Lower Bounds for Dynamic Partial Sums

Mihai Pătraşcu

Let $\mathbb{G}$ be a group. The partial sums problem asks to maintain an array $A[1\,..\,n]$ of group elements, initialized to zeroes (a.k.a. the identity), under the following operations:

UPDATE(k, Δ)**:** modify $A[k] \leftarrow \Delta$, where $\Delta \in \mathbb{G}$.
QUERY(k)**:** returns the partial sum $\sum_{i=1}^{k} A[i]$.

For concreteness, let us work on a machine with w-bits words ($w \geq \lg n$), and take $\mathbb{G}$ to be $\mathbb{Z}/2^w\mathbb{Z}$, i.e. integer arithmetic on machine words (modulo 2^w). Then, the partial sums problem can be solved trivially in $O(\lg n)$ time per operation, using augmented binary trees.

In this note, we describe a simple lower bound, originating in [1], showing that the problem requires $\Omega(\lg n)$ time per operation.

1 The Hard Instance

The proof works for any choice of $\mathbb{G}$. Letting $\delta = \lg |\mathbb{G}|$, we will show that any data structure requires an average running time of $\Omega(\frac{\delta}{w} \cdot n \lg n)$ to execute a sequence of n updates and n queries chosen from a particular distribution. If $\delta = w$, we obtain an amortized $\Omega(\lg n)$ bound per operation.

The hard instance is described by a random permutation π of size n, and a uniformly random sequence $\langle \Delta_1, \ldots, \Delta_n \rangle \in \mathbb{G}^n$. Then, the hard instance is:

```
1  for t ← 1 to n                    ▷ At "time t" we do the following:
2      do QUERY(π(t))
3         UPDATE(π(t), Δ_t)
```

A very useful visualization of an instance is as a two-dimensional chart, with time on one axis, and the index in A on the other axis. The answer to a query QUERY$(\pi(t))$ is the sum of the update points in the rectangle $[0, t] \times [0, \pi(t)]$; these are the updates which have already occurred, and affect indices relevant to the partial sum. See Figure 1 (a).

2 Information Transfer

Let $t_0 < t_1 < t_2$ (with t_1 non-integral to avoid ties). We will be preoccupied by the interaction between two adjacent intervals of operations: the time intervals $[t_0, t_1]$ and $[t_1, t_2]$. Since the algorithm cannot maintain state between operations, such interaction can only be caused by the algorithm writing a cell during the first interval and reading it during the second.

Definition 1. *The* information transfer $IT(t_0, t_1, t_2)$ *is the set of memory locations which:*

- *were read at a time* $t_r \in [t_1, t_2]$.
- *were written at a time* $t_w \in [t_0, t_1]$, *and not overwritten during* $[t_w + 1, t_r]$.

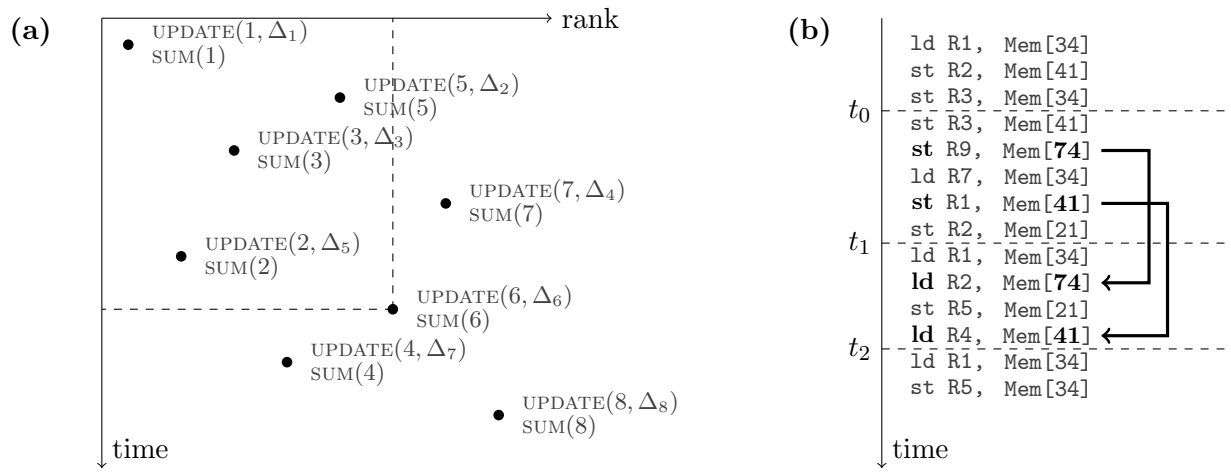

Figure 1: (a) An instance of the partial sums problem. The query QUERY(6) occurring at time 6 has the answer $\Delta_1 + \Delta_2 + \Delta_3 + \Delta_5$. (b) The execution of a hypothetical cell-probe algorithm. $IT(t_0, t_1, t_2)$ consists of cells 41 and 74.

The definition is illustrated in Figure 1 (b). Observe that the information transfer is a function of the algorithm, the permutation π, and the sequence Δ.

For now, let us concentrate on bounding $|IT(t_0, t_1, t_2)|$, ignoring the question of how this might be useful. Intuitively, any dependence of the queries from $[t_1, t_2]$ on updates from the interval $[t_0, t_1]$ must come from the information in the cells $IT(t_0, t_1, t_2)$. Indeed, $IT(t_0, t_1, t_2)$ captures the only possible information flow between the intervals: an update happening during $[t_0, t_1]$ cannot be reflected in a cell written before time t_0.

Let us formalize this intuition. We break the random sequence $\langle \Delta_1, \ldots, \Delta_n \rangle$ into the sequence $\Delta_{[t_0,t_1]} = \langle \Delta_{t_0}, \ldots, \Delta_{\lfloor t_1 \rfloor} \rangle$, and $\Delta^\star$ containing all other values. The values in $\Delta^\star$ are uninteresting to our analysis, so fix them to some arbitrary $\mathbf{\Delta}^\star$. Let A_t be the answer of the query QUERY$(\pi(t))$ at time t. We write $A_{[t_1,t_2]} = \langle A_{\lceil t_1 \rceil}, \ldots, A_{t_2} \rangle$ for the answers to the queries in the second interval.

In information theoretic terms, the observation that all dependence of the interval $[t_1, t_2]$ on the interval $[t_0, t_1]$ is captured by the information transfer, can be reformulated as saying that the *entropy* of the observable outputs of interval $[t_1, t_2]$ (i.e., the query results $A_{[t_1,t_2]}$) is bounded by the information transfer:

Lemma 2. $\mathrm{H}\big(A_{[t_1,t_2]} \;\big|\; \Delta^\star = \mathbf{\Delta}^\star\big) \;\le\; w \;+\; 2w \cdot \mathbf{E}\big[|IT(t_0, t_1, t_2)| \;\big|\; \Delta^\star = \mathbf{\Delta}^\star\big]$.

Proof. The bound follows by proposing an *encoding* for $A_{[t_1,t_2]}$, since the entropy is upper bounded by the average length of any encoding. Our encoding is essentially the information transfer; formally, it stores:

- first, the cardinality $|IT(t_0, t_1, t_2)|$, in order to make the encoding prefix free.
- the *address* of each cell; an address is at most w bits in our model.
- the *contents* of each cell at time t_1, which takes w bits per cell.

The average length of the encoding is $w \;+\; 2w \cdot \mathbf{E}\big[|IT(t_0, t_1, t_2)| \;\big|\; \Delta^\star = \mathbf{\Delta}^\star\big]$ bits, as needed. To finish the proof, we must show that the information transfer actually encodes $A_{[t_1,t_2]}$; that is, we must give a *decoding algorithm* that recovers $A_{[t_1,t_2]}$ from $IT(t_0, t_1, t_2)$.

Our decoding algorithm begins by simulating the data structure during the time period $[1, t_0-1]$; this is possible because $\Delta^\star$ is fixed, so all operations before time t_0 are known. It then *skips* the time period $[t_0, t_1]$, and simulates the data structure again during the time period $[t_1, t_2]$. Of course, simulating the time period $[t_1, t_2]$ recovers the answers $A_{[t_1,t_2]}$, which is what we wanted to do.

To see why it is possible to simulate $[t_1, t_2]$, consider a read instruction executed by a data structure operation during $[t_1, t_2]$. Depending on the time t_w when the cell was last written, we have the following cases:

$t_w > t_1$**:** We can recognize this case by maintaining a list of memory locations written during the simulation; the data is immediately available.

$t_0 < t_w < t_1$**:** We can recognize this case by examining the set of addresses in the encoding; the cell contents can be read from the encoding.

$t_w < t_0$**:** This is the default case, if the cell doesn't satisfy the previous conditions. The contents of the cell is determined from the state of the memory upon finishing the first simulation up to time $t_0 - 1$. □

3 Interleaves

In the previous section, we showed an upper bound on the dependence of $[t_1, t_2]$ on $[t_0, t_1]$; we now aim to give a lower bound. Refer to the example in Figure 2 (a). The information that the queries in $[t_1, t_2]$ *need to know* about the updates in $[t_0, t_1]$ is the sequence $\langle \Delta_6,\ \Delta_6+\Delta_3+\Delta_4,\ \Delta_6+\Delta_3+\Delta_4+\Delta_5\rangle$. Equivalently, the queries need to know $\langle \Delta_6,\ \Delta_3+\Delta_4,\ \Delta_5\rangle$, which are three independent random variables, uniformly distributed in the group $\mathbb{G}$.

This required information comes from *interleaves* between the update indices in $[t_0, t_1]$, on the one hand, and the query indices in $[t_1, t_2]$, on the other. See Figure 2 (b).

Definition 3. *If one sorts the set $\{\pi(t_0), \ldots, \pi(t_2)\}$, the interleave number $IL(t_0, t_1, t_2)$ is defined as the number of transitions between a value $\pi(i)$ with $i < t_1$, and a consecutive value $\pi(j)$ with $j > t_1$.*

The interleave number is only a function of π. Figure 2 suggests that interleaves between two intervals cause a large dependence of the queries $A_{[t_1,t_2]}$ on the updates $\Delta_{[t_1,t_2]}$, i.e. $A_{[t_1,t_2]}$ has large conditional entropy, even if all updates outside $\Delta_{[t_1,t_2]}$ are fixed:

Lemma 4. $\mathrm{H}\big(A_{[t_1,t_2]} \;\big|\; \Delta^\star = \boldsymbol{\Delta}^\star\big) \;=\; \delta \cdot IL(t_0, t_1, t_2)$.

Proof. Each answer in $A_{[t_1,t_2]}$ is a sum of some random variables from $\Delta_{[t_0,t_1]}$, plus a constant that depends on the fixed $\Delta^\star$. Consider the indices $L = \{\pi(t_0), \ldots, \pi(\lfloor t_1 \rfloor)\}$ from the first interval, and $R = \{\pi(\lceil t_1 \rceil), \ldots, \pi(t_2)\}$ from the second interval. Relabel the indices of R as $r_1 < r_2 < \cdots$ and consider these r_i's in order:

- If $L \cap [r_{i-1}, r_i] = \emptyset$, the answer to $\textsc{query}(r_i)$ is the same as for $\textsc{query}(r_{i-1})$, except for a different constant term. The answer to $\textsc{query}(r_i)$ contributes nothing to the entropy.
- Otherwise, the answer to $\textsc{query}(r_i)$ is a random variable independent of all previous answers, due to the addition of random Δ's to indices $L \cap [r_{i-1}, r_i]$. This random variable is uniformly distributed in $\mathbb{G}$, so it contributes δ bits of entropy. □

Comparing Lemmas 4 and 2, we see that $\mathbf{E}\big[|IT(t_0, t_1, t_2)| \;\big|\; \Delta^\star = \boldsymbol{\Delta}^\star\big] \geq \frac{\delta}{2w} \cdot IL(t_0, t_1, t_2) - 1$ for any fixed $\boldsymbol{\Delta}^\star$. By taking expectation over $\Delta^\star$, we have:

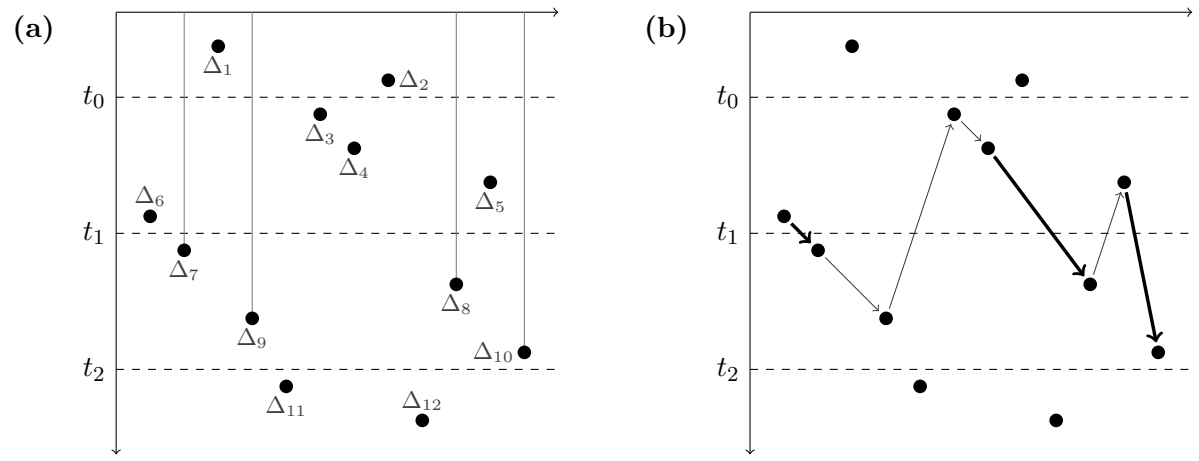

Figure 2: (a) The vertical lines describe the information that the queries in $[t_1, t_2]$ from the updates in $[t_0, t_1]$. (b) The interleave number $IL(t_0, t_1, t_2)$ is the number of down arrows crossing t_1, where arrows indicate left-to-right order.

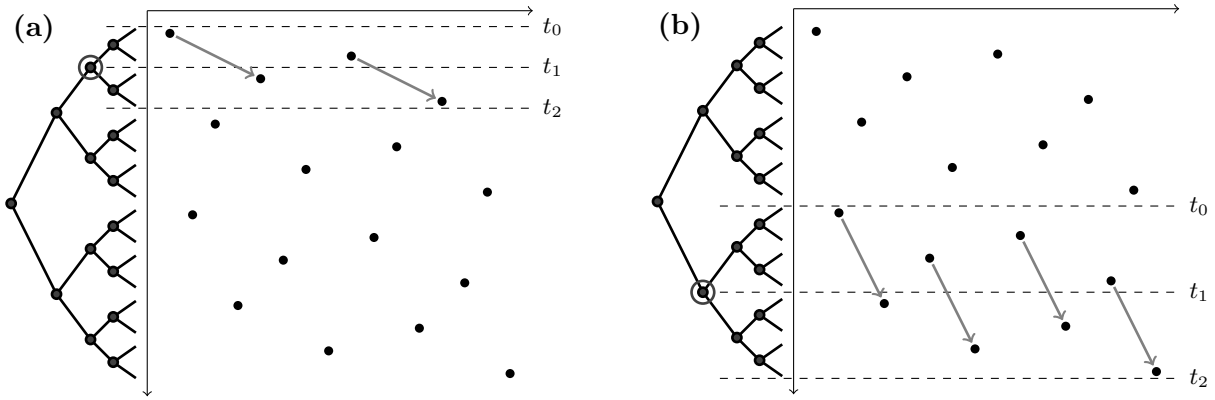

Figure 3: The lower bound tree for $n = 16$. The nodes have interleave 2 in (a), and 4 in (b).

Corollary 5. *For any fixed π, $t_0 < t_1 < t_2$, and any algorithm solving the partial sums problem, we have* $\mathbf{E}_\Delta\big[|IT(t_0, t_1, t_2)|\big] \geq \frac{\delta}{2w} \cdot IL(t_0, t_1, t_2) - 1$.

4 A Tree For The Lower Bound

The final step of the algorithm is to consider the information transfer between many pairs of intervals, and piece together the lower bounds from Corollary 5 into one lower bound for the total running time of the data structure. The main trick for putting together these lower bounds is to consider a lower-bound tree $\mathcal{T}$: an arbitrary binary tree with n leaves, where each leaf denotes a time unit (a query and update pair). In other words, $\mathcal{T}$ is built "over the time axis," as in Figure 3.

For each internal node v of $\mathcal{T}$, we consider the time interval $[t_0, t_1]$ spanning the left subtree, and the interval $[t_1, t_2]$ spanning the right subtree. We then define:

- the information transfer *through* the node: $IT(v) = |IT(t_0, t_1, t_2)|$. Essentially, $IT(v)$ counts

the cells written in the left subtree of v, and read in the right subtree.

- the interleave at the node: $IL(v) = IL(t_0, t_1, t_2)$.

Theorem 6. *For any algorithm and fixed π, the expected running time of the algorithm over a random sequence Δ is at least $\frac{\delta}{2w} \sum_{v \in \mathcal{T}} IL(v) \; - \; n$.*

Proof. First, observe that on any problem instance (any fixed Δ), the number of read instructions executed by the algorithm is at least $\sum_{v \in \mathcal{T}} IT(v)$. Indeed, for each read instruction, let t_r be the time it is executed, and $t_w \leq t_r$ be the time when the cell was last written. If $t_r = t_w$, we can ignore this trivial read. Otherwise, this read instruction appears in the information transfer through exactly one node: the lowest common ancestor of t_w and t_r. Thus, $\sum_v IT(v)$ never double-counts a read instruction.

Now we apply Corollary 5 to each node, concluding that for each v, $\mathbf{E}_\Delta[IT(v)] \geq \frac{\delta}{2w} \cdot IL(v) - 1$. Thus, the total expected running time is at least $\frac{\delta}{2w} \sum_v IL(v) - (n-1)$. It is important to note that each lower bound for $|IT(v)|$ applies to the expected value under the same distribution (a uniformly random sequence Δ). Thus we may sum up these lower bounds to get a lower bound on the entire running time, using linearity of expectation. □

To complete our lower bound, it remains to design an access sequence π that has high total interleave, $\sum_{v \in \mathcal{T}} IL(v) = \Omega(n \lg n)$, for some lower-bound tree $\mathcal{T}$. From now on, assume n is a power of 2, and let $\mathcal{T}$ be a perfect binary tree.

Claim 7. *If π is a uniformly random permutation, $\mathbf{E}_\pi\big[\sum_{v \in \mathcal{T}} IL(v)\big] = \Omega(n \lg n)$.*

Proof. Consider a node v with $2k$ leaves in its subtree, and let S be the set of indices touched in v's subtree, i.e. $S = \{\pi(t_0), \ldots, \pi(t_2)\}$. The interleave at v is the number of down arrows crossing from the left subtree to the right subtree, when S is sorted; see Figure 2 (b) and Figure 3. For two indices $j_1 < j_2$ that are consecutive in S, the probability that j_1 is touched in the left subtree, and j_2 is touched in the right subtree will be $\frac{k}{2k} \cdot \frac{k}{2k-1} > \frac{1}{4}$. By linearity of expectation over the $2k-1$ arrows, $\mathbf{E}_\pi[IL(v)] = (2k-1) \cdot \frac{k}{2k} \cdot \frac{k}{2k-1} = \frac{k}{2}$. Summing up over all internal nodes v gives $\mathbf{E}_\pi\big[\sum_v IL(v)\big] = \frac{1}{4} n \log_2 n$. □

Thus, any algorithm requires $\Omega(\frac{\delta}{w} \cdot n \lg n)$ cell probes in expectation on problem instances given by random Δ and random π. This shows our $\Omega(\lg n)$ amortized lower bound for $\delta = w$.

References

[1] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also SODA'04 and STOC'04.