CACHING AND ITS EFFECTS ON TILED MATRIX MULTIPLICATION

Current systems for machine learning systems commonly involve large matrix multiplications with thousands of indices and up to trillions of calculations. When combined with the massive data sets necessary for machine learning, any sort of optimization for the multiplications between matrices will have significant time savings for algorithms in machine learning.

One simple and fundamental optimization is to tile matrices when multiplying them. This has advantages from a major time-intensive step in the multiplication process, which is actually fetching the data that must be multiplied. First fetching the value for each index of the matrix causes delays if the value isn’t in a local cache, because the processor must reach into the main memory that is much farther away rather than being able access it in the physically closer and quicker data access in the cache. Matrix multiplication is especially prone to this error, because its repetitive use of the same values over and over requires many slow fetches to the main memory if the matrices being multiplied are larger than the space available in the cache and forcing the cache to drop some values between cycles of multiplication.

Tiling minimizes the number of slow fetches to main memory by maximizing the use of a value while it’s in the cache. By dividing the matrices being multiplied into smaller blocks that can fit into the cache, the algorithm minimizes the number of slow fetches required. We will demonstrate the improvements tiling can bring to matrix multiplication processes by comparing a naïve vs. a tiled matrix multiplication algorithm implemented in C and their performance on matrices of different sizes and then analyze how some of these changes may be traced back to the caches of the processor.

**NAÏVE IMPLEMENTATION**

The naïve implementation of the matrix multiplication simply iterates along the output matrix, by fully calculating each index of the output matrix before moving on to the next index. This gives the output for that index of the matrix immediately, but is very inefficient in using the cache. For example, given the 4x4 matrix multiplication:

A matrix’s indices are often repetitively used