gemm3(): Constant-workspace high-performance multiplication of three matrices

Krzysztof A. Drewniak, Tyler M. Smith, Robert van de Gejin

February 7, 2018

1 Background

1.1 High-Performance `gemm()`

Before discussing `gemm3()`, it is important to review the techniques for the operation $C \coloneqq \alpha AB + \beta C$, that is, `gemm()`. For simplicity, we'll present the operation as $C \mathrel{+}= AB$ for simplicity. A naive implementation would proceed as follows (where A is m by k, B is k by n, and c is m by n) This

Algorithm 1 Naive implementation of `gemm()`

```
procedure GEMM(A, B, C)
    for i ← 0 up to m do
        for j ← 0 up to n do
            for c ← 0 up to k do
                C_{i,j} ← C_{i,j} + A_{i,c} B_{c,j}
```

algorithm has serious performance issues in that it accesses the memory of one of the operands (A for row-major storage and B for column-major) at a stride of k, which is almost always a number that makes it impossible for the processor to stream both matrices' values into memory through prefectching or to vectorize the memory accesses, which would allow multiple elements of C to be computed simultaneously on the same CPU core. Therefore, it is effectively never used in practice except as a verification tool for more efficient algorithms.

Many of the high-performance `gemm()` algorithms in use today are based on the approach of Goto**TODO cite** These algorithms massively improves performance by taking advantage of the multi-level cache present on modern CPU architectures. They operate by reducing the `gemm()` to a series of sub-problems that are sized such that their inputs and/or outputs fit into the levels of the system's cache, and additionally by rearranging the inputs to those subploblems into a form that can be streamed from cache by the *microkernel*, a highly-optimized inner loop.

One commonly-used algorithm of this type is the BLIS algorithm,