# MA4261 - HWK1 - Matrix-Matrix Multiply on the CPU

TOBIAS BISCHOFF[*]

*California Institute of Technology, Pasadena, California*

[*] *Corresponding author address:* Tobias Bischoff, Division of Geological and Planetary Sciences, 1200 E California Blvd, Pasadena, CA 91125.

E-mail: tobias@caltech.edu

# ABSTRACT

The algorithm follows in spirit the algorithm for $C = C + AB$. The main modification lies in how the matrix $B$ is accessed, e.g., changes from columns to row, etc. The “hardest“ modification to make is in the packing routines as a new packing routine is required for packing $B^T$. The multi-threaded implementation follows the one in class. A $10\ \%$ speed-up was found by optimizing vector loads and storage in SIMD.jl helping LLVM under the hood.

# List of Figures

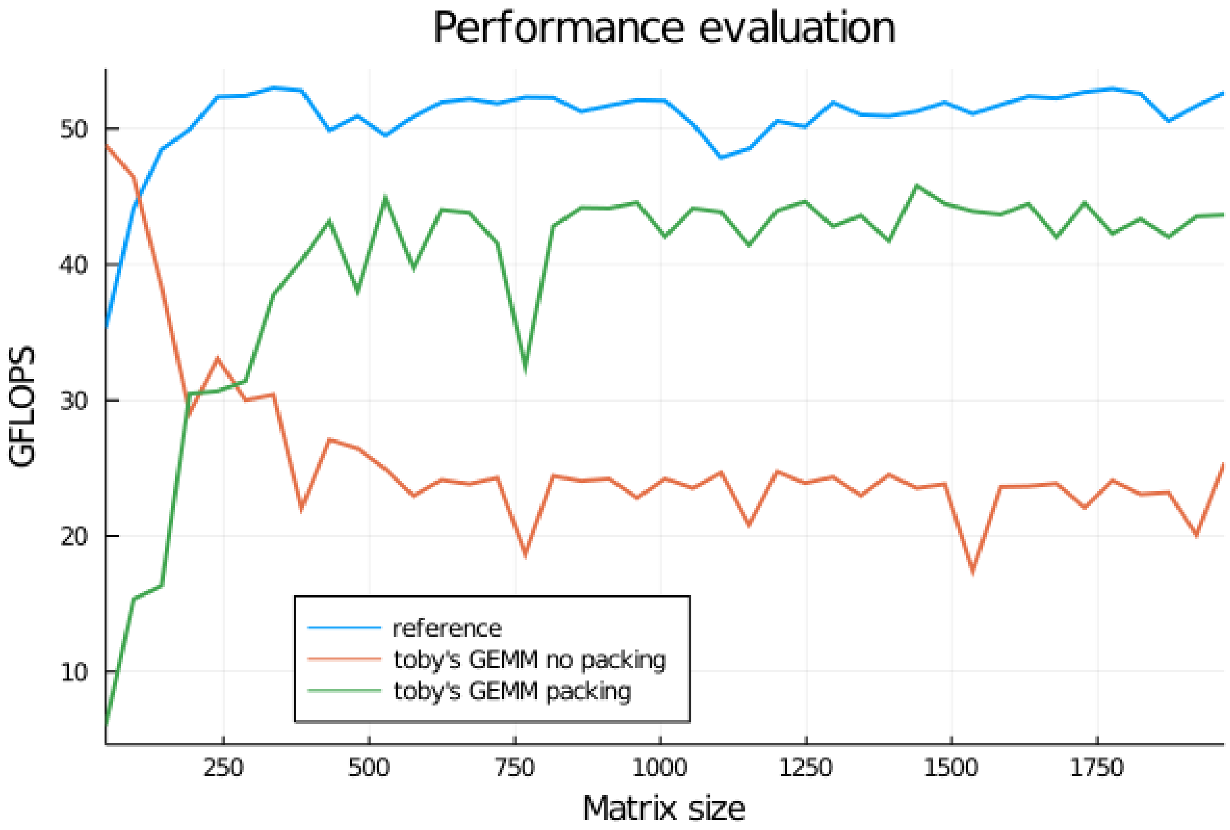

**Figure 1:** Performance evaluation, comparing three implementations of $C = C + AB^T$. The **blue** line shows the performance of the reference implementation. The **red** line shows the unpacked single-threaded version by the author, and the **green** line shows the packed single-threaded version by the author. We can see that without packing the performance drops off quickly, pointing to poor cache management. By comparison the packed version and the reference version show no sign of performance reduction in tested size range. The reference implementation performs about 10-20% better than the best implementation by the author. Testing was performed with a Quad-Core Intel Core i7 at 3.1 GHz, 32KB L1 cache, with 256KB L2 cache, 8MB L3 cache, and 16GB RAM. The parameters found to work best for this setup are $mc = 96, nc = 2048, kc = 256, mr = 6, nr = 4, vl = 4$ at double precision.

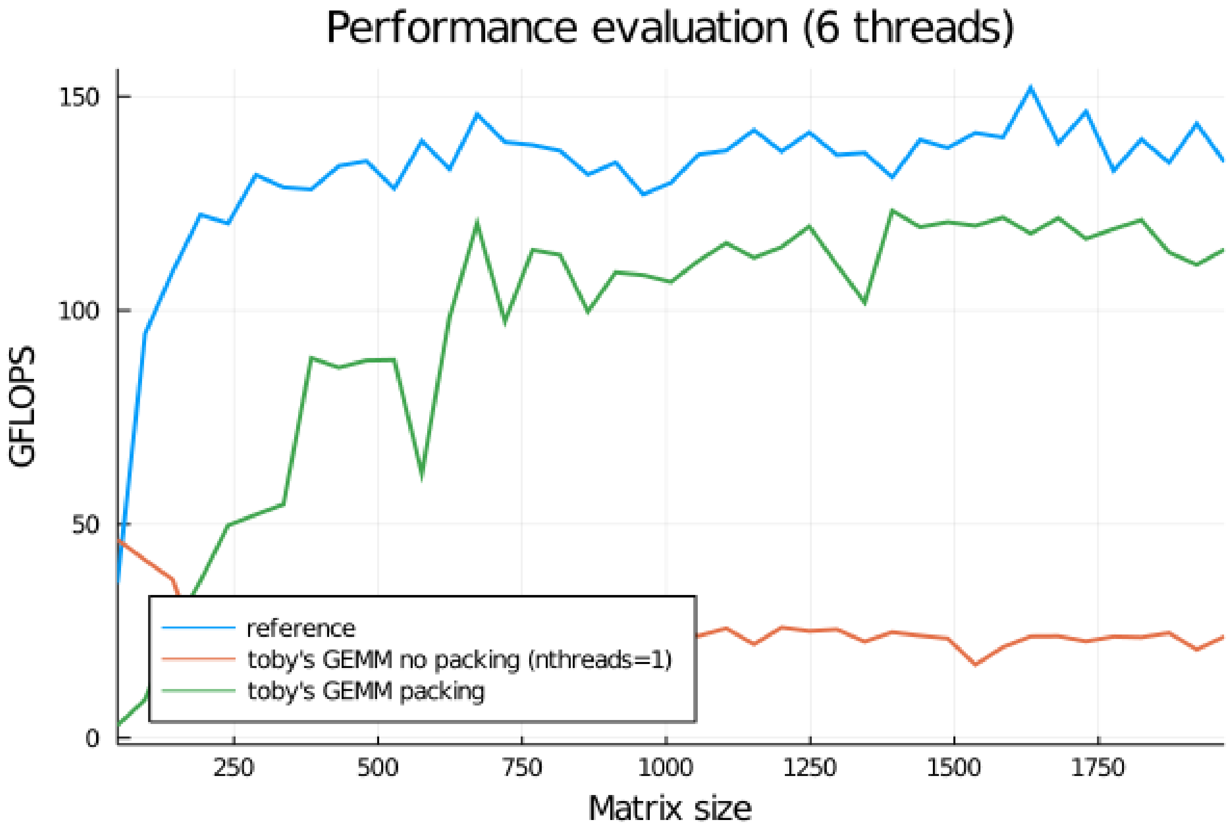

**Figure 2:** Same as Figure 1, but this time with 6 threads. The **blue** line again shows the performance of the reference implementation, but this time with 6 threads. The **red** line shows the unpacked single-threaded version by the author, and the **green** line shows the packed six-threaded version by the author. We can see that without packing the performance drops off quickly, pointing to poor cache management. By comparison the packed version and the reference version show no sign of performance reduction in tested size range. The reference implementation performs about 10-20% better than the best implementation by the author. Testing was performed with a Quad-Core Intel Core i7 at 3.1 GHz, 32KB L1 cache, with 256KB L2 cache, 8MB L3 cache, and 16GB RAM. The parameters found to work best for this setup are $mc = 96, nc = 2048, kc = 256, mr = 6, nr = 4, vl = 4$ at double precision. We did not achieve linear scaling with the number of threads.