Yurii Huba
SFU ID: 301540732
Instructor: David Mitchell
CMPT 225

Assignment 4

Experiment 1:

For this experiment I created 3 integer arrays with size of 5000000 elements. I added one more array to show that loop used to perform 16 multiplications takes significantly more time to proceed than just 16 multiplications in a row (which is an interesting observation I made while I was doing the assignment).

	Array with loop of 16 multiplications	Array with 16 multiplications	Array with 1 multiplication	Average "cache miss" access time	Average "in cache" access time
First try	4.231	3.139	1.278	1.278e-05	1.16312e-06
Second try	3.507	2.642	1.121	1.121e-05	9.50625e-07
Third try	5.41	3.099	1.49	1.49e-05	1.00563e-06

All measured time is in milliseconds.

The results of this experiment show that performing 16 times more multiplications and 'in cache' accesses of array elements causes only a slight increase in time of processing (about 1 or 2 milliseconds). Further investigation of average 'in cache' and 'cache miss' access time proved that most of the processing time of computers is spent on finding and copying variables to the higher levels of cache, while accessing variables that are already in cache lines 1 and 2 is comparatively fast.

Talking about increased time of processing loop compared to simple 16 multiplications, I assume that it is caused by need to spent additional time on processing loop iterator and instructions.

Experiment 3:

For this experiment I created unrolled linked list with 8000000 integers (node capacity is 10000 elements), simple linked list with 8000000 integers and integer array with the same size as previous two.

	Unrolled linked list	Linked list	Array
First try	18.267	26.216	16.456
Second try	17.651	28.258	16.104
Third try	16.541	28.36	15.392

All measured time is in milliseconds.

The results of the experiment show that traversing array or unrolled linked list is significantly faster than traversing simple linked list, while array traversal is faster than unrolled linked list traversal. The reason for this is that almost every access of linked list element causes cache miss, and processor spends long time finding this element and copying it to cache line. On the other hand, unrolled linked list allows to decrease number of cache misses as each node of it contains a large number of elements rather than 1, and processor can process these elements without moving to another memory location. Furthermore, array is fastest one to traverse because all its elements are stored in a sequential memory locations and processor can access them with minimum number of cache misses by recognizing the pattern of traversing.

Experiment 4:

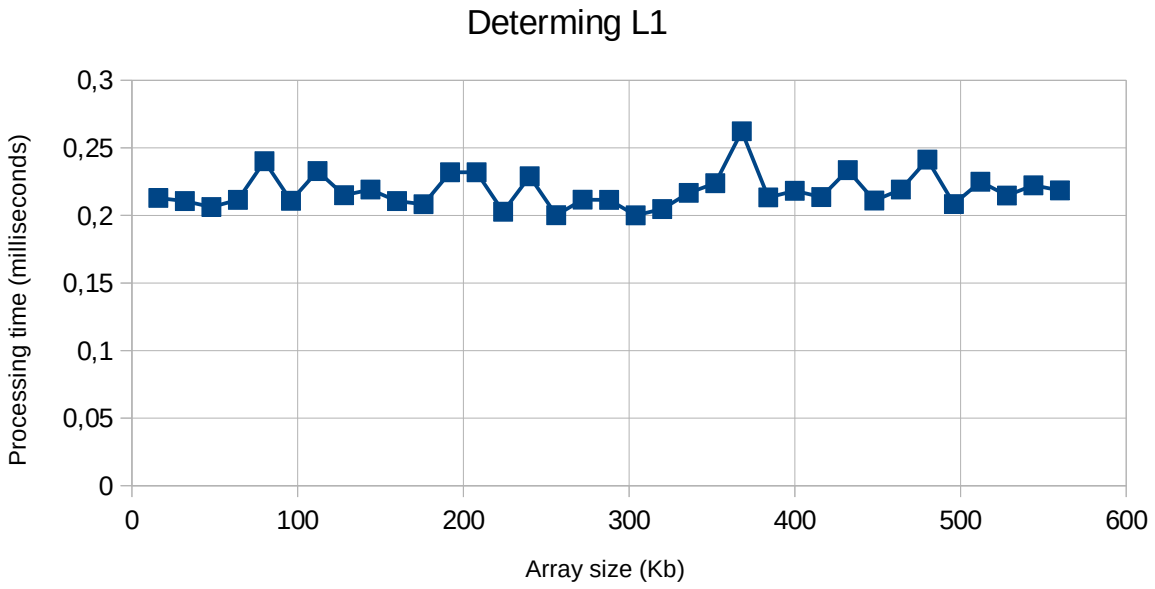
For these experiment I created B-tree (node size is 128) and AVL tree with 8000000 integers each. I used array of random integers to fill both trees and another array of random 3000000 integers to choose which elements that I will access.

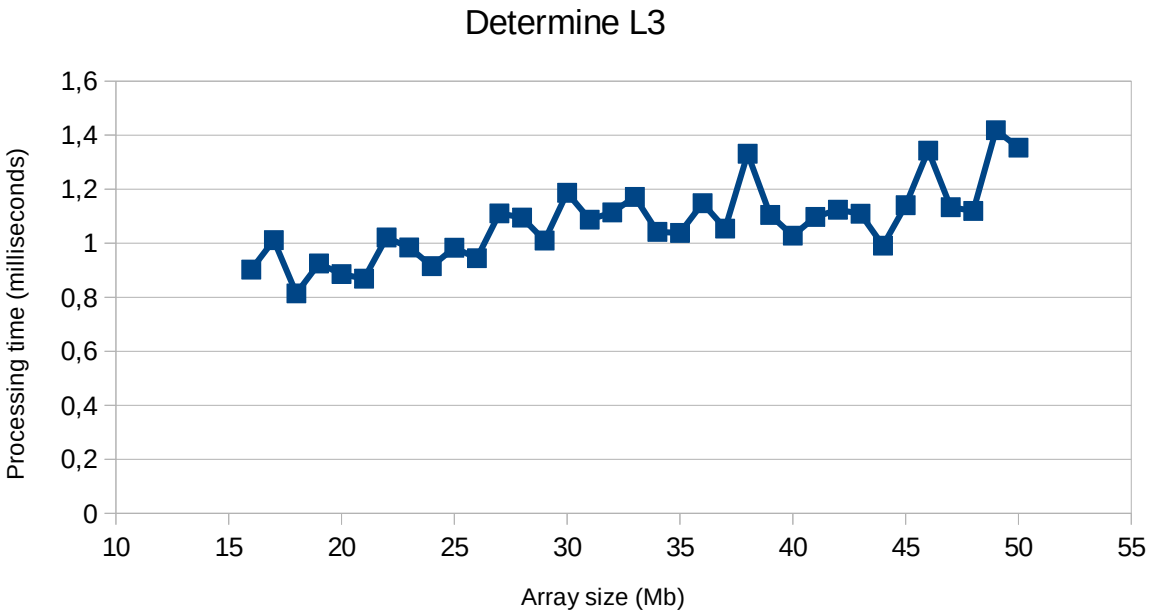
	AVL tree	B-tree
First try	3187.5	1993.4
Second try	3176.2	2019.55
Third try	3298.91	2228.56

All measured time is in milliseconds.

The results of this experiment show that searching elements of AVL tree takes significantly more time than searching elements of B-tree. The reason for that is that maximum height of AVL tree with n nodes is 2log(n), while maximum height of B-tree (node size of 128) with n nodes is $\log_{64}(n)$. Therefore, on average B-tree search in my experiment would be 12 times than AVL tree search, because of the reduced height of B-tree, which leads to decrease of B-tree traversing time compared to traversing time of AVL.

Experiment 2:





For the first part of this experiment I used integer arrays from 16Kb to 562Kb and for the second part I used arrays from 16Mb to 50Mb. For each part I used 35 different arrays increasing their size by 16Kb in the first part and 1Mb in the second part. To make sure that array contents are in the cache I run them 50 times and take average processing time as my data for each array size.

The result of the first part of this experiment (determining L1) shows that processing time didn't change significantly while I was increasing array size, and I think that the reason for that is that processors of 11th generation have very fast time accessing L2 level of cache, which is larger than L1 and it's very difficult to track the transition between these two levels of cache without having direct control over them.

The result of the second part o this experiment (determining L3) shows that processing time increases slightly when I make testing array larger and larger. The reason why we don't see a significant increase when L3 cache misses happen is again optimization of processors of 11th generation. I think, that predicting patterns of memory accessing allows new processors to minimize cache misses and stable processing time by a lot, so programmer doesn't notice such significant time difference when changing cache level as before.