Third homework

Janez Kuhar

May 11, 2024

Abstract

We implemented a vectorized version of the K-means algorithm for image pixel segmentation. We've measured the results of the scalar and vectorized implementation for proposed combination of image pixel data types and cluster sizes. Finally, we've commented our results.

Introduction

In this assignment, we have have been tasked with implementing the vectorization of the K-means algorithm for grayscale-level quantization. Within the codebase of the scalar version, two critical functions required vectorization. Firstly, the `assign_points_to_clusters` function assigns pixels to their nearest cluster based on the difference between each pixel and the cluster centroid. Secondly, the `update_centroids` function recalculates the centroids of clusters based on the pixels assigned to each cluster. he K-means algorithm iteratively applies these two functions until the change in centroid values falls below a specified threshold. Our task was to implement efficient vectorized versions of these functions to enhance the performance of the K-means algorithm for grayscale-level quantization.

Methods

We've compared scalar and vectorized version of the program using `unsigned char`, `float`, and `double` as the image pixels data type. In addition, we've tried cluster sizes 4, 8, and 16. Bellow are data type-specific implementation notes for the vectorized version of the algorithm.

8-bit unsigned integers

We decided not to vectorize function `assign_points_to_clusters` as despite our best efforts, the vectorized results were slower.

For `update_centroids`, we've vectorized accumulation of pixel intensities in a cluster. In [1], a clever trick is proposed to sum up 16 bytes in a 128-bit vector. We take the next 16 pixels in the cluster and pack them into a 128-bit vector. Than we make 4 copies of the vector, by shifting the original by 2^k where k goes from 0 to 3. A mask is then applied to every shifted vector, selecting only the first byte in each 32-bit lane. We sum all the pixel intensities in a cluster in this fashion. Finally, we combine the 4 partial sums in the 128-bit accumulator vector into the final sum.

Floats

We've parallelized `assign_points_to_clusters` for cluster sizes 4 and 8. The idea was to prefill a vector with cluster centroids and then broadcast the image intensity to another vector to compute the differences between an image pixel and all cluster centroids at once. We've computed the absolute value by setting the highest bit to 0. We couldn't find an efficient way to vectorize this function for cluster size equal to 16.

For `update_centroids`, we simply summed up 8 elements at the time.

Doubles

We've parallelized `assign_points_to_clusters` for cluster size equall to 4. The idea was to prefill a vector with cluster centroids and then broadcast the image intensity to another vector to compute the differences between an image pixel and all cluster centroids at once. We've computed the absolute value by setting the highest bit to 0. We couldn't find an efficient way to vectorize this function for cluster sizes 8 and 16.

For `update_centroids`, we simply summed up 4 elements at the time.

Results

We have performed our measurements on ARNES HPC cluster. Compute nodes were equipped with 64 core *AMD Epyc 7H12* processors, with **AVX2** support.

Table 1: *8-bit unsigned integers: Runtimes (in seconds) for scalar and vectorized version for cluster sizes (k) 4, 8, and 16.*

k	sca	vec	speedup
4	4.87	4.8	1.01
8	3.69	3.66	1.01
16	18.06	18.08	1.00

Table 2: *32-bit floats: Runtimes (in seconds) for scalar and vectorized version for cluster sizes (k) 4, 8, and 16.*

k	**sca**	**vec**	**speedup**
4	14.11	10.11	1.40
8	8.56	4.96	1.73
16	19.01	18.63	1.02

Table 3: *64-bit doubles: Runtimes (in seconds) for scalar and vectorized version for cluster sizes (k) 4, 8, and 16.*

k	**sca**	**vec**	**speedup**
4	14.86	10.24	1.45
8	8.85	8.45	1.05
16	20.3	18.98	1.07

Discussion

Tables 1, 2, and 3 show measurements for 8-bit integer, single-precision, and double presicion floating point resprectively. Where we did not vectorize `assign_points_to_clusters`, no significant speedups were obtained.

References

[1] "Optimization of summing bytes with SSE instructions". In: (2018). URL: `http://0x80.pl/notesen/2018-10-24-sse-sumbytes.html`.