High Performance Computer Architecture
$[E0 - 243]$
Assignment 2

Minimizing TLB Misses Using Large Pages

Anshika Modi
M.Tech CSA
SR: 24-1-24095
anshikamodi@iisc.ac.in

October 20, 2024

Abstract

The Translation Lookaside Buffer (TLB) is a critical component in modern computer architectures, serving to speed up virtual-to-physical address translation. TLB hits plays an utmost role in improving the memory access latency. However, TLB misses can lead to significant performance bottlenecks, particularly in memory-intensive applications. This report focuses on optimizing TLB performance by mapping specific memory regions to 2MB large pages, thereby reducing the frequency of TLB misses.

I. Introduction

It is not often easy to find enough physical memory contiguity to allocate large/huge pages. Thus, it is important to be judicious while choosing the virtual address region that we want to map with large pages to minimize the number of TLB misses.
In this report, we have evaluated the given workload in the .so library provided whose memory access patterns are not uniformly distributed, by measuring the number of TLB misses across various virtual addresses using `perf mem` [1] tool and also after allocating huge pages of size 2MB to the same workload to see the effect on TLB misses, whether it can enhance the performance of the given program or not.

II. Methodology

A. Memory Access Data Collection

- The first step is to compile the given main.c file to get an executable `main` file which gives the starting and the ending address of the allocated virtual address and also the execution time of the program. This can be done using the command:

```
make run SRNO=24095
```

- To collect memory access data, the `perf mem` tool is used to sample the addresses of memory accesses and track TLB misses. The following command can be used to capture memory access information: [2]

```
perf mem record ./main 24095
```

  This records memory accesses and TLB misses during the execution of the workload by generating a file by the name of `perf.data`.

- This perf.data file is analyzed using perf report command and the analysis data is stored into a `.txt` file using the command:

```
perf mem report > mem_accesses.txt
```

B. Data Analysis

- The `mem_accesses.txt` file generated using `perf record` contained various fields like: overhead, memory accesses, virtual addresses, TLB acceses, etc. [3]for various instances of the workload. Our main focus is to analyse the number of TLB misses in the virtual address regions of 2MB.

- This collected memory traces in `mem_accesses.txt` was analyzed using a custom Python script `analyze.py` to identify the most frequently accessed virtual address regions and those experiencing the highest number of TLB misses. This python script can be run using the command:

```
python3 analyze.py 8
```

- The script takes the maximum number of large pages as an argument (n) and groups memory accesses into 2 MB virtual address regions to rank these addresses in decreasing order of the number of TLB misses experienced in the region and outputs the n most optimal regions that can be used to map to large pages to improve the performance (subsequently decreasing the number of TLB misses) in a file named as `largepages.txt`

C. Use of Large Pages

- The `main.c` file was modified to map the memory regions present in `largepages.txt` file to large pages of size 2 MB. The mmap() system call with the MAP_HUGETLB flag is used to allocate large pages in memory using the following code: [4]

  ```
  void *addr = mmap((void*)base_addresses[i], 2*1024*1024, PROT_READ | PROT_WRITE,
  MAP_PRIVATE | MAP_ANONYMOUS | MAP_FIXED| MAP_HUGETLB, -1, 0);
  ```

- After updating the `main.c` file to allocate huge pages to our workload, again run the file using `make run SRNO=24095` and again record and report the TLB misses using the `perf mem` tool in a second `mem_accesses2.txt` file.

- In this `mem_accesses2.txt` file, we can see in the 'TLB Access' field, the misses has been decreased.

III. Observation and Analysis

The `mem_accesses.txt` file was analyzed using a python script (as the file contained huge dataset) to generate the graph of the variation of number of TLB misses for the 2 MB virtual address regions. Figure 1 shows the distribution of TLB misses across different 2MB virtual address regions.

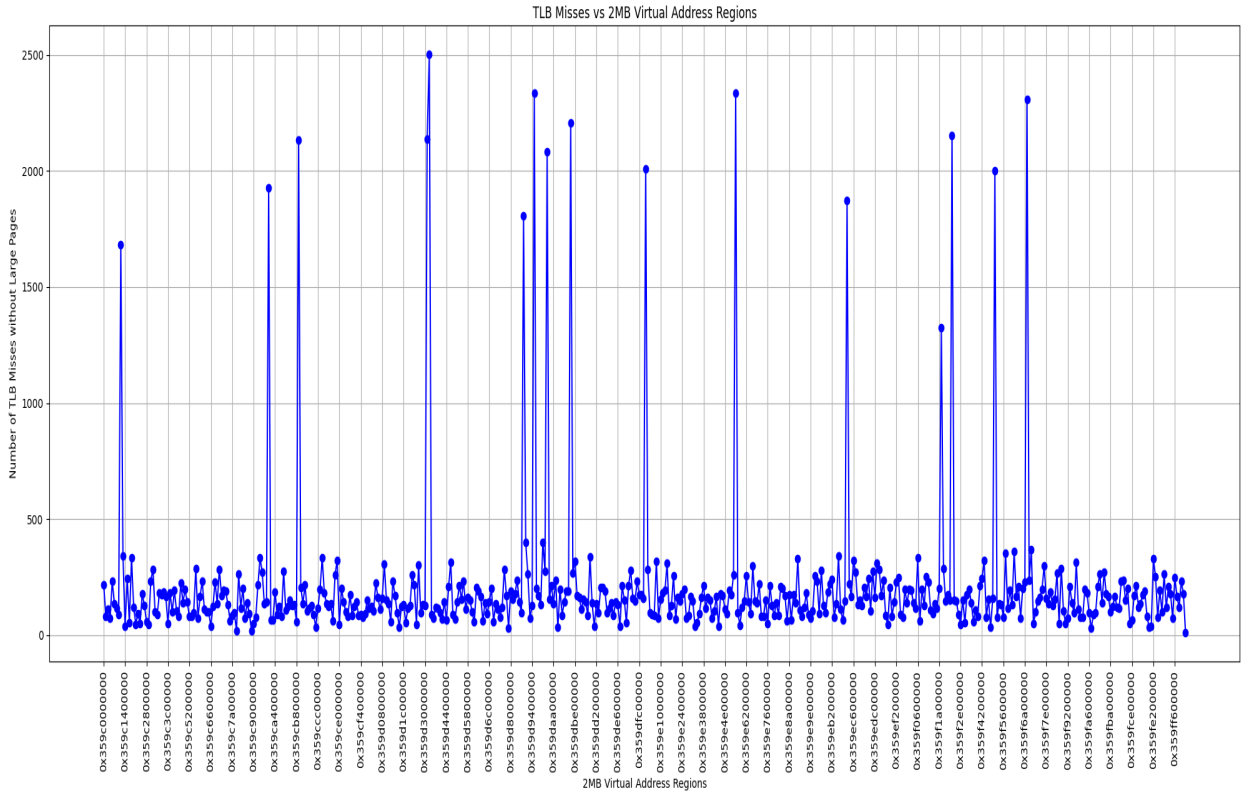

Figure 1: Variation of TLB misses with Virtual Address regions of 2MB

After allocating 8 huge pages to the most heavily accessed and most TLB misses faced regions identified by the analyze.py script in `largepages.txt` file, the TLB misses reduced significantly. Figure 2 shows the distribution of TLB misses across different 2MB virtual address regions after allocating huge pages of size 2MB.

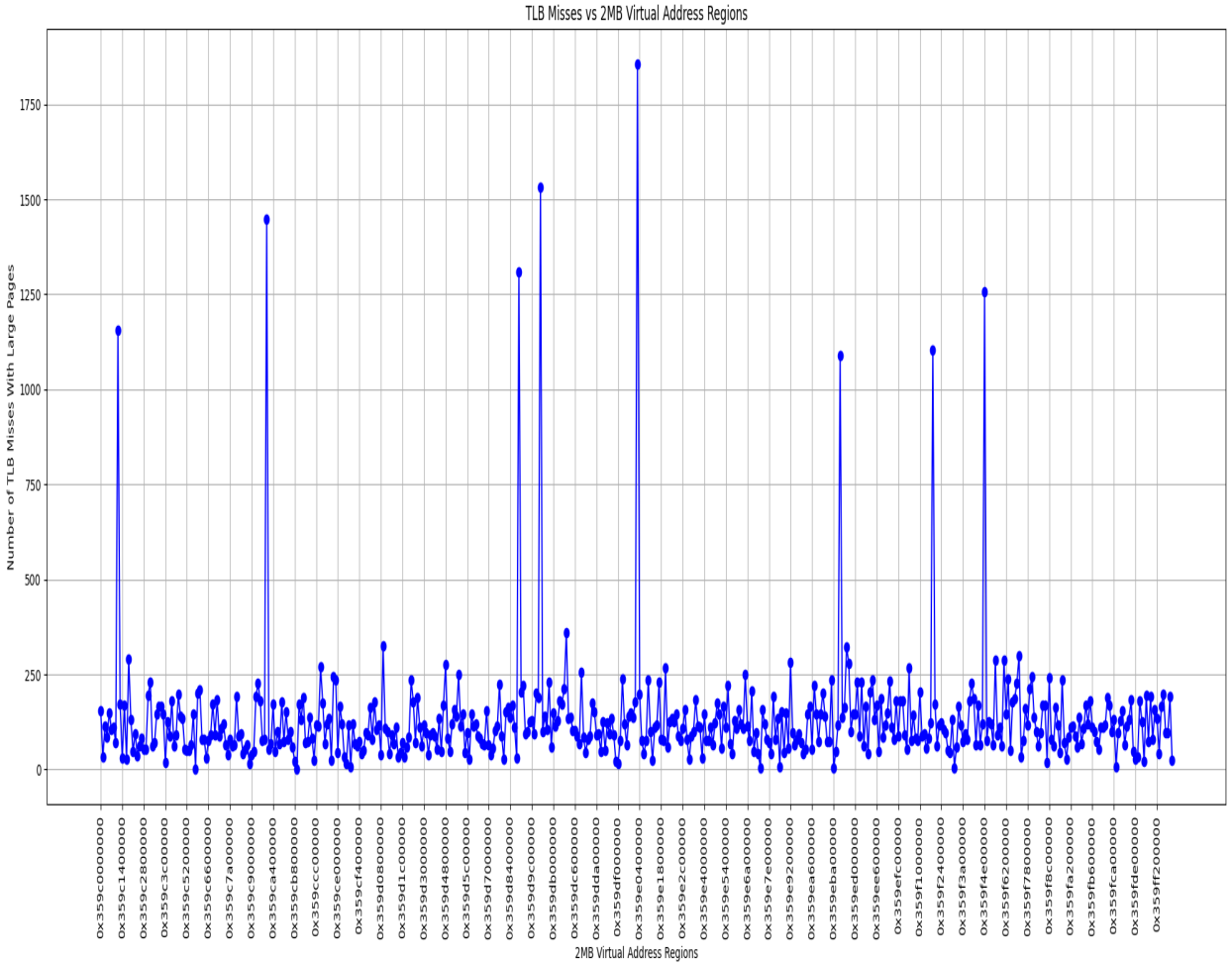

Figure 2: Variation of TLB misses with Virtual Address regions of 2MB with Large Pages

As seen from Figure 1 and Figure 2, Before using large pages, the workload experienced a high number of TLB misses in several memory regions. After analyzing the memory trace, we identified the top 8 regions with the most TLB misses and by mapping these regions to large pages, the TLB misses were significantly reduced, particularly in these regions with the highest memory access activity and frequent TLB misses. Figure 3 shows the comparison graph of both the cases ie, before and after the allocation of large pages.

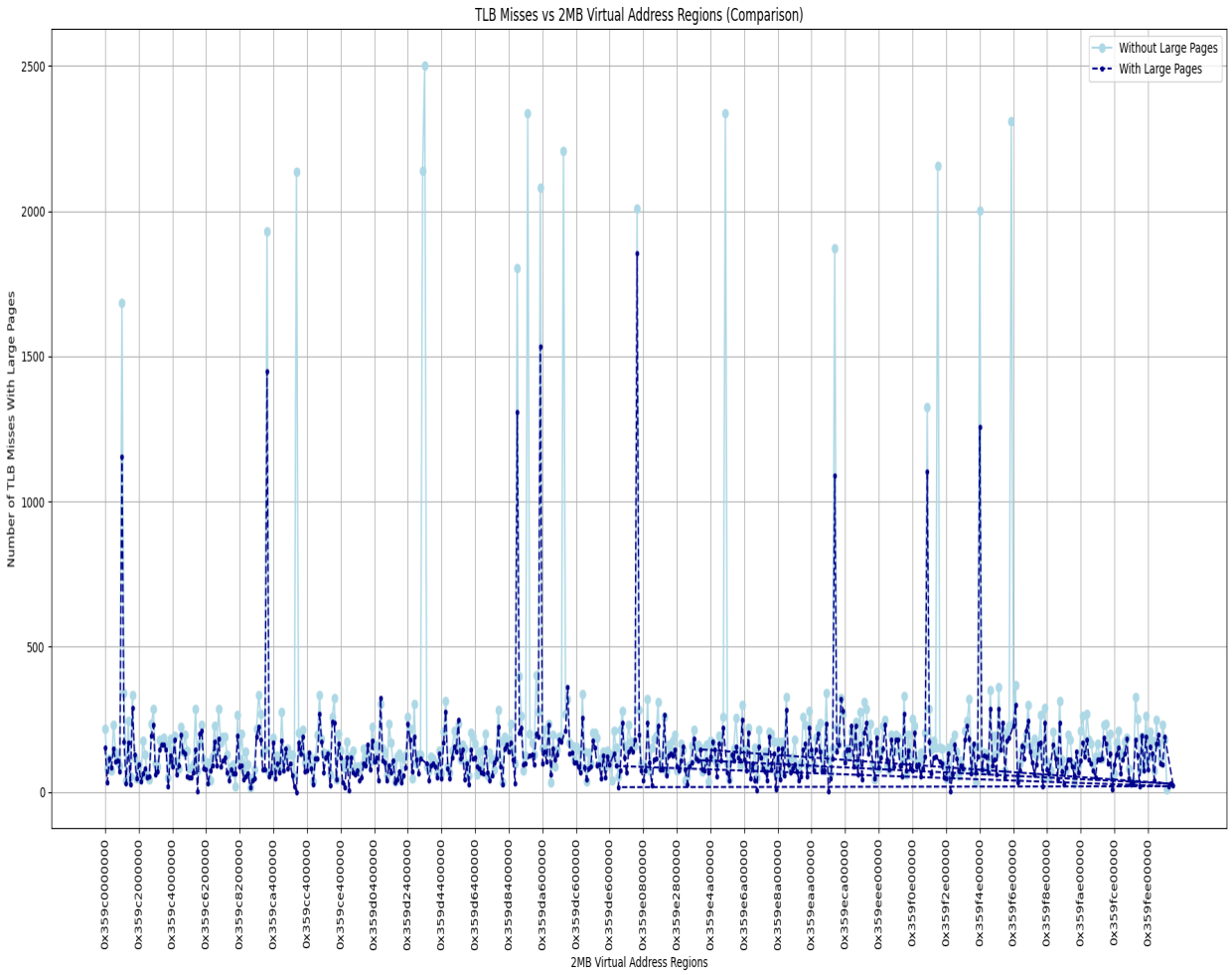

Figure 3: Effect of Large Pages on Number of TLB Misses

The program's performance in terms of their execution times was tested before and after deploying large pages. Table 1 summarizes the execution times for both the cases:

Table 1: Execution Times

Program	Time (in *sec*)
Without Large Pages	32.264690078
With Large Pages	20.121676454

The observed SpeedUp after using Large Pages is as follows:

$$SpeedUp = \frac{\texttt{Time\_Before}}{\texttt{Time\_After}} = \frac{32.264690078}{20.121676454} = 1.6034792206186 \quad (1)$$

Therefore,

$$SpeedUp \approx 1.6$$

The reduction in TLB misses directly contributed to this performance improvement, as fewer TLB misses result in fewer page table walks and reduced memory access latency and thereby speedup of greater than 1 is encountered.

IV. Conclusion

In this assignment, we successfully optimized the memory usage of the workload by identifying the most TLB-miss-prone virtual address regions and mapping them to large pages. This optimization resulted in a significant reduction in TLB misses and improved overall program performance, offering a speed up of 1.6 with large pages. This highlights the importance of memory management techniques, such as using large pages, to enhance the efficiency of memory-intensive workloads. Careful selection of virtual address regions for large pages allocation and mapping can enhance program performance for workloads that access memory in a non-uniform manner.

References

[1] URL: https://man7.org/linux/man-pages/man1/perf.1.html.

[2] RedHat. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/recording-and-analyzing-performance-profiles-with-perf_monitoring-and-managing-system-status-and-performance.

[3] RedHat. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/profiling-memory-accesses-with-perf-mem_monitoring-and-managing-system-status-and-performance.

[4] URL: https://www.man7.org/linux/man-pages/man2/mmap.2.html.