Hardware für Eingebettete Systeme

Lab 4: AVX instrinsics and masking

Prof. Dr. Mario Porrmann
M.Sc. Marc Rothmann

1 Masking intrinsics

Using the `blend` intrinsics, we can merge two vectors based on a mask that we supply. E.g., the function:

```
__m256 _mm256_blend_ps (__m256 a, __m256 b, const int imm8)
```

Implements the following operation:

```
FOR j := 0 to 7
    i := j*32
    IF imm8[j]
        dst[i+31:i] := b[i+31:i]
    ELSE
        dst[i+31:i] := a[i+31:i]
    FI
ENDFOR
dst[MAX:256] := 0
```

This can be used to control the data flow in AVX programs.
Implement the following functions using AVX intrinsics:

a) For two vectors a and b:

$$c_i = \begin{cases} a_i - b_i & \text{if} \quad a_i \geq b_i \\ a_i & \text{otherwise} \end{cases}$$

b) For a vector a:

$$c_i = \begin{cases} sqrt(a_i) & \text{if} \quad a_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Hint: It's probably easier to use `_mm256_blendv_ps` instead of `_mm256_blend_ps`.

2 Vectorizing K-Means

The K-Means algorithm is a clustering algorithm that aims to partition n observations into k clusters. Each cluster is represented by a cluster center (the mean). The observations belong to the cluster with the closest cluster center. K-Means iteratively moves the centers to minimize the within cluster variance.
This is the pseudo-code of the K-Means algorithm:

Initialize k means m_0, ..., m_{k-1}
for t = 1, ..., EPISODES:

1. Assign each data point x_j to a cluster S_i:

$$S_i^t = \{x_j : ||s_j - m_i^t||^2 \leq ||x_j - m_{i^*}^t||^2 \quad \text{for all} \quad i^* = 1, ..., k\}$$

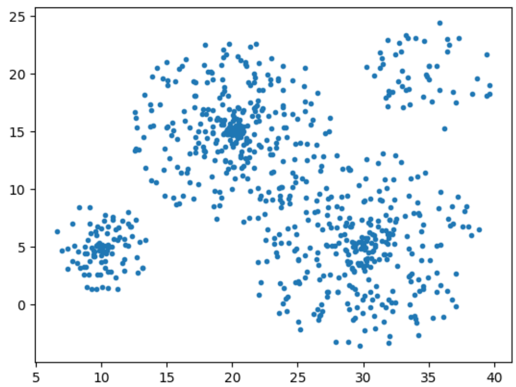
2. Update the mean m_i of each cluster S_i:

$$m_i^{t+1} = \frac{1}{|S_i^t|} \sum_{x_j \in S_i^t} x_j$$

return m_0, ..., m_{k-1}

Algorithm 1: K-Means

The file clusters.csv contains 750 2D-points which can be partitioned into 4 clusters. The following figure visualizes the data.



The cluster centers are:

$$m_0 = (10, 5)$$
$$m_1 = (30, 5)$$
$$m_2 = (20, 15)$$
$$m_3 = (35, 25)$$

a) The file `kmeans.c` contains un-vectorized k-means code. Implement the vectorized version `kmeans_simd`. You need to use the following compiler flags:

```
gcc -mavx -O3 kmeans.c -lm
```

Verify that your algorithm approximates the correct cluster centers.

b) Compare the speed of your implementation to the speed of the un-vectorized version. You can use the `eval.py` script to generate a plot showing your computed cluster centers in the data.

Abgabe zum 4.6.2023!