Data Processing on Modern Hardware
Assignment 4 – SIMD Vectorization

Handout: **3rd June 2020**
Due: **10th June 2020 by 9am**

Part 1: Aggregation

The following aggregation query counts the number of elements that are lower than 42. You can assume that R is pre-populated with 100 million entries.

```
SELECT COUNT(*)
FROM   R
WHERE  R.a < 42
```

Use the code skeleton provided in the gitlab repository[1] for two functions that implement the aggregation query listed above (`count8` and `count64`) to solve the following tasks:

1. Investigate whether `GCC` and `clang` can auto-vectorize these functions under different optimization settings (O3, O2, O1). You can use `https://godbolt.org` for your analysis.
2. Implement a branch free version of count8 and count64.
3. Implement count8SIMD and count64SIMD version of the code using AVX-512 intrinsics [2].
4. Discuss how the branch free version and SIMD-vectorization change the performance of the algorithm? Include the compiler flags and your performance numbers/profiling in the submission file/report.

Part 2: Dictionary decompression

You are given an array with 4-bit dictionary compressed 32-bit integers. The code skeleton provided to you has a function `dictDecompress4to32` that decompresses the elements of the array. Your task is to do the following:

1. Speed up the given code by creating a scalar version that loads 8 bytes at a time, instead of 1 byte.
2. Implement an AVX-512 version that uses the `gather` instruction for the dictionary look-up.
3. Implement an AVX-512 version that stores the dictionary in a SIMD register and uses `permute` instead of `gather`.
4. Discuss the properties of the different versions of the code.

[1] https://gitlab.db.in.tum.de/dpmh-ss20/hw4
[2] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#avx512

Submission guidelines

This homework has a duration of one week. Fork the repository, commit your changes in the git, and invite us (*@dpmh*) to hand in your homework.

The programming language of this homework is C++. We provide you a simple code skeleton, feel free to add functions. For performance measuring of the experiments you can either use the provided `perfEvent.hpp` and the commented code blocks or use the tools you applied in the previous assignments.