Annika Wiesinger

Charlie DiGiovanna

Tim Hung

22 April 2016

TLB Effectiveness Analysis

Many factors can affect the performance of a Virtual Memory system. The purpose of Program 1 is to simulate a Virtual Memory system to illustrate the performance of different factors that affect the system. Our group decided to analyze the effectiveness of a Translation Lookaside Buffer (TLB). We isolated the TLB as the independent variable in our program and maintained the page replacement policy as a constant. A TLB functions as a memory cache that stores recent address translations from virtual memory to physical addresses to speed up the system. Ideally, most memory references will be handled without having to access the page table in main memory because the TLB will hold the translation. The effectiveness of the TLB varies depending on the pages the program accesses. Our program compares the number of page faults when using a TLB and when not using a TLB for a varying percentage of repetition within the page references.

Although caching is a fundamental performance technique in computer systems, there is a point when a cache looses its effectiveness due to various constraints such as size. Therefore, analyzing TLB effectiveness is a relevant topic to find the threshold at which the TLB actually improves performance rather than costing a cumbersome overhead. Our research question therefore, is simply at what point is the TLB worth the overhead? Specifically, we started with a stream of no repeated references and gradually increased the amount of repetition to show the TLB’s impact on the number of page faults.

The first part of our program generates the output file that is piped into the second part of our program. The output file takes in the number of processes, a minimum and maximum address space size, and a page table size. These parameters are used to generate references for each process using the page table. The file that is output from this program is piped into the second executable. The program takes in each reference from the output file, checks if the reference is in the TLB first, and then checks the page table if the reference is not in the TLB. If the TLB is not full, the reference is added to the TLB. If the reference is not found in the page table also, it is either added to the page table or the page replacement algorithm is called and a page fault is generated. By checking the TLB first, the page table can be avoided for repeated references that are already in the TLB.

To generate data, we generate the output file from the first program with various inputs and pipe that file into the second program which outputs a file showing the results. One example of input for the first file is 5 processes with a minimum address space size of 4096, a maximum address space size of 8192 and a page table size of 16384. We expect that as the repetition increases the existence of the TLB will result in lower page faults. Below is an example of the final output.

-- Repetition at 0% --

NO TLB:

Num Page Faults: 54148

Elapsed Time: 1308.45ms

W/ TLB:

Num Page Faults: 27786

Elapsed Time: 1320.89ms

-- Repetition at 10% --

NO TLB:

Num Page Faults: 52389

Elapsed Time: 1396.89ms

W/ TLB:

Num Page Faults: 29539

Elapsed Time: 1384.9ms

-- Repetition at 20% --

NO TLB:

Num Page Faults: 51298

Elapsed Time: 1431ms

W/ TLB:

Num Page Faults: 30633

Elapsed Time: 1509.29ms

-- Repetition at 30% --

NO TLB:

Num Page Faults: 57593

Elapsed Time: 1097.54ms

W/ TLB:

Num Page Faults: 24390

Elapsed Time: 1175.35ms

-- Repetition at 40% --

NO TLB:

Num Page Faults: 55745

Elapsed Time: 1229.82ms

W/ TLB:

Num Page Faults: 26208

Elapsed Time: 1256.97ms

-- Repetition at 50% --

NO TLB:

Num Page Faults: 59171

Elapsed Time: 1102.84ms

W/ TLB:

Num Page Faults: 22847

Elapsed Time: 1013.82ms

-- Repetition at 60% --

NO TLB:

Num Page Faults: 60669

Elapsed Time: 1124.94ms

W/ TLB:

Num Page Faults: 21402

Elapsed Time: 1076.88ms

-- Repetition at 70% --

NO TLB:

Num Page Faults: 64135

Elapsed Time: 972.305ms

W/ TLB:

Num Page Faults: 18088

Elapsed Time: 1019.29ms

-- Repetition at 80% --

NO TLB:

Num Page Faults: 68667

Elapsed Time: 880.892ms

W/ TLB:

Num Page Faults: 13974

Elapsed Time: 817.516ms

-- Repetition at 90% --

NO TLB:

Num Page Faults: 74928

Elapsed Time: 564.681ms

W/ TLB:

Num Page Faults: 9362

Elapsed Time: 615.355ms

-- Repetition at 99% --

NO TLB:

Num Page Faults: 81086

Elapsed Time: 439.088ms

W/ TLB:

Num Page Faults: 23424

Elapsed Time: 484.779ms

As seen in the data above, the page fault rate is drastically lower when the TLB is used. Without a TLB, the number of page faults tends to increase as the repetition is increased. On the other hand, with the TLB the number of page faults tends to decrease as the repetition is increased. Although we expected the number of page faults to decrease as the TLB is used with increased repetition, we did not necessarily predict that the TLB would still be more effective when there is not repetition. Another observation of the results generated is that as repetition is increased, the program’s elapsed time gradually decreases. This demonstrates the effect of repetition on the program, there is less “work” to be done when references are repeated often.

The TLB assists in storing additional information so that memory does not have to be accessed with each reference, and suggests that the TLB will become more helpful as repetition increases. Based on our results, it seems the TLB is worth its overhead due to the large difference in number of page faults when there is not a TLB in use. There are other factors that can contribute to the effectiveness of the TLB, but our program focused on analyzing the page fault rate as a metric to demonstrate TLB effectiveness. As discussed in the book, it is not effective to increase TLB size as it then becomes slow. The TLB is meant to be a relatively small, fast cache to increase program performance. Our program attempted to simulate this in order to determine the value and effectiveness of the TLB in terms of page fault rate.