieces", one for each block position, and the remaining problem is to add these $\lceil N/r \rceil$ pieces, each of which is an integer of $2r$ bits. Observe that if we break each piece into an r-bit "low-order half" and an r-bit "high-order half", then separately adding the low-order halves and separately adding the high-order halves are very simple tasks; indeed, they are instances of the computation of $Encode_{2^r}$ on inputs of size at most $\lceil N/r \rceil$. We actually use this observation only to add groups of m consecutive halves, where m is chosen as a power of 2 with

$$\tfrac{1}{16} \log p \leq mr \leq \tfrac{1}{8} \log p.$$

By Lemma 4.3, this can be done in constant time using a total of at most

$$m \cdot 2^{mr} \cdot N \cdot n \leq \log p \cdot p^{1/8} \cdot p^{1/2} \cdot p^{1/4} \leq p$$

processors.

Now each output value is represented by a collection of $O(N/\log p)$ larger pieces. We finally add these in the obvious binary-tree fashion in $O\left(\log \frac{N}{\log p}\right)$ time using at most $N \cdot n \leq p$ processors. ∎

This concludes the description of the fast case. We now turn to the case of optimal speedup.

Lemma 4.5: The prefix sums of n integers of N bits each can be computed using p processors,

$$O\left(\frac{n}{p} + \frac{\log n}{\log \log n} + \log \min\{N, n\}\right)$$

time and $O(n + p)$ space.

Proof: We use the same basic algorithm as in the fast case. There are two differences: Firstly, since we can no longer rely on the relation $p \geq 2^{(4 \log n)^8}$ to provide us with an abundance of processors, we have to introduce a preprocessing stage that achieves the necessary *processor advantage*, i.e., ratio of number of available processors to problem size. Secondly, even after the preprocessing stage we will not have enough processors to implement prefix sum devices capable of handling blocks of $\lceil \log n \rceil$ bits directly; note that prefix sums of about this many bits will arise during the computation in the tree T even if the values fed into the leaves of T are single bits. The solution to this problem is to subdivide each block into smaller units called *bytes*, each of which can be handled directly, and then to let T operate in a pipelined fashion: Each PSD receives each of its inputs byte by byte and produces its output byte by byte in the same way.

As before, we ignore values of n smaller than some unspecified constant. Since the standard parallel prefix summation algorithm works in $O(n/p)$ time if $p < n/\log n$, we can assume that $p \geq n/\log n$. We also assume that $N \leq \sqrt{n}$, since otherwise the bound to be shown is $O(\log n)$. For the preprocessing, divide the n input numbers into at most $n/(\log n \cdot 2^{\log n/\log\log n} \cdot N)$ groups of $O(\log n \cdot 2^{\log n/\log\log n} \cdot N)$ consecutive numbers each. Use the standard parallel algorithm to compute all prefix sums within each group (the *local* prefix sums), which takes $O(n/p + \log n/\log\log n + \log N)$ time. In particular, this computes for each group its *group sum*, i.e., the sum of all numbers in the group. Use the algorithm to be described below to compute the prefix sums of the group sums (the *global* prefix sums). Now each prefix sum of the original problem corresponding to some input element can be obtained as the sum of the local prefix sum computed for that element and the global prefix sum of the preceding group. Computing all prefix sums in this way takes $O(\lceil n/p \rceil)$ time.

All that remains is to describe the computation of the global prefix sums. Let us redefine n to be the number of groups, i.e., the size of the (global) problem at hand, in which case the number of available processors is at least $n \cdot 2^{\log n/\log\log n} \cdot N$. Also note that the group sums are integers of $N + r$ bits.

Let k be the smallest power of 2 no smaller than $(\log n)^{1/5}$ and consider a tree T as before, i.e., with maximum degree at most k and height $h = O(\log n/\log k) = O(\log n/\log\log n)$. We again break each input number into blocks of $r = \lceil \log n \rceil$ bits each and compute the prefix sums of each group of n blocks in corresponding positions using one copy of T. The total number of processors needed to implement every node of every tree as a PSD with constant delay is at most

$$2^{k^4} \cdot n \cdot (N + r) \leq p.$$

As mentioned above, since $r > k$, the computation in T cannot be quite as straightforward as in the fast case, and we have to resort to pipelining. Define a *byte* as k consecutive bits. We modify the prefix sum devices as follows: An l-way PSD, in addition to the $l + 1$ registers $S_0, \ldots, S_l$, now has l carry registers $C_1, \ldots, C_l$ and a multiplier register M. All registers are initialized to 0, except that M holds the value 1 initially. Assume that a PSD is activated when $C_1, \ldots, C_l$ and M contain the values $c_1, \ldots, c_l$ and m, respectively, and when a k-bit integer a_i is applied to the ith input line, for $i = 1, \ldots, l$. For $i = 1, \ldots, l$, the PSD then computes $s_i = c_i + \sum_{j=1}^{i} a_j$, splits s_i into the low-order k bits s_i' and the high-order k bits s_i'', adds s_i' scaled by m (i.e., $m \cdot s_i'$) to S_i and stores s_i'' as the new value of C_i. Finally $m \cdot 2^k$ is stored in M, and s_l' is applied to the output line. It is easy to see that the PSD works according to intention, i.e., if nonnegative integers $a_1, \ldots, a_l$ are input synchronously and byte by byte, then their sum is output byte by byte, each output byte being produced a constant time after the consumption of the corresponding input bytes, and the prefix sums of $a_1, \ldots, a_l$ accumulate in the registers $S_1, \ldots, S_l$. Then the entire tree T computes as intended. Even with the modifications described above, 2^{k^4} processors are clearly sufficient to implement a k-way PSD with constant delay. The time needed by the pipelined computation to process blocks of r bits, one byte at a time, is

$$O(\lceil r/k \rceil + h) = O((\log n)^{4/5} + \log n/\log\log n) = O(\log n/\log\log n).$$

In contrast with the fast case, the final computation of each output value from $\lceil (N+r)/r \rceil$ pieces can be done in the straightforward way. This needs $O(\log N)$ time and at most $n \cdot (N+r) \leq p$ processors. ∎

Theorem 4.6: The prefix sums of n integers of N bits each can be computed using p processors,

$$O\left(\frac{n}{p} + \frac{\log n}{\log\log p} + \log \min\left\{\frac{N}{\log p}, n\right\}\right) \quad ,$$

time and $O(n+p)$ space.

Proof: For $p \geq 2^{(4\log n)^8}$ the assertion was proved in Lemma 4.4. For $p < 2^{(4\log n)^8}$ we have $\log\log p = O(\log\log n)$, so that the time bound is at least $\Theta(\log n/\log\log n)$. Furthermore, $\log N \leq \log n/\log\log n$ unless $N \geq 2^{\log n/\log\log n}$, in which case $\log N = O(\log(N/\log p))$ for the range of p under consideration. The claim therefore follows from Lemma 4.5. ∎

References

[1] P. Beame, Limits on the Power of Concurrent-Write Parallel Machines, *Inform. and Comput.* **76** (1988), pp. 13–28.

[2] P. Beame and J. Hastad, Optimal Bounds for Decision Problems on the CRCW PRAM, *J. ACM* **36** (1989), pp. 643–670.

[3] R. Cole and U. Vishkin, Faster Optimal Parallel Prefix Sums and List Ranking, *Inform. and Comput.* **81** (1989), pp. 334–352.

[4] S. Cook, C. Dwork and R. Reischuk, Upper and Lower Time Bounds for Parallel Random Access Machines Without Simultaneous Writes, *SIAM J. Comput.* **15** (1986), pp. 87–97.

[5] V. Grolmusz and P. Ragde, Incomparability In Parallel Computation, *in* Proc. 28th Annual Symposium on Foundations of Computer Science (1987), pp. 89–98.

[6] I. Parberry and G. Schnitger, Parallel Computation with Threshold Functions, *J. Comput. System Sci.* **36** (1988), pp. 278–302.

[7] P. Ragde, The Parallel Simplicity of Compaction and Chaining, *in* Proc. 17th International Colloquium on Automata, Languages and Programming (1990), Springer Lecture Notes in Computer Science, Vol. 443, pp. 744–751.

[8] S. Rajasekaran and J. H. Reif, Optimal and Sublogarithmic Time Randomized Parallel Sorting Algorithms, *SIAM J. Comput.* **18** (1989), pp. 594–607.