Computational Geometry (2023)
I/O-Efficient Algorithms

Peyman Afshani

September 22, 2023

1 Introduction

In this project, we will practice a bit with the idea of coming up with new models of computation to address different technologies. Many of the new emerging technologies are too complicated to act as the starting project, so instead we first visit an old device and then try come up with some simpler models for a new technology.

2 Magnetic Tapes

Our journey starts with *magnetic tapes*!

Currently, magnetic tapes are still the best choice when it comes to minimizing cost per storage for large systems and as a result they are still in use for back up or archiving. However, it is generally assumed that magnetic tapes are poor choices for any computational system and thus they are only relegated to generally being backup devices. In this project, we will try to explore to see if such devices can be used for processing data as well. As the first step of doing so, we will have to try and capture the essential limitations of magnetic tapes in a theoretical model.

Figure 1: Inside of a magnetic tape (wikimedia)

2.1 Simpler Technical Specification

The standards for the 9th generation of Linear Tape-Open (LTO-9) technology was released in 2021 with the later generations scheduled to be released in the upcoming years and thus this is still an actively developing technology. The 9-th generation LTO can store 18 TB of data in one cartridge on a magnetic tape that is more than one kilometer long! The specification mentions that this capacity is for uncompressed data and the claim is that with compression, the capacity can be increased to 45TB. The tapes wounds around a single reel and once inserted into a tape drive, it can be read by spinning the reel. However, as one can imagine, moving from the start of the tape to the end takes a very long time.

Data on the tape is written in a number of tracks. For instance, LTO-9 specifies 8960 tracks; each track resembles a long line of data, from the beginning of the tape until the end of the tape. The tape-reader is equipped with a read/write head that can read or write data from multiple adjacent tracks. In LTO-9, this number is 32, meaning, one can read or write from 32 adjacent tracks. This set of 32 tracks is called a "wrap". Thus, the tape reader than read one wrap from beginning to the end in one pass. One optimization that has been employed in laying out the wraps is that adjacent wraps are written in reverse order, meaning, even wraps are laid out from the beginning of the tape until the end while the odd wraps are laid out from the end to the beginning. As a result, if the machine reads one wrap, it can start reading the next wrap without rolling the tape to the beginning.

For example, assume we insert an empty tape that is rolled to its start in the tape reader and we would like to fill it out with data. The head starts at wrap 0 and it can write tracks 0 to 31. We can write these tracks until we reach the end of the tape; wrap 0 is now fully written. To write wrap 1 which contains tracks 31 to 63, we can simply rewind the tape in reverse order while writing it simultaneously. When we reach the beginning of the tape, we have filled out wrap 1 and then we can continue writing wrap 2 in another forward pass. TLO-9 contains 8960 tracks or $\frac{6656}{32} = 280$ wraps and thus repeating this back-and-forth process 140 times will fill the tape. One pass can take close to 3 minutes at the maximum data transfer rate of up to 400 megabyte per second; according to the specs, it takes more than 10 (probably closer to 14) hours to write a full tape while operating at the maximum speed.

Question 1 (modelling): Discuss and come up with a theoretically simple model for the tape. Mention the parameters that your model uses. There is no unique answer here so you can do this in a few different ways. Recall that the distance of 1 km involved here is astronomical (in the context of computational systems).

Hints and remarks. There is absolutely no way you can come up with a simple model while keeping all the details and the specifications of the LTO-9 tape mentioned above. You can use the following hints and comments to help you in this process.

- Think of what makes "tapes" different. How are they different from a hard disk, a flash drive and so on. Try to come up with a model that captures it.
- We have two goals here: one, we would like to be able to design algorithms and analyze them and two, our analysis should hold reasonably well on a real tape.
- As a result, having a notion of "distance" reflected in our computational model should be very good idea.
- Some important things to capture in the model: assume the tape holds an input of size N. You also need to assume that the memory (i.e., amount of RAM) is limited (as otherwise, you can just read the entire input to RAM and solve it there); so assume a memory of size M. Beyond this, you will need to think and figure out what else makes sense to add to the model.
- You are allowed to change your model based on the upcoming tasks. For instance, if you come up with a model and you realize that it is too difficult to come up and analyze algorithms for the upcoming tasks, you are allowed to go back and change your model.

Question 2 (partitioning): Assume that the tape contains N numbers and that you are given a number p and you want to rearrange the numbers in the tape to place the numbers smaller than p before the numbers larger than p. Come up with an algorithm and analyze it in your model.

Question 3 (sorting): Next, repeat the same process as above but for the problem of sorting the array of N elements. As a hint, try to use quicksort and assume that picking a random pivot will ensure that the array is partitioned into two equal sizes. Note that I am expecting a short answer here!

Question 4 (the limitations): This is a more challenging problem. Rework the indivisibility model of computation we explored in the I/O model and come up with an argument that shows sorting (or permutation) problem is difficult.

Hint. First, think about what could be a very difficult instance to sort. Once you figure that out, consider the "distance" that the elements have to travel to be placed in their final destination. Then, come up with an argument that shows this is difficult and thus find a lower bound for the sorting problem.

Question 5 (two tapes): Show that partitioning (from the 2nd question) can be done much more efficiently if we assume we have two tapes at our disposal. In particular, assume that the first tape contains the N values, and the second tape is empty.

3 The Flash Memory Model

By today's standards, hard disks are considered an old technology and they are being replaced by Solid State Disks (SSDs), at least in the consumer product market. However, the internal structure of SSDs is very complicated, specially when it comes to how they deal with *write* operations and another curious aspect of them is the asymmetry between reads and writes; reads are generally much faster and "cheaper" than writes (see Figure 2)

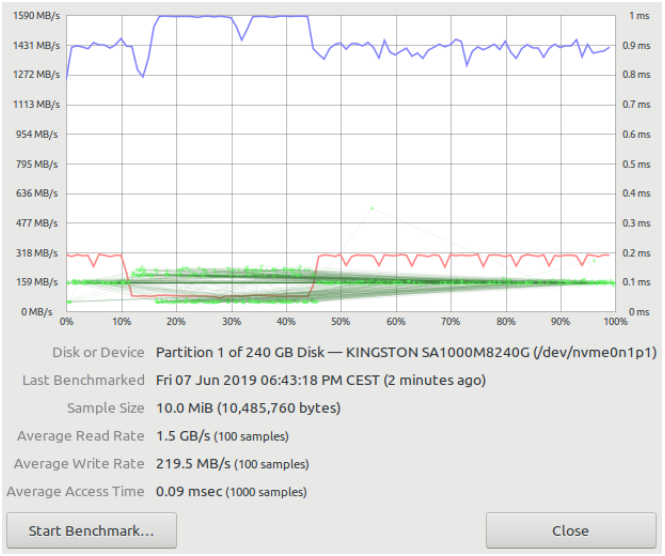

Figure 2: I used Gnome's disk benchmarking tool to test of my own SSD when I bought it in 2019. This is actually not very accurate but it shows the difference between reads and writes; later, I figured out a more accurate way of benchmarking them (see below).

Operation	Speed
Sequential Read	1500 MB/s
Sequential Write	750 MB/s
Random Read	5 MB/s
Random Write	0.08 MB/s

Table 1: My own (probably more accurate) benchmark of my SSD; the sequential Read/Write speed is what is being advertised by the producing company; they are also achieved only using very big block sizes (a few megabytes). The random read and write rows are measured using very small blocks (512 bytes) so it could actually be worse if we are accessing random bytes.

A very short summary is that while SSDs allow fast reads and writes, it is still true that accessing a *block* of elements is faster than accessing individual elements scattered all over the SSD (pay attention to the huge differences between sequential and random accesses in Table 1). Furthermore, *write* operations are much more costly than *read* operations and in fact *write* operations "wear down" the device; a major part of the internal complexity of SSDs is that they try to avoid writing the same physical location, to avoid hardware failures.

Modelling SSDs. Modelling SSDs is difficult and as with any real physical device, lots of details need to be ignored so that we can arrive at a theoretical model that is realistic and algorithmically workable. Here, we try to adapt the I/O model of computation to SSDs. Our replacement model is something like this: we assume the internal memory has size M, and we have a SSD that is divided into blocks of size B (some theoretical models even go further and consider different block sizes for read and writes; we keep things simple here). However, we would like to distinguish between reads and writes. This means that whenever we analyze an algorithm, we give one bound for the number of reads and another bound for the number of writes. And during the algorithm design process, our goal is mostly to minimize the *write* operations.

Question 6 (very write-optimized sorting): Consider an input S of N distinct numbers on an SDD disk. Show that it is possible to sort S using $O\left(\frac{N^2}{MB}\right)$ read I/Os such that every number is written exactly once (here we assume $N \geq MB$).

Question 7 (2-write sorting): Consider the problem from the previous part and additionally, assume that we are given a list L of $t = \sqrt{\frac{N}{B}}$ values, $\ell_1 < \cdots < \ell_t$ and we are guaranteed the following:

- ℓ_1 and ℓ_t are the minimum and the maximum elements of S, respectively.
- The number of elements of S that lie between ℓ_i and ℓ_{i+1} is $O(N/t)$.

Show that we can sort S using $O\left(\frac{N^{3/2}}{MB^{1/2}}\right)$ reads, where every element of S is written at most twice.

A remark. It is possible to generalize the idea shown above and reduce the number of reads by allowing the input elements to be written more times. If we allow the input elements to be written 3 times, we can reduce the exponent even more. The elements ℓ_i can also be picked randomly.

Question 8 (A write lower bound): The above two questions show that it is possible to reduce the number of writes by dramatically blowing up the number of reads. In this question, we will show that unfortunately this is unavoidable. Consider the indivisibility model and show that there exists a constant $\varepsilon > 0$, such that if the number of writes is at most $\varepsilon n \log_m n$ (recall that $n = \frac{N}{B}$ and $m = \frac{M}{B}$), then the number of reads must be $\Omega(N)$. In other words, reducing the number of writes by even a constant factor requires blowing up the number of reads by almost a B factor.

Comments. Look at the analysis of the permutation lower bound from the following angle. Think of the different choices that the algorithm has while trying to perform the permutation. Figure out how many choices does the algorithm have when it comes to a single read and compare it with the number of choices for a single write. Then allow for a different number of reads and writes. Also, you do not need to reproduce any steps of the proof that we have already done for the I/O lower bound.

Hints and other comments. To come up with your model of computation for the tape, you can talk to the other groups and you can brainstorm with as many people as you want. The last question should not have a too long of a calculations. If you feel that your calculations are taking a bit too long, then pay attention to the terms involved and try to simplify your bounds.