Computational Geometry: Theory and Experimentation (2023)
I/O Model Exercises

Peyman Afshani

Focus:

Seeing the impact of cache misses in practice
Solving Theoretical Exercises in the I/O model

1 A Programming/Testing Exercise for the I/O Model

Download `shuff.zip` from here. Note that you probably need to be logged into brightspace for the link to work. The zip file contains one C program and one Java program that basically do the same thing. If you want to run the C program in Windows, you will need to change the two functions used for measuring times before you can compile and run it.

The programs do the following. First, they allocate a large array whose size you will need to enter at the command line. You will in fact need to enter the logarithm of the size of the array. For example if you run it with parameter 28, it will allocate 1GB of data (2^{28} `int` values). Next, it will repeatedly do the following, in a `for` loop:

- In the i-th iteration of the `for` loop, it will partition the array into pieces of size 2^i.
- Then, it will uniformly and randomly shuffle each piece.
- Then, it will continue with the next iteration with $i + 1$.

Task 1. Theoretically explain the running time of each iteration of the loop. Assume that the size of the array is n.

Task 2. Compile and run the program for relatively large array size. Note that if you are running out of heap space, you can increase it with the `-Xmx` option followed by the amount of heap space you want (with a suffix "G" to denote it in gigabytes). For example, I had the following experience on my office computer:

```
> java Shuff 30
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at Shuff.main(Shuff.java:62)
> java -Xmx8G Shuff 30
logsize 30 divided into pieces of log size 4 took 2984.626894 Millis
...
```

So I could run the program by increasing the heap space to 8 Gigabytes.

Task 3. Collect and then plot the data. Does the theory explain the behavior you see? Can you explain it given what you know?

2 A Theoretical Question

Task 4. Consider a matrix A and a matrix B that are both in *row major ordering*. This means that the entries of each row are consecutive in memory. Assume that both matrices have size N^2 (i.e., they are $N \times N$ matrices). Analyze the ordinary method of matrix multiplication in the I/O mode.

For this question and the next, you can assume that $B < N$.

Task 5. Answer the same question but this time assume that B is in *column major ordering*.

See the Figure for a representation of these orderings.

Task 6. Assume a $\sqrt{N} \times \sqrt{N}$ input matrix is given in row major order. Describe an algorithm that turns it into a column major ordering. You want it to be faster than sorting! Specifically, show that when $M = B^2$, then this can be done in $O(N/B)$ I/Os but when $M = 2B$, then it can done in $O(\frac{N}{B} \log B)$ I/Os.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(a) A matrix in row major order.

1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35
6	12	18	24	30	36

(b) A matrix in column major order.

Figure 1: Matrix ordering. The consecutive numbers indicates the values that are consecutive in memory.

Task 7 (optional). Prove that when $M = 2B$, $\Omega(\frac{N}{B} \log B)$ I/Os are necessary to transpose a matrix (in the model where elements are considered atomic, a.k.a, "the indivisibility model"). Note that this could be a challenging problem.

You can use the following hints: First, show that you can assume that the algorithm works by loading 2 blocks, shuffling them in memory, then writing them back to the disk. This can be proven by taking any algorithm that does not work in this way and turning into another algorithm that does (by potentially blowing up the number of I/Os by a constant factor). Next, you will need to define a potential function that captures how far are we from the final result. If you can show that each step of the algorithm can only take us a small amount towards this goal, then we have a lower bound.

Task 8. Let T be a model of a terrain surface represented as a grid (two-dimensional array) of N height values (i.e., T is a $\sqrt{N} \times \sqrt{N}$ grid). And assume for convenience that all height values are distinct. Then the flow direction of a grid cell c is defined as the neighbor cell of c with the lowest height lower than c; if no neighbor cell of c has lower height than c, then c does not have a flow direction and is called a "sink". To model how water flows on the terrain, we assume that water on cell c flows to a neighbor of c following the flow direction of c; if c is a sink then water on c simply disappears. Now suppose we place one unit of water on each cell of T and let all water flow along flow directions until it disappears. The flow accumulation of a cell c is the total number of units of water that flows through c.

Design an $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ algorithm for computing the flow accumulation of each cell in a N cell grid terrain model, under the assumption that the model is given such that a cell in position i, j in the grid not only contains a height but also the height and position of its lowest downslope neighbor (if existing).

Hint. Use an I/O-efficient priority queue. Namely, use the fact that if you perform N operations on an I/O-efficient priority queue, then the total cost of those operations will be as claimed. This technique is a simple version of a technique known as the *time forward processing*. The techniques gets its name because intuitively, it sends the relevant information "forward" in time, using the queue, such that it can be retrieved at the appropriate time.