Ana Premovic
CMPT 225
December 4, 2022

My 2 submissions: Experiment 3 and 4

Bonus Submission: Experiment 1

Experiment 3

Implementations:

I used the clearest ones I could fine online.

Linked List: From GeeksForGeeks, a simple implementation of a singly linked list - https://www.geeksforgeeks.org/program-to-implement-singly-linked-list-in-c-using-class/

Unrolled Linked List: From CodingNinjas - https://www.codingninjas.com/codestudio/library/unrolled-linked-list

Size: I made the array, unrolled linked list, and linked list have **100,000 elements each**. This way, they are large enough to not fit completely in the caches.

Data (Example Trial)

Data Structure	Total Traversal Time (milliseconds)
Array	0.286
Unrolled Linked List	0.517
Linked List	0.603

Interpretation:

All my trials were similar. My findings correspond to what was expected. In class, we learned that since Linked List nodes are stored in "random" locations in memory, each access will be a cache miss, resulting in a longer traversal time. For arrays, as they are contiguous blocks of memory, there will be cache misses only at the start of every cache line. Unrolled Linked Lists act as a sort of middle ground between the two. They have nodes like a linked list, but far fewer in number. The Unrolled Linked List took about 2 times as long as the array, and the Linked List even longer, which is consistent with our in-class analysis.

Experiment 4

Implementations:

AVL Tree: Textbook author's code

B-Tree: From Programiz - https://www.programiz.com/dsa/b-tree

Size: I decided to make the number of elements even larger than the previous experiment, as the instructions state to use a "large set." The number of elements I chose is **700,000**.

Data (Over 30 Trials)

Data Structure	Average Access Time (microseconds)
B-Tree	0.0961794
AVL Tree	0.0754894

Interpretation:

I experienced a lot of wavering numbers; sometimes my B-Tree would be longer and sometimes my AVL Tree (even with the same elements). To mitigate the issue, I decided to calculate the average times over 30 trials.

Even though I executed my experiment carefully, there is some inconsistency with what was expected. Based on what we learned in class, a B-Tree should experience less cache misses than an AVL Tree, and therefore take a little less time. My best guess as to what happened has to do with the implementation of the B-Tree I used. Since the B-Tree is quite complicated, an attempt to code it myself in the little time given for this assignment would result in too many bugs. So, I spent time looking for implementations online. The best one I found is the one used, but it probably has a lot of inefficiencies compared to the Textbook Author's AVL Tree. Therefore, I believe the cache misses were executed correctly, but the inefficiencies in the B-Tree code balanced them out.

Bonus: Experiment 1

Sizes: To make sure the arrays could not fully fit in the caches, I created arrays of size **1,000,000**.

Implementation: To avoid the issue of hardware features that "pre-fetch" the next cache line into cache, I used the loop design recommended by David Mitchell in "Remarks on Each Experiment (1)."

Data (Example Trial)

Number of accesses	Total Loop Traversal Time (milliseconds)
1,000,000	315.666
1,000,000 / 16	18.287

Loop traversal	Average Array Access Time (milliseconds)
WITHOUT cache misses	0.00474272
WITH cache misses	0.000292592

Interpretation:

All my trials were similar. Though the experiment was carried out as carefully as possible, there are inconsistencies with what is expected. The times should be similar for each loop, because the number of

cache misses should be almost the same. I experienced difficulties with figuring out how to access the memory correctly to actually achieve the desired cache hits and misses.