John Gangemi – U68714612

Raj Patel – U69174395

Project 2: Measuring the Size and Cost of Accessing a TLB

Overview:

The TLB (translation-lookaside buffer) is a hardware cache which is part of the MMU (memory management unit) and tasked with storing popular virtual to physical address translations.

For the TLB you can either have a hit or a miss. If you have a hit it means that the translation to convert from the virtual address to physical address is present in the TLB so you can go to memory and fetch the desired data. However if you get a miss it means that now you have to now go to the page table and find the relevant page which is stored to the TLB and then from there go to the memory. A cache miss is very costly in time since you now have to access the TLB and the page table.

Having stated that the purpose of this project is to measure the size of the TLB based upon the cost of a TLB miss after a finite number of pages were accessed.

We were able to determine that the size of the TLB for our machine (C4 lab computers) was 64 pages (assuming a 4K page size) based upon the following article we found online www.realworldtech.com/nehalem/8/ . To find the page size for the C4 machines we used the command getconf PAGESIZE which returned a page size of 4096 bytes.

Methods Tried:

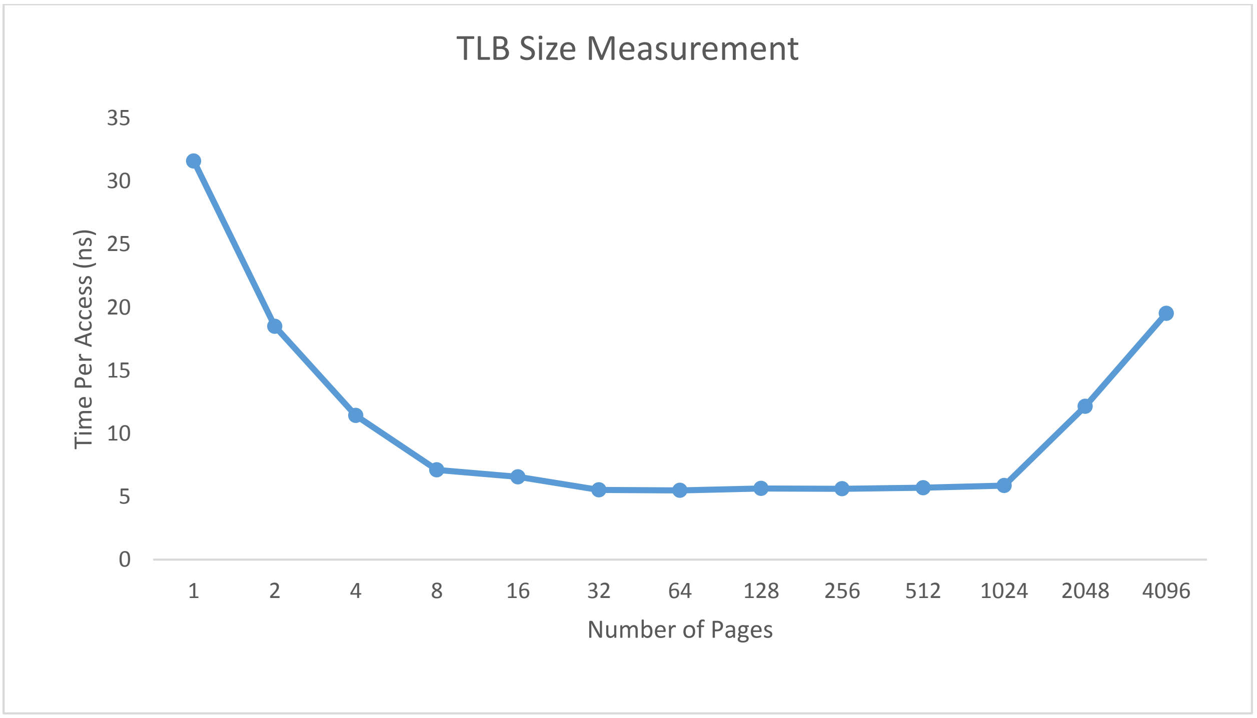
One of the main issues that we encountered was that we were getting negative values for the time. By using some print statements for debugging purposes we were able to figure out that we did the calculation for the time incorrectly. The original equation is shown below as equation (1) which we realized was incorrect since the end and start nanoseconds times need to be added in the equation not subtracted as shown in equation (2).

(1) nanoseconds = (end.tv_sec * conversion - end.tv_nsec) - (start.tv_sec * conversion - start.tv_nsec)

(2)nanoseconds = (end.tv_sec * conversion + end.tv_nsec) - (start.tv_sec * conversion + start.tv_nsec)

Another problem that we encountered is trying to figure out the most efficient way of calculating the time. The first method that we tried was gettimeofday().We tried placing the gettimeofday() in different places such as above and below the for loop, above and below the while loop and inside the for loop. However we realized that a monotonic clock is more precise than a clock that accounts for user time and system time so we switched over to clock_gettime() passing the declaration CLOCK_MONOTONIC as a function argument. We decided to place this function above and below the for loop.

Questions and Answers:

1) The getimeofday timer is not very precise since it measures in microseconds and we are trying to measure down to the nanoseconds.The interval of time that we are measuring in the for loop that was provided is not a sufficient span of time to get an accurate measurement of TLB accesses. A better measurement of time is clock_gettime() since it measures in nanoseconds and using the function argument CLOCK_MONOTONIC it ensured a more precise measurement as stated in the man page.
2) In order to get reliable results we decided to run 100,000 trials. We tried a hundred thousand, one million, ten million and also a hundred million and all of them gave us the same results so we decided to go with the smallest amount to save time when running the program.
3) 

4) In order to prevent the compiler from removing the main loop from the TLB size estimator we used a compiler flag (- O0) to ensure that compiler optimizations were disabled.
5) If you don't set the affinity mask for a process that uses critical time measurements to determine a specific aspect of performance, pertaining to an operating system, you could encounter a TLB flush since each core has its own TLB. If the OS switches cores for a process during run time a different TLB with different pages will be referenced thus giving invalid cache misses.
6) Other factors that can make the desired results inaccurate is low level hardware optimization for caches such as the principle of locality. Specifically special locality comes into play since when data is referenced more frequently i.e. an array in a large loop. Also there are other optimizations such as data access.
7) Using C4 lab machine 8 we attempted to predict the size of the TLB and the L2 cache based on the graph of generated data by our executable files. From our graph we can infer that our method of measurement is not precise until more than 4 pages are accessed. However it was noted that between 4 and 1024 page accesses an average access time was established leading

us to believe the first level access time is approximately 5.5 nanoseconds. After 1024 pages are accessed we believe that the L2 cache is being referenced as shown by the gradual increase in time access. Given our results we are inclined to believe the first level cache of the TLB can hold up to 1024 pages.

Conclusion:

The data generated by our executable does not match the data found online specifically from the source www.realworldtech.com/nehalem/8/. Reviewing the article we had expected to see a jump in access time after 64 pages for the first level of the TLB, however this was not the case. Therefore a logical flaw exists in our program's code or the complier/operating system is intervening in some unforeseen manner, well beyond the scope of this course.

Time of completion for this project was approximately 15 hours worst case. Most of our dilemmas evolved from inadequacies in the UNIX operating system standards. Similar to the first project we had to revert to the man pages for guidance on certain functions frequently. Also designing the graph to meet the visual requirements set forth in the project description document was rather difficult.

Thus we end this adventure in TLB land.