CSCE 654 HW1 Report

Aditya Kovilur

September 15 2025

1 Introduction

This report analyzes the performance of **DGEMM** and attention matrix computation (with and without DGEMM). The observations have been made through testing on the **Perlmutter** supercomputer with an interactive node.

2 DGEMM

This is an optimized implementation for dense matrix-multiplication with floation point operations. The observations were made with thread counts of [1, 2, 4, 8, 16, 32] and various scheduling strategies. The results are shown in Table 1 and Table 2 and the plot is shown in Figure 1.

Threads	**Time (s)**	**Rate (GFLOP/s)**
32	5.048054	3.40
16	10.394406	1.65
8	18.153665	0.95
4	34.489467	0.50
2	67.694542	0.25
1	135.039040	0.13

Table 1: DGEMM performance across thread counts (updated).

Scheduling Strategy	**Performance (GFLOPS)**
`schedule(dynamic, 64)`	1.61
`schedule(dynamic, 128)`	1.94
`schedule(dynamic, 256)`	1.67
`parallel for`	1.91
`parallel for collapse(2)`	2.95

Table 2: Performance comparison of different OpenMP scheduling strategies.

3 Attention

The computation of the attention matrix involves multiplying elements of 3 matrices. This can be done by abstracting out the matrix multiplication part with DGEMM or parallelising vector multiplications and reducing overhead. The observations are shown in Table 3 and Table 4. The comparison plot is shown in Figure 2. The number of FLOPs is taken as 4 * L * L * D + 60 * L * L by taking approximations for std::exp and division operations as 50 and 8 respectively.

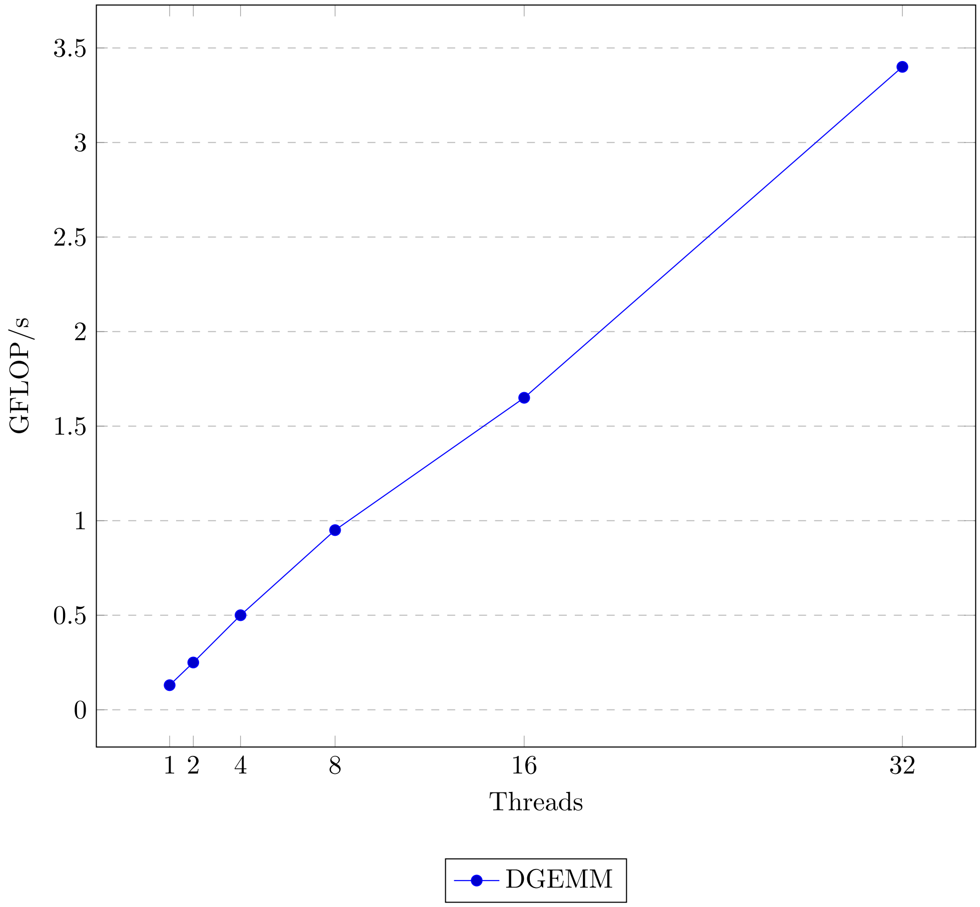

Figure 1: DGEMM performance scaling (updated).

Threads	**Time (s)**	**Rate (GFLOP/s)**
32	3.632724	0.07
16	6.628532	0.04
8	11.779136	0.02
4	22.991414	0.01
2	45.062622	0.01
1	90.646831	0.00

Table 3: Attention via DGEMM.

Threads	**Time (s)**	**Rate (GFLOP/s)**
32	1.365518	0.18
16	2.402984	0.10
8	4.066465	0.06
4	8.013019	0.03
2	15.704003	0.02
1	31.831728	0.01

Table 4: Attention direct.

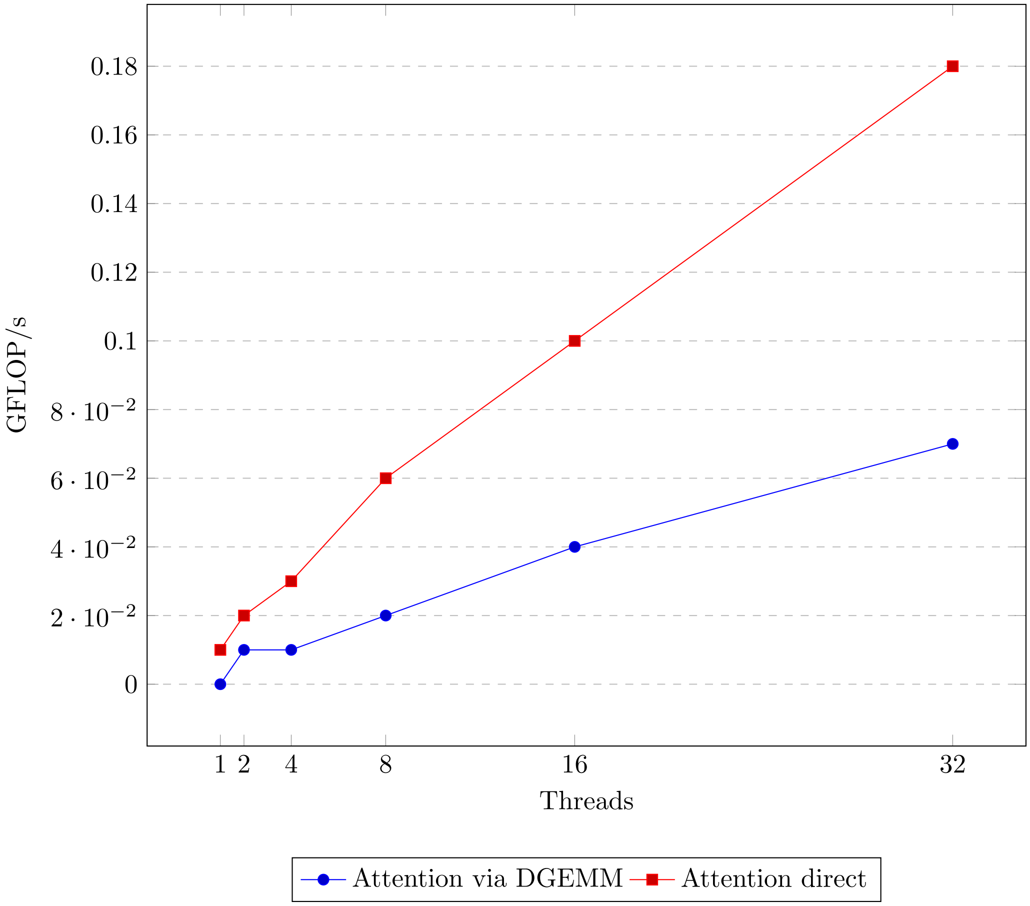

Figure 2: Attention performance scaling (updated).