Parallel Multi-Channel Multi-Kernel Convolution

James Tait, Seamus Woods, Tom Wisniowski

Algorithmic Improvements

The first improvements we made to the function were simply modifications to the existing code. We noticed that the output was continuously overwritten when placed within the ‘c’ for loop. We moved it out of that for loop so the final output was stored just the once.

We also noticed that ‘a’ did not change when ‘m’ did, so we calculated two ‘b’s, and subsequently two ‘sums’, at once, in order to evaluate ‘a’ half as many times.

SIMD SSE - x86 Intrinsics

The second optimization we added to the algorithm.. __m128d data type (2 64 bit double precision floats).

The reason for not using __m128(4 32 bit single precision floats) was because of a loss in accuracy.

Using this data type we were able to halve the amount of times a for loop was run.

Implemented for the number of kernels and the number of channels.. This was particularly good for the number of kernels since it is the outermost loop, meaning the entire code was only being executed half the amount of times.

Threaded Parallelization - OpenMP

Using “#pragma omp parallel”, we were able to run threads concurrently.

This allowed us to divide up the work between these threads which sped up our program by at least the factor of 2.

“Parallel For” divided the iterations of a for loop between the threads while using the extension “collapse()” merged several for loops into an iteration space and divided according to the schedule clause.

Execution Times

Small: 16 16 1 32 32.. 2089 Microseconds

Medium: 128 128 3 512 512.. 3.88 Seconds

128 128 7 512 512.. 15.4 Seconds

Large: 256 256 3 1024 1024.. 44.92 Seconds