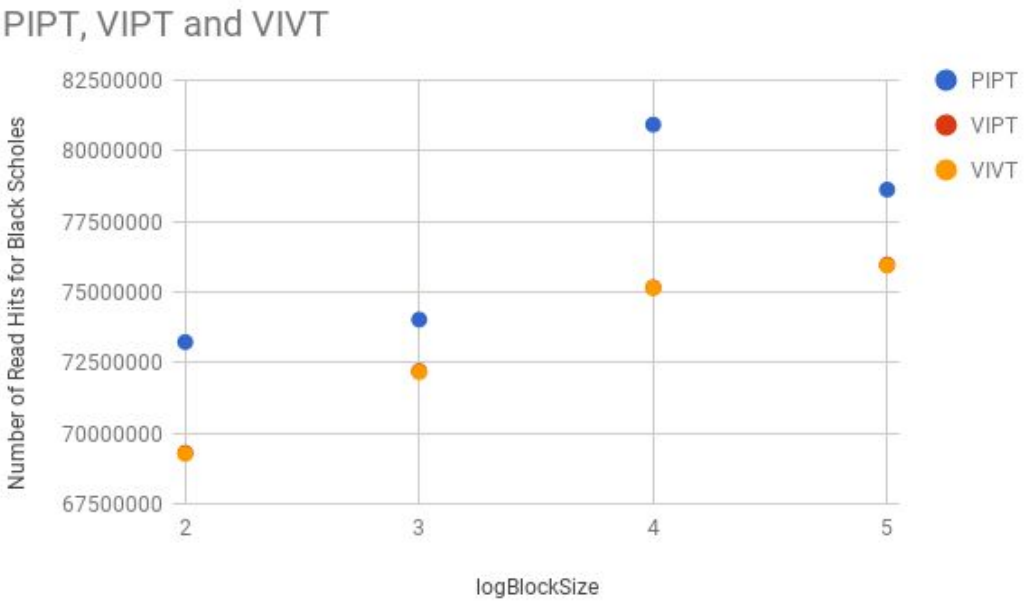
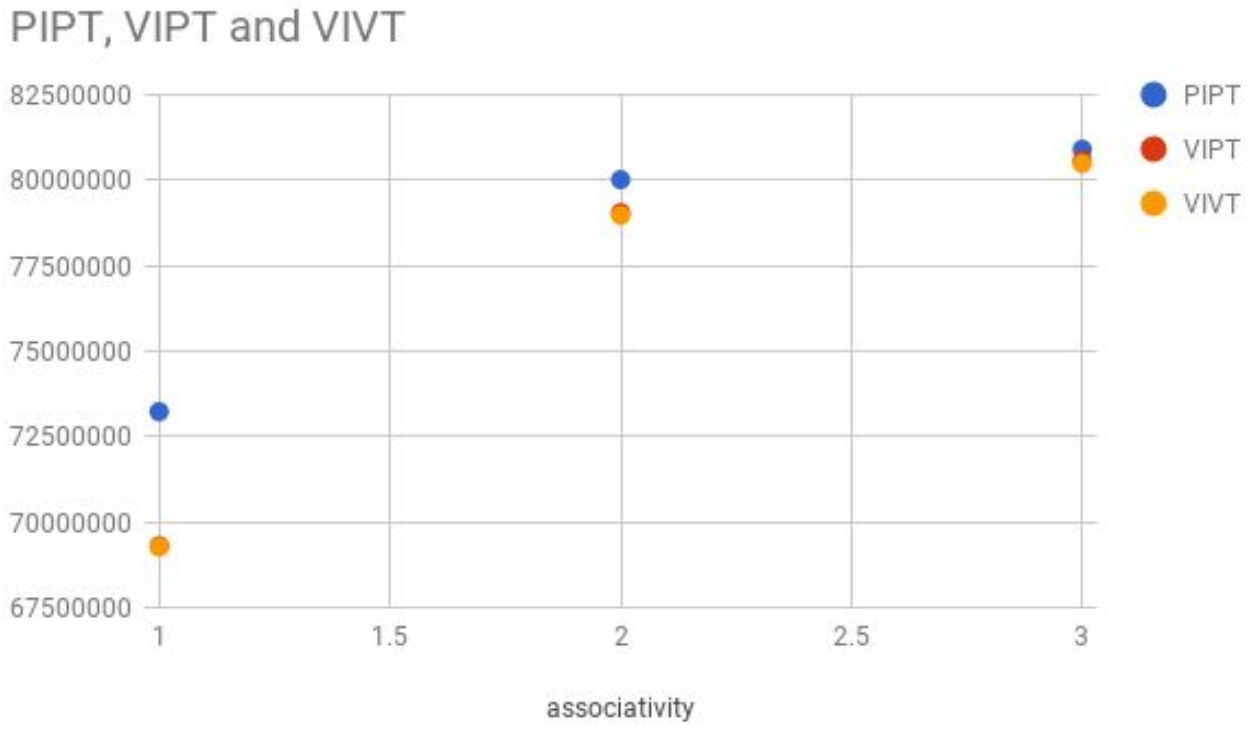
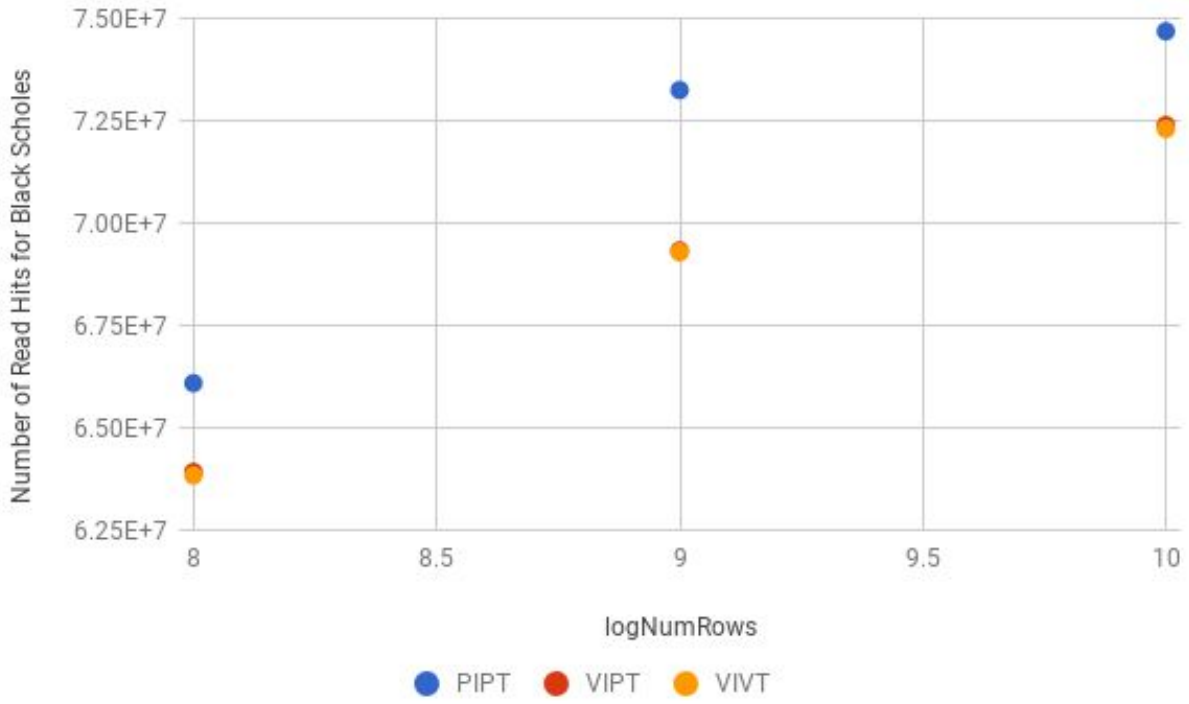
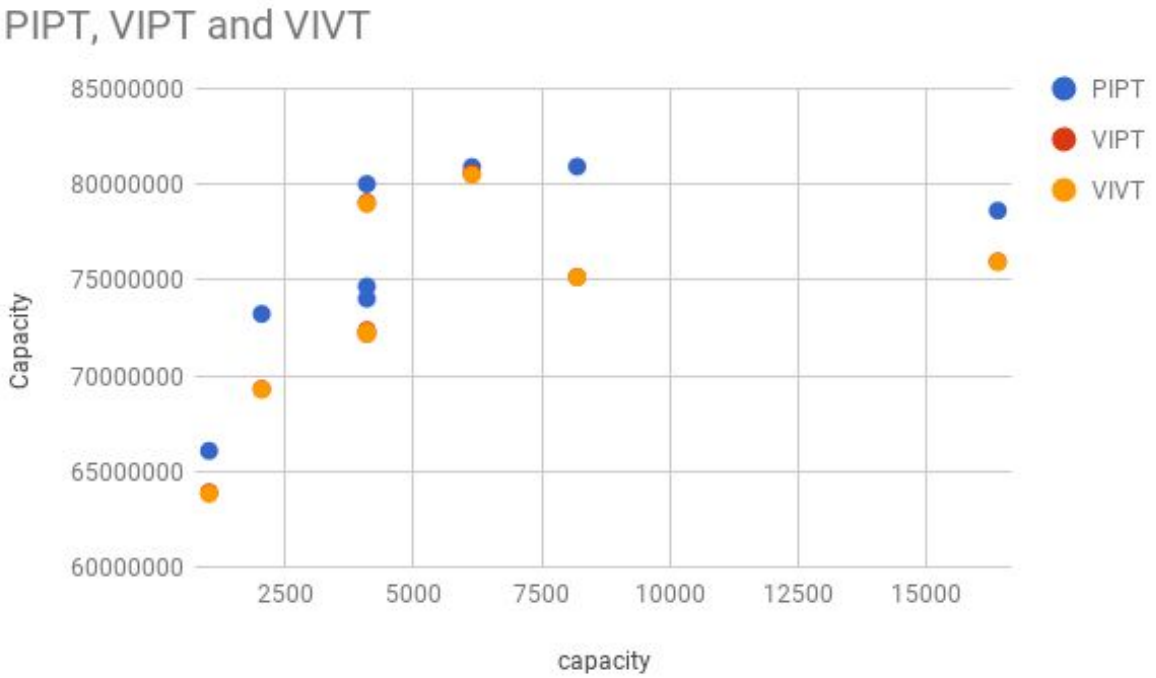
1. The following are charts demonstrate the number of read hits for the Black Scholes test when varying the logNumRows, logBlockSize, and associativity, and total capacity (NumRows * Block Size * associativity).









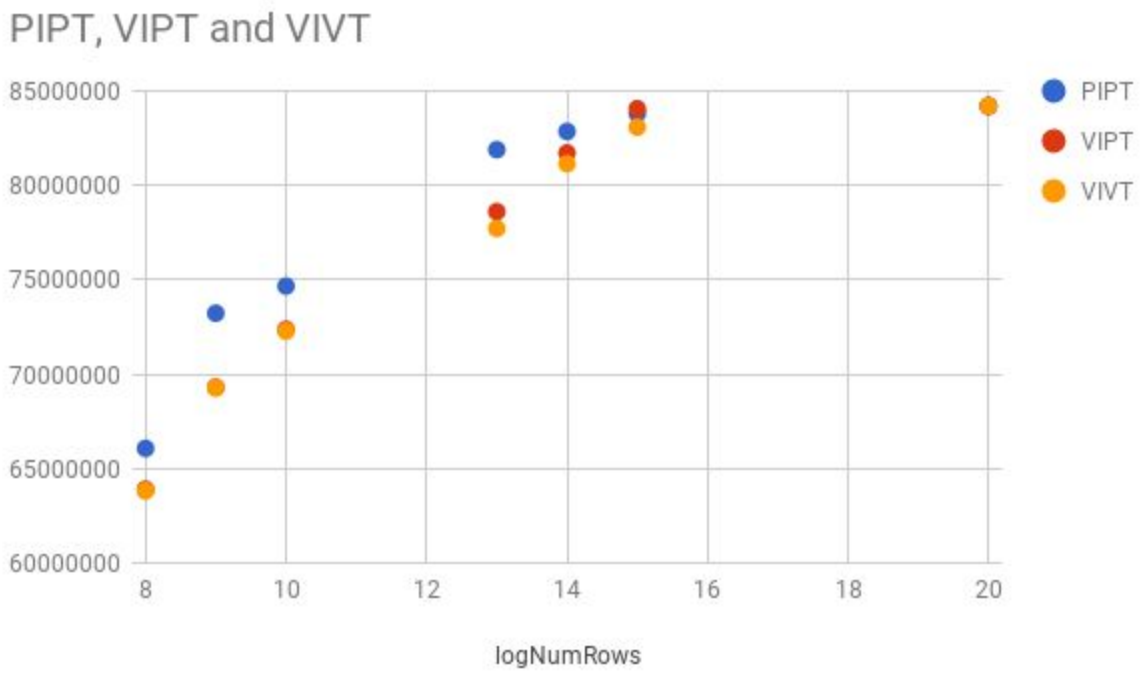
From the following charts, it seems that while increasing the logNumRows has a beneficial impact on hit rate to an extent, it does not increase the effectively as drastically as does increasing the associativity or changing the block size. This makes sense since if the capacity of

the physical memory is close the capacity of the cache size already, then expanding the cache size further should not yield as much improvement. The higher order bits would be more likely to be the same in most cases. However, increasing the block size allows for a larger section of data to be stored within each data block. This means that more data can fit within a particular index of the cache. The caveat is in the PIPT cache, where suddenly the hit amount drops for 16 byte data block. This could be a result of the way that TLB translation occurs.

PIPT has the best hit rate, while VIVT and VIPT are similar and much lower. The higher hit rate for PIPT can be explained in the fact that if the lower bits of the virtual address are very similar due to high spatial locality, while the physical memory is actually much more dispersed, then the distribution of data will be relatively even throughout the cache, while a virtual indexing might have clusters of data around particular indices.

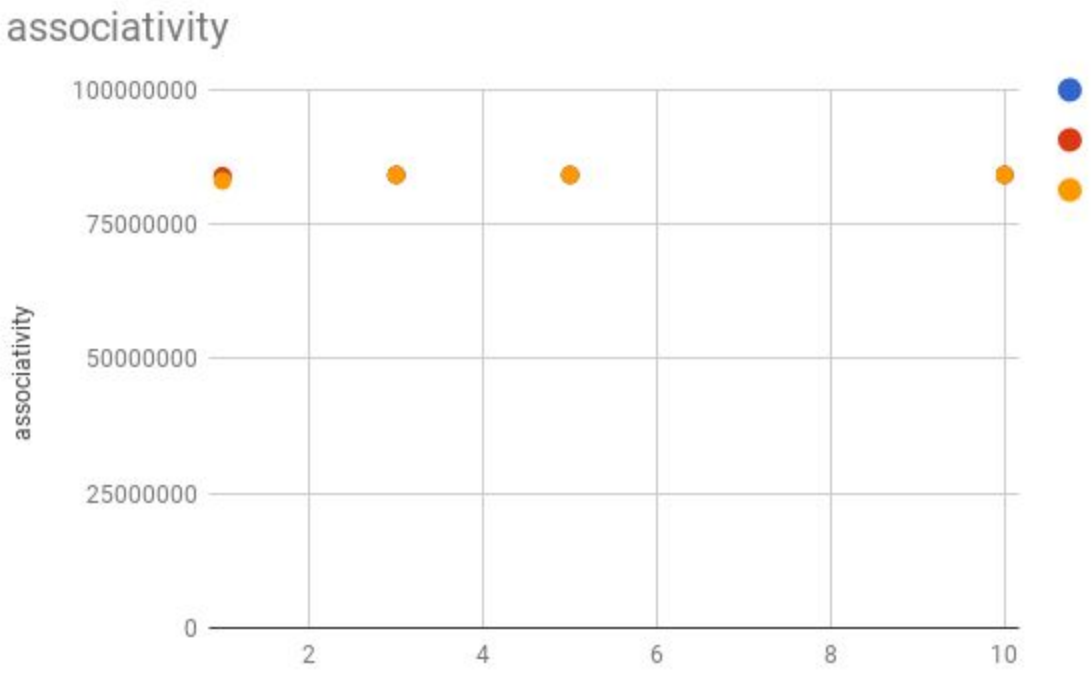
The reason virtual and physical tagging yield roughly the same results is that VIVT really only is an issue if there are multiple users or multiple programs accessing the same data using different virtual tags. Since the analysis only occurs within a single program, this issue does not come up.

2. To determine the size of the data, generally, we must manipulate the number of rows, associativity, and block size and see when increasing values does not yield significant hit rate improvements. First, we increase the number of rows until it stops yielding much improvement. This occurs at logNumRows = 15.

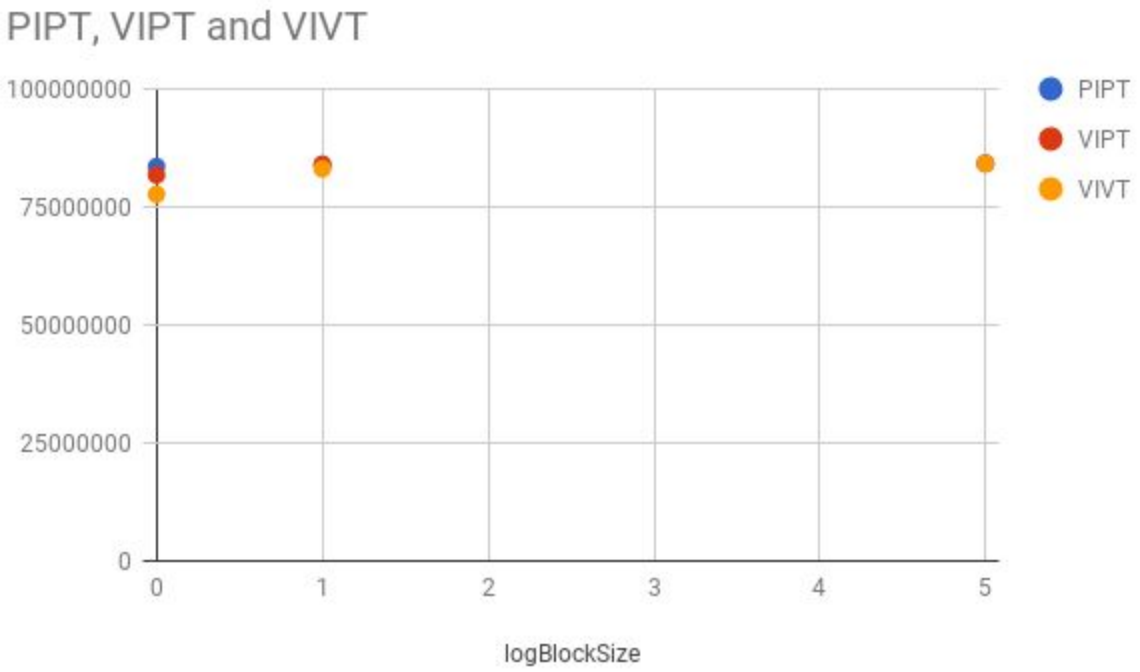


This suggests that the span of physical memory used is 2^15 pages. Second, we increase the block size multiplied by the associativity. By increasing the block size * associativity until we stop yield significant improvements to hit rate, we can get roughly the fraction of each page used. Generally, by decreasing block size and increasing the associativity, this will give us a finer grain measurement of the fraction of each page used.

When we fix the logNumRows to 15, changing associativity has no effect on the hit rate. Therefore, we can fix associativity at 1.



Furthermore, even when we increase logBlockSize, from 0, there is no significant change in the hit rate.



number of rows * (page / row) * (block size * associativity) = memory used per program.

2^15 * (1 * 1) = 2^15 bytes of data used = 3 MB.

Since this is a tedious process, I will test this over one more program. Bodytrack. Bodytrack seems to plateau in performance at 2^11 rows * 2^2 block size * 2 associatiativity = 2^14 bytes of data = 1.5 MB.

Now, we will fix this back to the base configuration and compare the hit rate (logNumRows = 9, logBlockSize = 2, associativity = 1).

For this base configuration, blackscholes get a hit rate of 86.6%, while bodytrack gets a hit rate of 71.3%. This makes intuitive sense since black scholes does not have many memory accesses that differ within the same page (notice that we only need a block size of 1 byte and associativity of 1 for black scholes) while bodytracking is optimal at a block size of 4 bytes and associativity of 2. Since the base configuration only has associativity 1, this better resembles the data accessed by black scholes than bodytrack. Although black scholes does span over a larger range of memory (2^15 vs 2^11) with only 2^9 rows in the cache, it seems this effect does not overtake the effect of within page accessing. In general, it seems as though if the data within the same page does not fit into a block, the hit rate will be lower. If the data is generally on different

pages, the hit rate will be higher, so long as there are enough indices to match the number of pages.

3. This configuration does not make sense. In order to have physical indexing, translation must occur before the access to the cache. This takes up time. Since we are already doing the translation, we might as well use the physical tag as well. The reason and VIPT exists is because we can index the cache simultaneously doing the translation and checking the tag.

4. For PIPT, there is no modification needed since for multiple programs, the physical index will be the same and they will replace each other. For VIPT, the tag will need to be compared to the other tags in the cache to make sure there is no overlap.

VIVT will not work for multiple programs since different virtual tags can translate to one physical tag. Overwriting one of them will not affect the other, causing issues in synchronization. To fix this, make sure that virtual addresses to the same location agree in index bits, causing direct conflict in the cache.

For a 256 byte page size and 512 cache size, supposing we have not implemented the indexing fix in software, the only cache configuration that can work is one with a single row. Any permutation block size and associativity such that block size * associativity = 512 will work (although a 512 block size is useless since the page size is only 256 bytes).

5. For Black Scholes, the number of accesses is 34632539, while unaligned accesses is 287291, making the unaligned access rate <1%. With such a low rate of unaligned accesses, the assumption makes sense. The explanation for the low rate of unaligned accesses could be compiler optimization. Because there is such a large cost with unaligned memory accesses, there is specific optimization to low rate.