TLB Simulator Performance Based on Changes to Flushing, Fnstruction Fount, Page and Table Size.

Nikita Yevtukh & Oleg Krushenskyy

Our TLB simulator will be running four different sorting programs. They are all sorting programs to make sure we have empirical data to draw conclusions from. Two of the programs have a space complexity of of 1 (Bubble-sort & Heapsort) and two with space complexity of n (Mergesort and Quicksort). This will help to do comparisons and because we are doing a TLB simulator we are more interested in the space complexity. The reason we are worried in space complexity is because the main purpose of the our TLB simulator is to see if the page references are in the TLB cache in the CPU. To explain lightly, if it's a hit (the page reference is in the cache), that's good and the program will run fast, if there is a page miss, then the page reference is not in the cache and must be loaded from memory which is a costly expense in terms of CPU cycles. What we are trying to showcase is that TLB with certain parameters can improve (reduce) the miss rate (missed page reference/ total page reference). These parameters are the policy of either FIFO, LRU which is a queue policy. Next is having different page sizes and table size. And lastly the TLB flush period, where it dumps and reloads the TLB cache every specified period (number of page references). The paper [1], textbook and wiki article were used to understand this assignment.

The list of programs are:
Program 1 - Mergesort - space: n
Program 2 - Quicksort - space: n
Program 3 - Heapsort - space: 1
Program 4 - Bubble Sort - space: 1

If we analyze only from the data where the inputs are changed we can clearly see from the (FIFO/LRU LRG vs MED vs SML) graphs that since mergesort and quicksort are in space complexity of n the TLB miss rate and the total references (from the excel file) steadily increase. This is due to there being a larger amount of elements to sort and there for more page references, in fact the page references seem to appear to increase by a magnitude of 10. This coincides with the fact that each sample size done is also increases by a magnitude of 10. The algorithms for each of these two sorts are fairly similar in terms that they are divide and conquer, but merge sort performs slightly better due to less swaps and comparisons. These two algorithms also do better at larger sample sizes then smaller ones as seend when comparing the LRG to SML graphs.

On the other hand looking and heapsort and bubble sort not only are the the slower algorithms but there space complexity is constant. The graphs FIFO MED and FIFO SML demonstrate this if you notice the miss rate does go up only slightly and on the excel sheet we see that the magnitude of page references only go up by 3. Though they might not be constant O(1) exactly the slight variation we see could be from the lab machine itself and its virtual memory management.

The reason bubble sort results are so scattered we think is due to the algorithm itself, since we are swapping so many variables so many times, the program needs so much space to store all the copies for swapping. Keep in mind that we also decreased the Med sample size of bubble sort to 1000 rather than 5000 like the others due to the shear amount of page references that need to be handles it slowed our simulator down too much. Where the small sample size was reaching an average of ~8.7 million page references and the medium sample size was reaching ~33.6 million page references which is roughly 3X more than heapsort at 5000.

Bubble sort then of course compared to the other sorting algorithms had a better (less) miss rate at small sample size and worse at 1000 sample size. This is due to bubble sort being very effeicent in sorting and space management for small samples but then grows quadratically as this paper supports [2]. Furthermore the response in the question asked on quora [3] supports our conclusion that the larger bubble sort the worse (more) miss rate is.

Based on the Policy changes alone, from the LRU vs FIFO graphs we can clearly see a trend that LRU does perform better than FIFO. We can also note that comparing the FIFOMED and LRUMED graph that increasing either page size or table size improves (lowers) the miss rate.

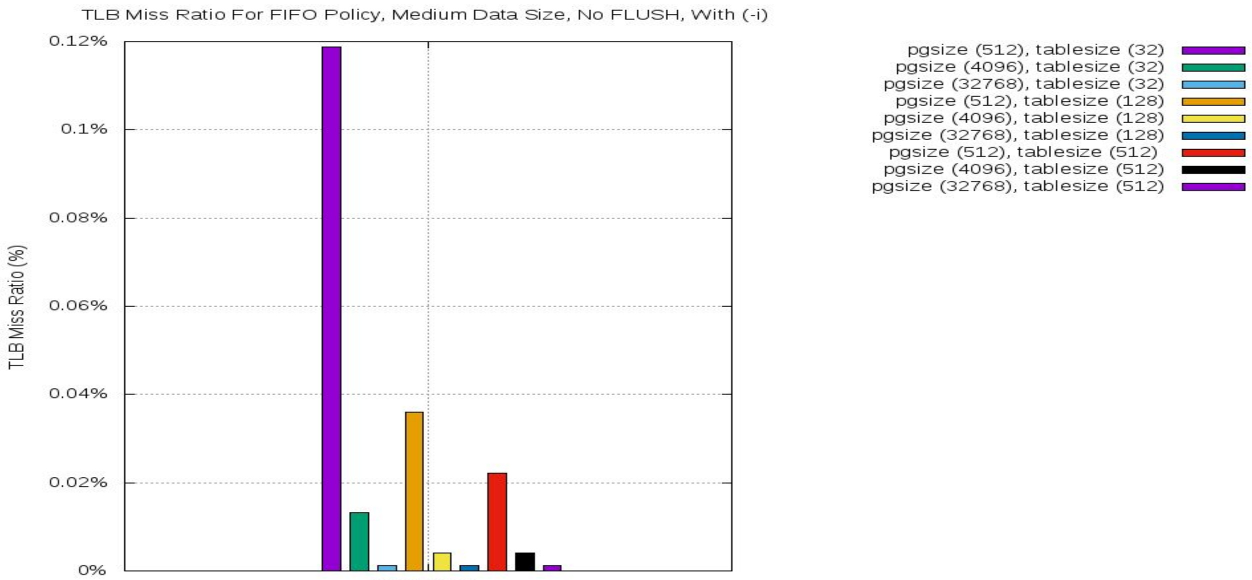
Based on changes to page size alone, from FIFO and LRU graphs small up to large sample size we can see the miss rate improves (lowers) with an increase to page size. This should be the case because we are essentially telling that the program now has a larger page per section and there for there will be a smaller miss rate. Its analogous to how if we made a bucket bigger for collecting rain drops from a leaky roof, it can hold more before it spills over, which are our misses.

Based on changes to table size alone, from FIFO and LRU graphs small up to large sample size we can see the miss rate improves (lowers) with an increase to table size. This should be the case because this allows for more pages being held by the cache and there for less page reference misses. Using the same analogy as above it's like increasing the room to fit more buckets.

Based of changes to both page and table size, we improve the miss rate but also hit a performance wall. This is made clear most of all on the excel sheet on any FIFO of LRU table we can see that as we get higher in both sizes, the misses stop changing about ar 32768 page size and 128 table size. Since we always miss the very first page references every time, this is probably the case were no matter how much more we increase both sizes we cannot lower miss rate.

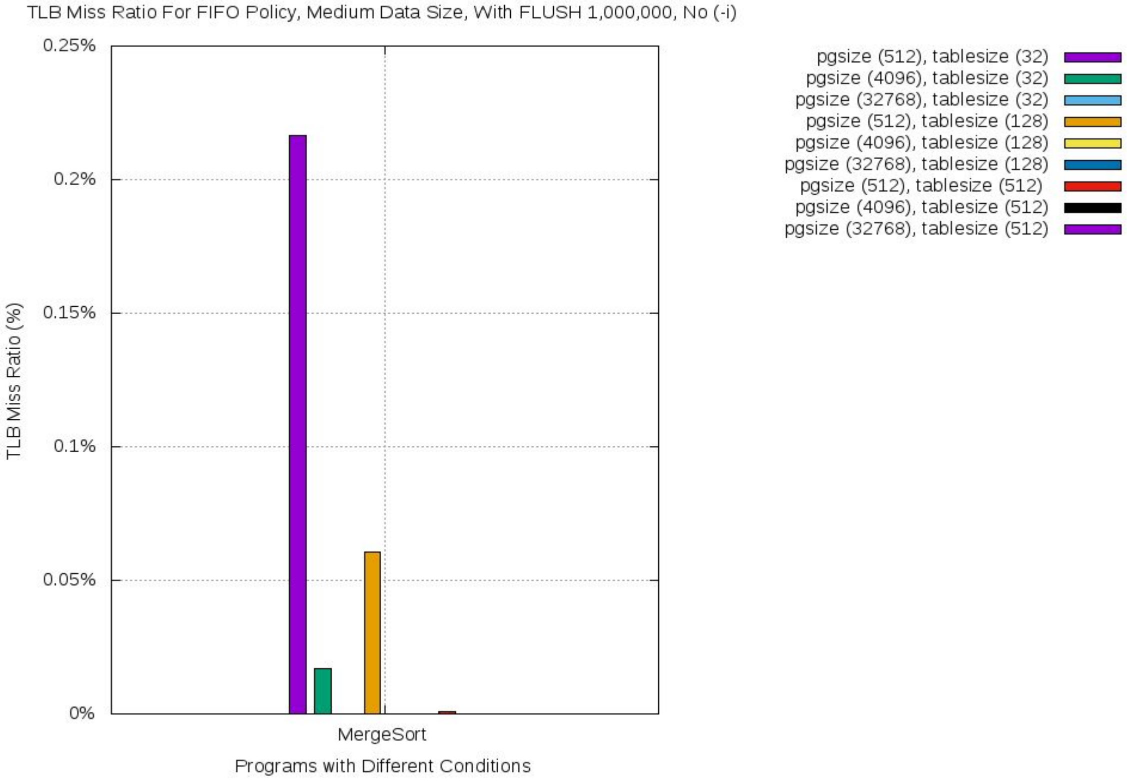
We can also note that in the large sample size there was no difference in LRU and FIFO at the performance wall. This can be clearly seen by examining the LRU of heapsort where at 32768 page size and 128 table size, it missed 47 and the FIFO heapsort at 32768 page size and 128 table size, was also 47. This concludes that there is also a performance wall for the policy type as well.

Looking at the -i option to exclude the we compare the graph tlb_i (below) to the FIFOMED graph.



Here we start to recognise that we get a higher miss rate in all variations to page size and table size. This is due to the fact that before, with no -i options we were dealing with ~1 million more page references as stated in the excel sheet. Therefore we would have more overall hits to misses and have a better miss rate. The conclusion we can draw is that this option best demonstrates the fact that it helps the TLB account for more costly operations suchs stores and loads that with this option gives a tester a more accurate profile of the performance of the program. We can also note that the variations contribute to the same factor of improvements with or without the -i.

Looking at the -f option to exclude the we compare the graph tlb_F (below) to the FIFOMED graph. Here we have more of an immediate impact to the miss rate from a flush period of 1,000,000. This means we completely clear the TLB every 1,000,000 page references. From the graph we can first tell that miss rate worsens (increases) in all variations then merge sort without flushing. Secondly we can clearly see that by increasing both the page and table size we improve (lower) miss rate. We determined that if you set a flush period in the same magnitude as the total page references it will worsen the miss rate. For example here the total page references without flush was ~6.1 million page references and so the flush period should be 1 million. For the results we can determine that having a flush period lower will worsen miss rate due to the TLB constantly removing it page references. Also having small page size and table size worsens with the flush period gives a worse performance of miss rate since the flush removes all the small page sizes and table sizes the TLB is trying to hold on to. On a final note, we noticed when running the TLB simulator with time in terminal, the flush slows down the simulator since it has to do extra deletions it didn't need to do before, however having a flush would be essential if you need to trade off miss rate for a TLB size reduction since your data structures holding the page references would be smaller in size.



In conclusion we can say that to overall increase performance of a TLB cache, we first must increase page size and increase table size. Using an LRU scheme increases performance. Having a large sample size and the -i period would help showcase the cache performance. Finally if you are going to have a flush to say save space or limit the data structure in the TLB the having a highest flush period will keep the TLB performance but a smaller flush period would save you space for the TLB.

References:

[1]: KONIREDDYGARI, SREEKANTH. Measuring and Improving TLB Performance for Linux GUI Applications. (Under the direction of Dr. Edward Gehringer and Dr. William Cohen). 09/04/2018. Webcite: https://repository.lib.ncsu.edu/bitstream/handle/1840.16/2072/etd.pdf?sequence=1&isAllowed=y

[2]: Ahmed M. Aliyu Dr. P. B. Zirra. Evaluation of Power Consumption of Modified Bubble, Quick and Radix Sort, Algorithm on the Dual Processor
https://pdfs.semanticscholar.org/47e8/13b1cbdfc50a7e1bfe21347aae9c499507f9.pdf

[3]:Which sorting algorithm will have a better cache performance: bubble sort or insertion sort?
https://www.quora.com/Which-sorting-algorithm-will-have-a-better-cache-performance-bubble-sort-or-insertion-sort