SPM - Assignment 1

Angelo Savino

March 11, 2025

1 Handouts

This assignment consisted in writing an optimized version of the softmax function. The three versions we compared are the following:

- `softmax_plain`: Basic version of the softmax function that does not employ any kind of SIMD optimizaation.
- `softmax_auto`: Version of the softmax function that relies on compiler optimizations to produce AVX code.
- `softmax_avx`: Hand-optimized version of the code that uses AVX intrinsics.

2 Softmax_auto

The `softmax_auto` binary was compiled using the `-march=native` and `-ffast-math` compiler flags, to allow for the use of respectively AVX instructions and fast math code. Also at the start of the `softmax_auto()` function, the input and output vector have been tagged with `__restricted__`, to allow the compiler to make better assumptions about the code we will need to generate.

3 Softmax_avx

The hand-optimized version optimizes all loops using avx instructions. The first loop is the one present in the `find_max` function, where we load 8 floats per iteration into a AVX2 register using `_mm256_loadu_ps`, then we find the max for each slot using `_mm256_max_ps`. Finally, we store the current max values we found in a temporary array of size 8, and we find the max value over the elements by iterating over them.

In the same fashion, we load compute the exponentials in groups of 8 by first subtracting to each of them the max value using `_mm256_sub_ps`, then compute the exponential using `exp256_ps`. Once we do this, we save the values into the output vector, and we also sum them up into another register which will be reduced to the sum of all exponentials at the end.

At last, we divide each element by the computed sum of the exponentials using the `_mm256_div_ps`.

4 Performance Comparison

The final performance comparison is summarized by fig. 1. As we can see, there is not much difference between the version when the size of the vector K is sufficiently small.

When vectors get biggger ($K > 10^3$), we can start noticing a difference in execution time. The unoptimized version is the slowest, while the auto optimized one results in a speedup of about 2.7x for the largest K we tested ($K = 10^{10}$) and the hand-optimized one improves our times by an additional 1.24x (3.3x faster than plain).

Overall, `softmax_avx` is the best performing implementation, but it comes at the cost of portability, meaning that if we would like to run our code on other machines that are not provided of the AVX2 extension, we would be better off usign our auto-optimized version if we can accept slighlty higher run times.

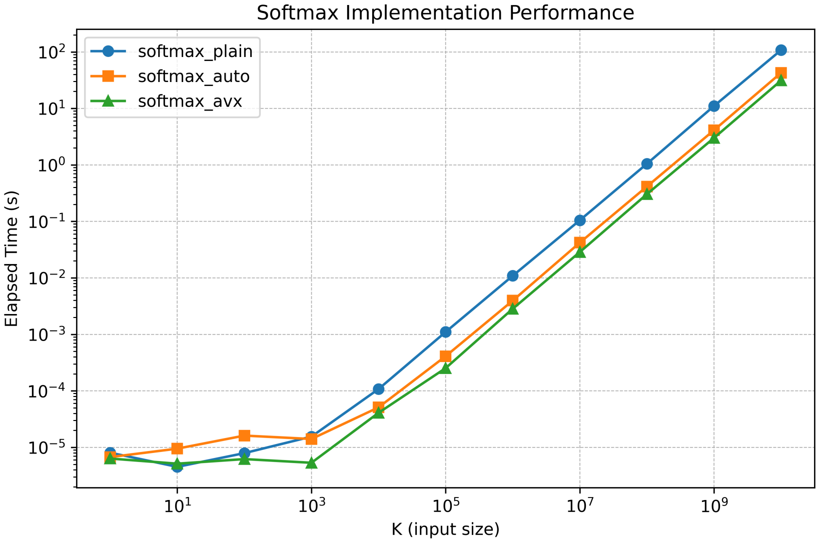

Figure 1: Comparison between all variants of softmax