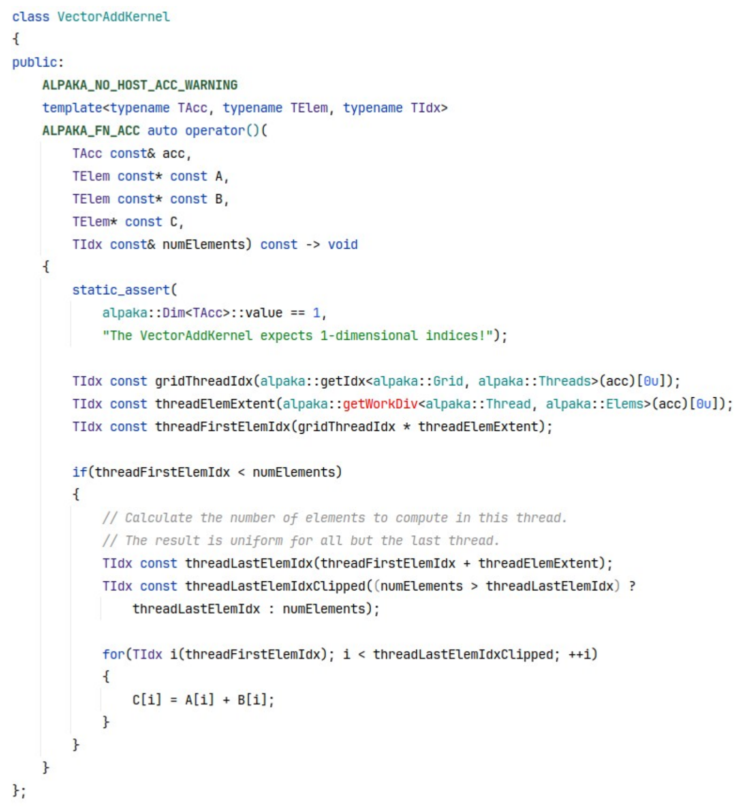
Extending the Alpaka performance portability library with CUDA Cooperative Groups for the CMS pixel reconstruction

M.O. Varvarin
mentors: Jiri Vyskocil,
Volodymyr Bezguba

Contacts: **Mykhailo Varvarin,**
Kyiv Academic University, Ukraine
michael.varvarin@gmail.com
+380 96 47 27 225

Alpaka

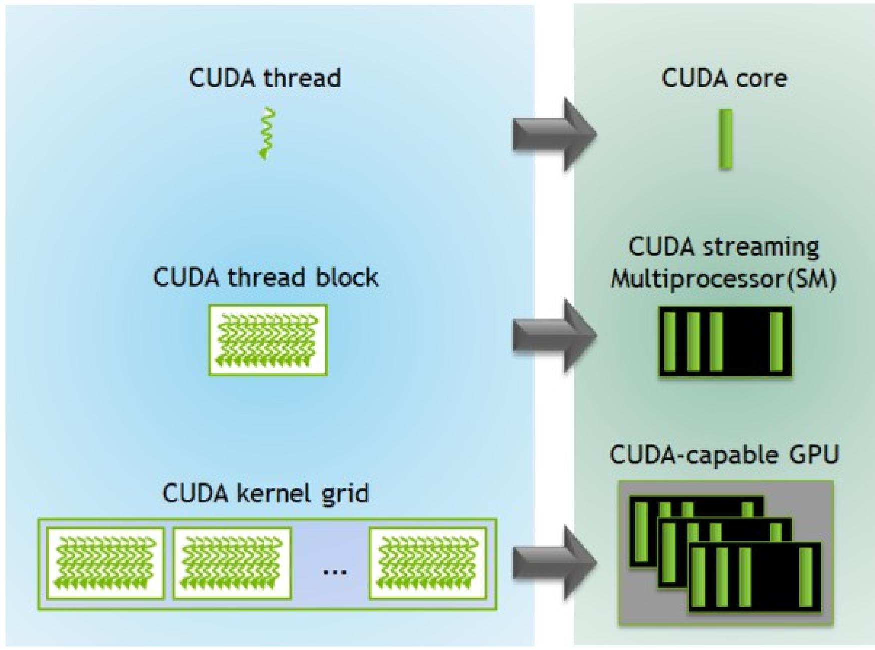
- alpaka library is a header-only C++17 abstraction library for accelerator development.
- Its aim is to provide performance portability across accelerators through the abstraction of the underlying levels of parallelism.
- Alpaka supports both GPU (CUDA, HIP and SYCL) and CPU (OpenMP, std::threads and Intel TBB) accelerators, with ability to recompile your code from one to the other, changing just a few lines of code.



```cpp
class VectorAddKernel
{
public:
    ALPAKA_NO_HOST_ACC_WARNING
    template<typename TAcc, typename TElem, typename TIdx>
    ALPAKA_FN_ACC auto operator()(
        TAcc const& acc,
        TElem const* const A,
        TElem const* const B,
        TElem* const C,
        TIdx const& numElements) const -> void
    {
        static_assert(
            alpaka::Dim<TAcc>::value == 1,
            "The VectorAddKernel expects 1-dimensional indices!");

        TIdx const gridThreadIdx(alpaka::getIdx<alpaka::Grid, alpaka::Threads>(acc)[0u]);
        TIdx const threadElemExtent(alpaka::getWorkDiv<alpaka::Thread, alpaka::Elems>(acc)[0u]);
        TIdx const threadFirstElemIdx(gridThreadIdx * threadElemExtent);

        if(threadFirstElemIdx < numElements)
        {
            // Calculate the number of elements to compute in this thread.
            // The result is uniform for all but the last thread.
            TIdx const threadLastElemIdx(threadFirstElemIdx + threadElemExtent);
            TIdx const threadLastElemIdxClipped((numElements > threadLastElemIdx) ?
                threadLastElemIdx : numElements);

            for(TIdx i(threadFirstElemIdx); i < threadLastElemIdxClipped; ++i)
            {
                C[i] = A[i] + B[i];
            }
        }
    }
};
```

CUDA cooperative groups

- Traditionally CUDA had only block-level (1024 threads) synchronization. This requires usage of dynamic parallelism for a lot of algorithms, which has a large overhead.
- Cooperative groups are a new abstraction that add support for synchronization both on sub-block level and the whole grid level, allowing for more optimization.



Thank you for your attention.