DSRACSIS: GEMM

1. assignment

Luka Šveigl, ls6727@student.uni-lj.si, 63200301

I. Introduction

As part of the first assignment of our Computer systems course, we had to implement a blocked version of General Matrix Multiplication (GEMM) and evaluate it on the Slurm supercomputer. In this report, we describe the implementation, before presenting the results of the evaluation. We conclude the report with a discussion and interpretation of the results.

II. Methodology

Our implementation of the Blocked General Matrix Multiplication (GEMM) began with a basic blocked approach. Initially, we iterated through matrices in blocks without leveraging caching mechanisms, leading to suboptimal bandwidth utilization. To address this, we transitioned to a caching and tiling approach, as suggested in existing literature [1].

In our optimized approach, we pre-allocated memory for blocks of matrices A and B in temporary variables before commencing computations. This pre-allocation resulted in minor speed improvements. Additionally, we allocated memory for a temporary block within the destination matrix. This temporary block served as a buffer for intermediate computations before being written to the actual destination matrix. This method minimized unnecessary memory accesses and enhanced computational efficiency.

III. Results

To evaluate the performance of our implementation, we executed the algorithm ten times and averaged the resulting bandwidths in order to reduce the impact of outlier results. Average bandwidths based on the **kc** parameter are present in Figure 1, where each of the plots represents a combination of the **mc** and **nc** parameters.

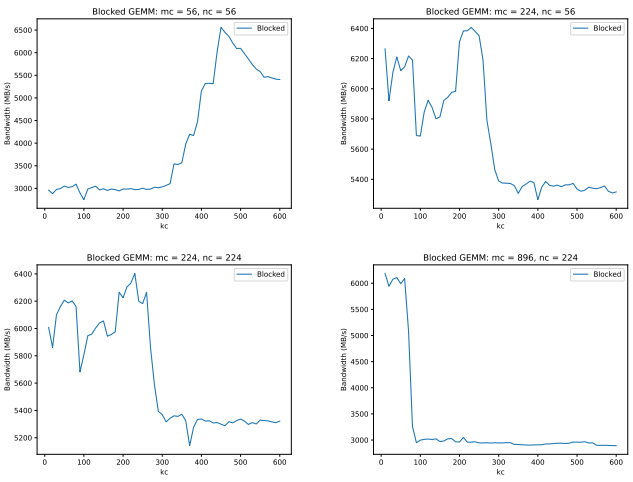

Fig. 1: Results of the Blocked GEMM implementation.

As we can observe in Figure 1, in the case where both $mc = 56$ and $nc = 56$, the bandwidth consistently improves as the kc parameter increases, meaning bandwidth increases as the size of the matrix increases. However, we can observe a breaking point at around $kc = 450$ where the measured bandwidth starts decreasing, which we can attribute to the cache not being filled optimally and/or the amount of data being too much for the cache. We can observe similar results in cases where $mc = 224$, $nc = 56$ and $mc = 224$, $nc = 224$. The only outlier here is the case where $mc = 896$, $nc = 224$, where the bandwidth maxes out at the beginning, before dropping off massively, which points to the fact that the blocks are too big to optimally fill the cache as the size of the matrix increases.

In Figure 2 we present the comparisons between the naive and blocked GEMM implementation. As we can observe, the blocked implementation consistently produces higher bandwidths, no matter the matrix and block sizes when compared to the naive implementation. We assume this is due to caching consistently improving performance, where even a sub-optimal cache reduces hits to memory compared to no caching.

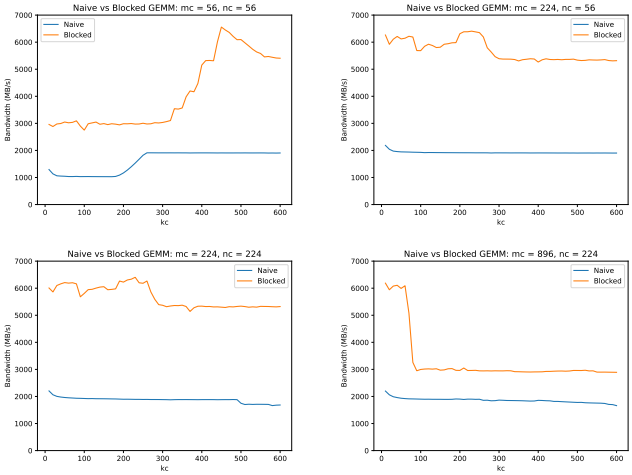

Fig. 2: Comparisons between the Naive and Blocked GEMM implementation.

IV. Discussion

Our evaluation of the blocked General Matrix Multiplication (GEMM) implementation revealed consistent improvements in bandwidth with increasing matrix size. This indicates effective cache utilization and reduced cache misses. However, we noted diminishing returns beyond a certain matrix size, suggesting suboptimal cache handling.

Comparisons with the naive GEMM implementation consistently favored the blocked approach, showcasing its superior cache utilization and overall computational efficiency. These findings underscore the significance of algorithmic optimizations such as blocking in enhancing performance on modern architectures.

Future work could focus on fine-tuning block sizes and exploring alternative memory management strategies to further optimize performance. Additionally, extending the evaluation to other matrix operations and hardware architectures could provide valuable insights into the broader applicability of the blocked approach.

REFERENCES

[1] G. Alaejos, A. Castelló, H. Martínez, P. Alonso-Jordá, F. D. Igual, and E. S. Quintana-Ortí, "Micro-kernels for portable and efficient matrix multiplication in deep learning," *J. Supercomput.*, vol. 79, p. 8124–8147, dec 2022.