Evaluation of Matrix Multiplication Program using Perf

Anshika Modi
M.Tech CSA
Indian Institute of Science
Bangalore,India
SR: 24-1-24095
anshikamodi@iisc.ac.in

Archie Gaur
M.Tech CSA
Indian Institute of Science
Bangalore,India
SR: 24-1-24826
archiegaur@iisc.ac.in

***Abstract*—This paper evaluates the performance of the Matrix Multiplication Programme on three different Loop Orderings on Matrix sizes: 2048×2048 and 8192×8192 using the Performance Monitoring tool – PERF. This study mainly focuses on analyzing the performance based on the number of cache misses, TLB misses, and page faults as they play a major role in determining the overall performance. Various different optimization techniques like huge pages allocation and tiling is also applied to compare the performance metrics on both matrix sizes.**

***Index Terms*—PERF, tiling, cache misses, TLB misses, page faults**

I. Introduction

Perf is a Performance Analysis tool that is used to evaluate a system's performance by counting various events like TLB misses, data cache misses, page faults etc. It helps in sampling and profiling and also in identifying performance bottlenecks. In order to improve performance, allocating a large memory with the help of Huge Pages can possibly decrease the program execution time in some cases. Huge Pages is a Memory Management Technique that uses larger memory than the default page size to improve system performance[1]. They help reduce TLB misses by managing more memory with fewer TLB Entries. While using huge pages, internal and external fragmentation is reduced as large memory allocations are mostly continuous. mmap() is used to allocate huge pages. It is a function call that helps map a file to a process's address space. Malloc allocates memory from the heap in small chunks and uses the default page size of the system. Thus, cannot be used to allocate huge pages directly. Mmap() causes less internal and external fragmentation as compared to malloc because large memory allocations are generally continuous, unlike the allocation of small memory chunks using malloc.[2] Similarly,for optimization we have Tiling technique applied on matrix multiplication in which matrix operations are broken into smaller blocks called tiles to reduce cache misses and improve usage of cache, thus increasing overall performance.

II. Methodology

The main objective is to evaluate the Matrix Multiplication performance under various loop variants and focusing on the impact of optimization techniques such as allocating huge pages , Tiling(blocking). Using Perf Tool,following events are recorded to analyze the matrix multiplication performance with the help of command: "perf stat -e"[3].

A. DTLB Load Misses

Number of times the requested data address cannot be found in TLB.

B. L1-dTLB-misses

Number of misses in Level 1 DTLB Cache.

C. L1 Data Cache Load misses

Number of load misses in Level 1 cache.

D. L2-DTLB Misses

Number of misses in level 2 Translation look aside buffer. If miss occurs in L1 DTLB, then L2 Data Cache is searched for the content.

E. Last Level Cache (LLC) load misses

Number of load misses in the last-level cache.

F. L2-rqsts-all_demand_miss

Number of demand misses in Level 2 Cache.

G. Page Faults

Number of times a page is not found in the main memory(RAM).

These events are generated for all three loop orderings:(i,j,k), (k,i,j), (j,i,k) for both 2048 X 2048 and 8192 X 8192 matrix sizes.Their execution time is recorded and compared amongst the different loop variants of the Matrix Multiplication program.
Mmap() is used here to allocate large pages of size 2MB to determine the difference in their execution before and after using huge pages.. While using huge pages, internal and

external fragmentation is reduced as large memory allocations are mostly continuous. Similarly, the optimization technique of tiling is used to improve performance. It maximizes the number of cache hits by ordering the application data accesses. Performance metrics are evaluated for all the loop orderings for both the matrix sizes and the difference in execution time is measured.

III. Observation

For the 2048×2048 Matrix size, the (k, i,j) loop variant gives the best overall performance as it reduces cache misses and gives better execution time by optimizing row-wise access. In (k,i,j) loop variant,A[i][k], B[k][j] are accessed row-wise (contiguous memory) leading to better cache locality.
L1, L2 and L3 misses are lower than misses in other loop variants as the row-wise access ensures better utilization of cache blocks, reducing the need for fetching and replacing cache blocks from higher levels of memory. It also gives fewer page faults because rows are stored contiguously in memory and access is also carried out row-wise. (j,i,k) loop variant gives better performance than (i,j,k) but worse than (k,i,j) as in this loop order, B[k][j] is accessed row-wise but A[i][k] is accessed column-wise so cache misses reduce for B but increases for A.
(i,j,k) gives the worst performance as matrix B is accessed column-wise which results in higher Cache misses due to frequent replacement of cache blocks and higher TLB misses due to more page translations. More page faults because each column access in B will require loading a new page.
Similarly for matrix size 8192 X 8192: (k,i,j) gives best performance in terms of minimizing page faults and memory accesses and (i,j,k) gives the worst performance.
Figure 1 and Figure 2 shows the various parametric comparison given by perf for all the three loop variants for matrix 2048×2048 and 8192×8192 respectively.

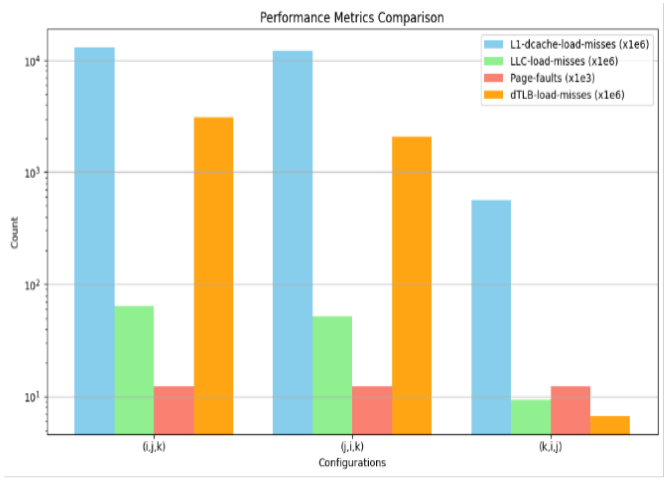

Fig. 1. Performance Metrics for 2048×2048 with 4KB Page Size

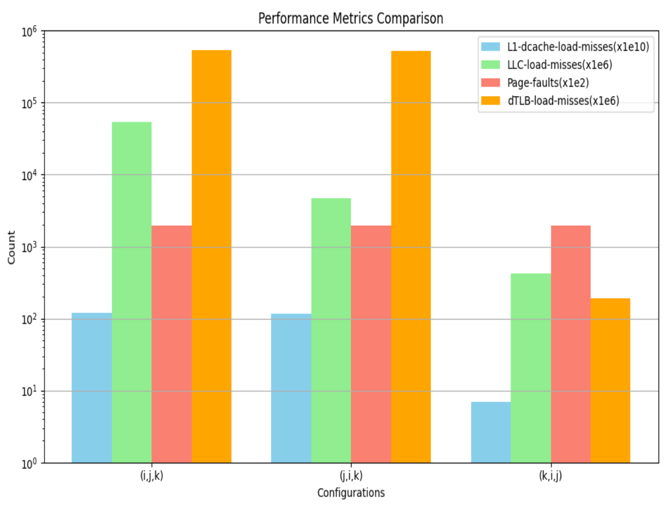

Fig. 2. Performance Metrics for 8192×8192 with 4KB Page Size

A. Effect of Huge Pages

For Both 2048×2048 and 8192×8192 matrix sizes, allocating huge pages with mmap() leads to significant decrease in L1 data cache load misses reduces for all configurations, but LLC load misses increases for (i,j,k) and (j,i,k). This could happen because at LLC level , the larger page sizes could cause more contention.
For (k,i,j) Configuration , LLC load misses reduces. Page faults and TLB misses are drastically reduced for all configurations after allocating huge pages.

Figure 3 and Figure 4 shows the various parametric comparison given by perf for all the three loop variants for matrix 2048×2048 and 8192×8192 respectively using huge page size of 2MB.

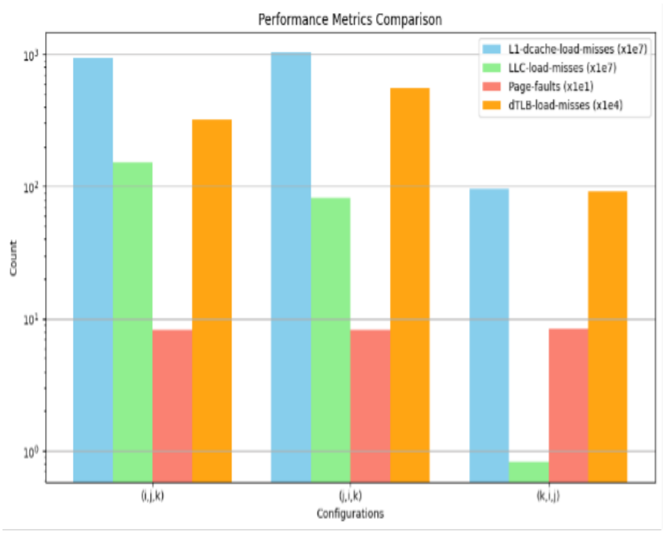

Fig. 3. Performance Metrics for 2048×2048 with 2MB Page Size

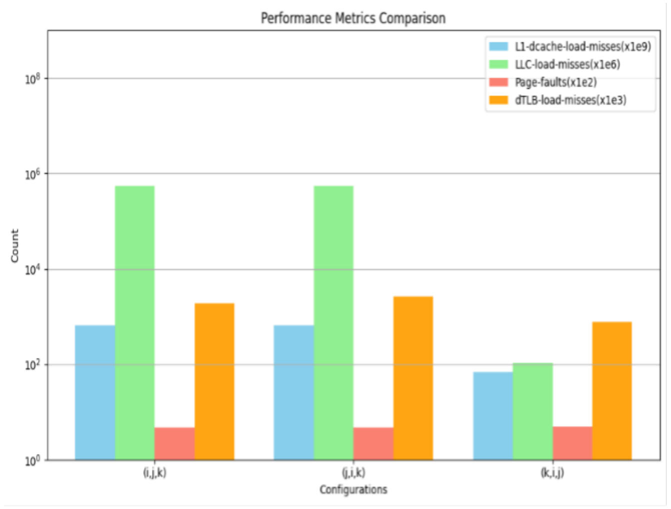

Fig. 4. Performance Metrics for 8192 × 8192 with 2MB Page Size

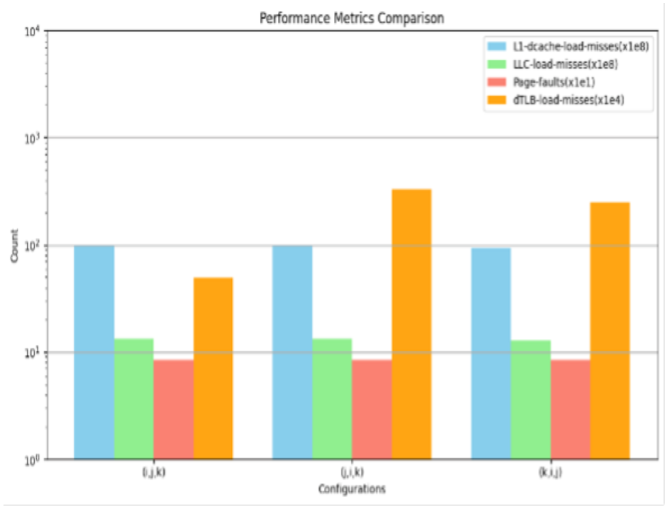

Fig. 6. Performance Metrics for 8192 × 8192 with tiling.

B. *Effect of Tiling*

L1 and LLC cache misses are reduced significantly for (i,j,k) configuration. Page faults and TLB misses are reduced for both matrix sizes. The major impact of Tiling is seen in 8192 x 8192 size because large size matrices cause inefficient memory accesses and cache contention so, Tiling helps in improving memory access patterns and minimizing cache block replacements by converting matrix computations to smaller tiles that are able to fit in CPU's cache. Thus, Tiling improves performance by minimizing page faults, cache overheads especially in case of bigger matrices.

Figure 5 and Figure 6 shows the various parametric comparison given by perf for all the three loop variants for matrix 2048 × 2048 and 8192 × 8192 respectively.

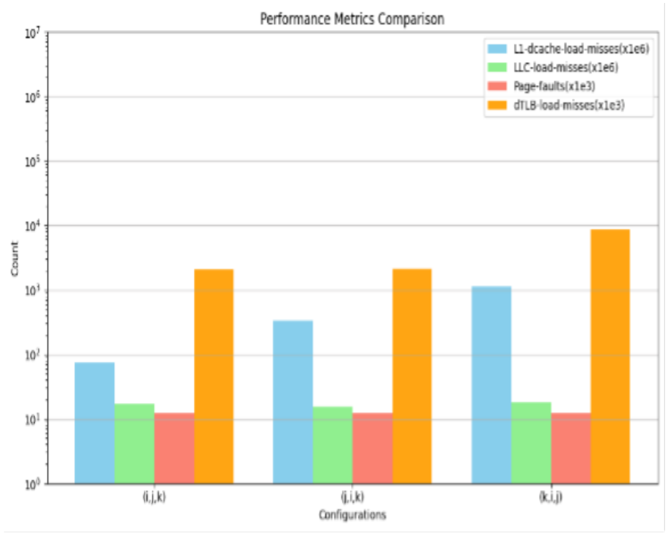

Fig. 5. Performance Metrics for 2048 × 2048 with tiling.

IV. Conclusion

We have used PERF to compare the various loop orders of matrix multiplication for different page sizes like 2MB and 4KB and in all the configurations, the (k,i,j) configuration gives the best performance in all the applied techniques, be it using page size of 4KB or 2MB or by using the tiled version of program.
The combined version of tiling and huge pages is a good optimization technique especially for large size matrices as it improves memory access patterns reducing cache misses, TLB misses and page faults significantly.
The use of tiling for larger matrices leads to significant improvement in matrix multiplication performance and huge pages minimize page faults and provide better TLB efficiency.

References

[1] R. Krishnakumar. *HugeTLB - Large Page Support in the Linux Kernel*. URL: https://linuxgazette.net/155/krishnakumar.html.
[2] Satyadeep Ashwathnarayana. *Understanding Huge Pages*. URL: https://www.netdata.cloud/blog/understanding-huge-pages/.
[3] *Perf tutorial*. URL: https://perf.wiki.kernel.org/index.php/Tutorial.