FIAS Frankfurt Institute for Advanced Studies

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

High-Performance Computer Architectures Practical Course

Single Instruction Multiple Data (SIMD) Instructions and Vector Class `Vc`

Prof. Dr. Ivan Kisel
Robin Lakos
Akhil Mithran
Oddharak Tyagi

May 27, 2025

Repetition

Last week we discussed SIMD instructions that allow to apply the *same operation on multiple data in parallel*.

We provided the FVec header that wraps `__m128` intrinsics for 4×32-bit `float` values together with the intrinsics itself. Such a wrapper can provide some quality of life features when working with SIMD instructions.

One of the main problems with wrappers: The intrinsics are strictly defined for specific vector register sizes (e.g. 128 bit) and new wrapper headers are required in case someone wants to use, for example, 256 bit or 512 bit registers.

Solution: In the first lectures, we already discussed templates, allowing to apply the same functionality for different data types. The idea: A library that reads out the maximum register size it was compiled for that additionally makes use of fully templated functions, to allow flexible usage of SIMD on different processors.

Vector class `Vc`

The Vector class (`Vc` library is a free software explicitly for vectorization of C++ code. The library provides all the functionality of the provided `FVec` header and much more.

`Vc` provides branch operators with masks and random memory access with the gather and scatter functionalities.

Masks: Allow vectorized conditional branches where the result of a boolean condition check is stored in the mask. An overloaded `()-operator` can then be used to apply functionalities only to those elements that have a `true`-value in the mask, which are the elements that fulfilled the condition.

Gather: Allows to provide a vector of indices along with the data array to collect the data from the given element positions.

Scatter: Allows to provide a vector of indices along with the data array to distribute the data to the given element positions.

Additionally, `Vc` is a template library that allows to run the same code on different architectures.

In this week, we will again apply SIMD instructions by using the `Vc` library. Since `Vc` will be integrated in to the C++ standard (expected in C++26), we will use the `std::experimental` library to apply these functionalities for some exercises.

The std::experimental Library Example

```cpp
#include <experimental/simd>
#include <iostream>

namespace stdx = std::experimental;

using float_v = stdx::native_simd<float>;

int main() {
    alignas(16) std::array<float, 4> a_arr = {1.f, 2.f, 3.f, 4.f};
    alignas(16) std::array<float, 4> b_arr = {.1f, .2f, .3f, .4f};
    alignas(16) std::array<float, 4> c_arr;

    float_v& a = reinterpret_cast<float_v&>(a_arr[0]);
    float_v& b = reinterpret_cast<float_v&>(b_arr[0]);
    float_v& c = reinterpret_cast<float_v&>(c_arr[0]);

    c = a + b;

    for (size_t i = 0; i < c.size(); ++i)
        std::cout « "c[" « i « "] = " « c[i] « std::endl;

    return 0;
}
```

The `VcDevel` Library

In its current state, `std::experimental` does not support all functionalities (e.g. gather and scatter) of `Vc`.

Therefore, you are asked to install the VcDevel library on the computers on-site by running the provided bash script. For details, please have a look at the exercise sheet.

In `Vc`, the required functions are available. If you follow the instructions on the exercise sheet, the last exercise should run ”out of the box”.