Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Exploring the portability of the ALICE ITS clustering to alpaka

Dr. Leonardo Cristella

Tecnologo @ INFN Bari

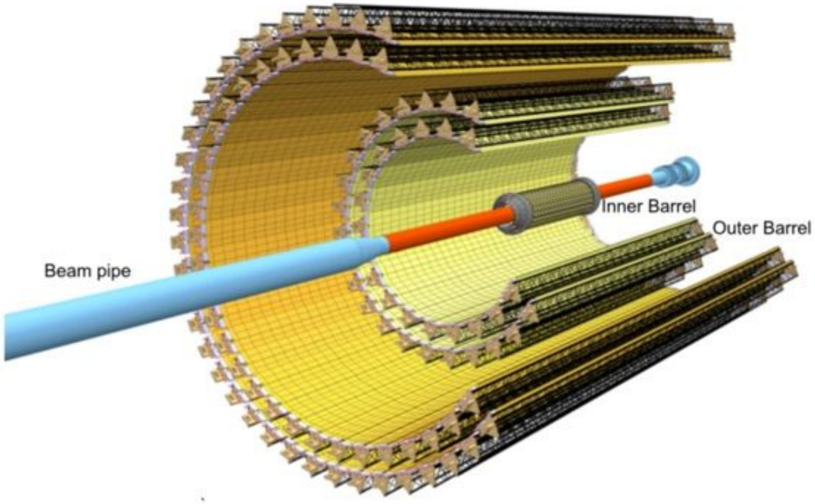
Progetto ICSC - Spoke 2 - Codice CN00000013 -
CUP I53C21000340006 - Missione 4 Istruzione e ricerca

Outline

- ITS Clustering
- alpaka and its benefits
- First attempt to integrate alpaka in AliceO2
- Conclusions
- Future plans

The Inner Tracking System (ITS)

- **7 layers of silicon sensors**
- Sensitive area of **10 m²**
- Over **12.5 billion pixels** in total
- Provides **high granularity** for precise tracking
  - Enhance tracking resolution
  - Enable detection of short-lived particles like charm and beauty hadrons
  - Operates in high-radiation environ



ITS Clustering

- Raw Data:
    - Millions of hits from pixel sensors.
    - Hits need to be grouped to reconstruct particle trajectories.
- Challenges:
    - Handling large data volumes efficiently.
    - Maintaining accuracy in noisy environments.
- Key role:
    - Converts pixel hits into structured clusters, essential for high-resolution tracking and particle identification.
        - **Skip “masked” pixels**
            - invalid or noisy pixels

The alpaka library

- Key features:
    - Unified programming model for heterogeneous systems.
    - Write once, run on multiple backends (CPU, GPU, FPGA, etc.).
- Why alpaka?
    - Performance portability without vendor lock-in.
    - Flexibility to adapt to evolving hardware ecosystems.
- Integration into AliceO2:
    - Modular design allows for incremental adoption in existing workflows.
- CMS experience:
    - Integration development started in 2022
    - First deployment in August 2024

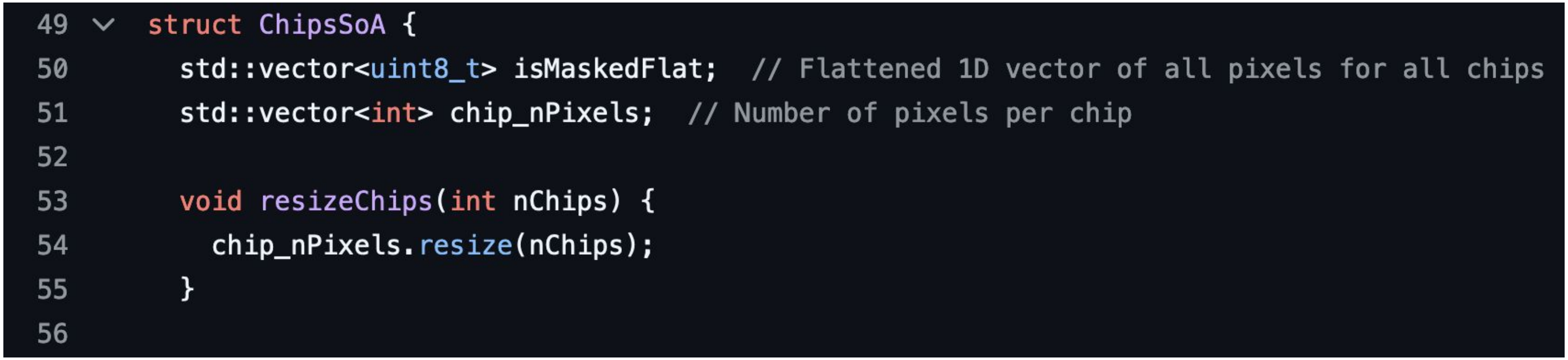
alpaka - Abstraction Library for Parallel Kernel Acceleration

Introducing alpaka to ITS Clustering

- Challenges:
    - Existing code: `Detectors/ITSMFT/common/reconstruction/src/Clusterer.cxx` heavily reliant on CPU-based processing.
        - data organized as **Array of Structs**
    - Need to refactor for alpaka's execution model:
        - Enables efficient memory access for Alpaka kernels: **Struct of Arrays (SoA)**
        - Flattened representation of pixel data.
        - Compact storage for `isMasked` pixels and cluster metadata.
- Impact:
    - Reduced memory overhead.
    - Improved performance on vectorized hardware.

Chips SoA

Minimal implementation of an **SoA** to store the information of whether a pixel is masked or not:



```
struct ChipsSoA {
  std::vector<uint8_t> isMaskedFlat;  // Flattened 1D vector of all pixels for all chips
  std::vector<int> chip_nPixels;  // Number of pixels per chip

  void resizeChips(int nChips) {
    chip_nPixels.resize(nChips);
  }

```

The size of `isMaskedFlat` is given by the sum of the number of pixels per chip (`nChips` is the total number of chips for a given ROF).

ClusterKernel for Alpaka execution

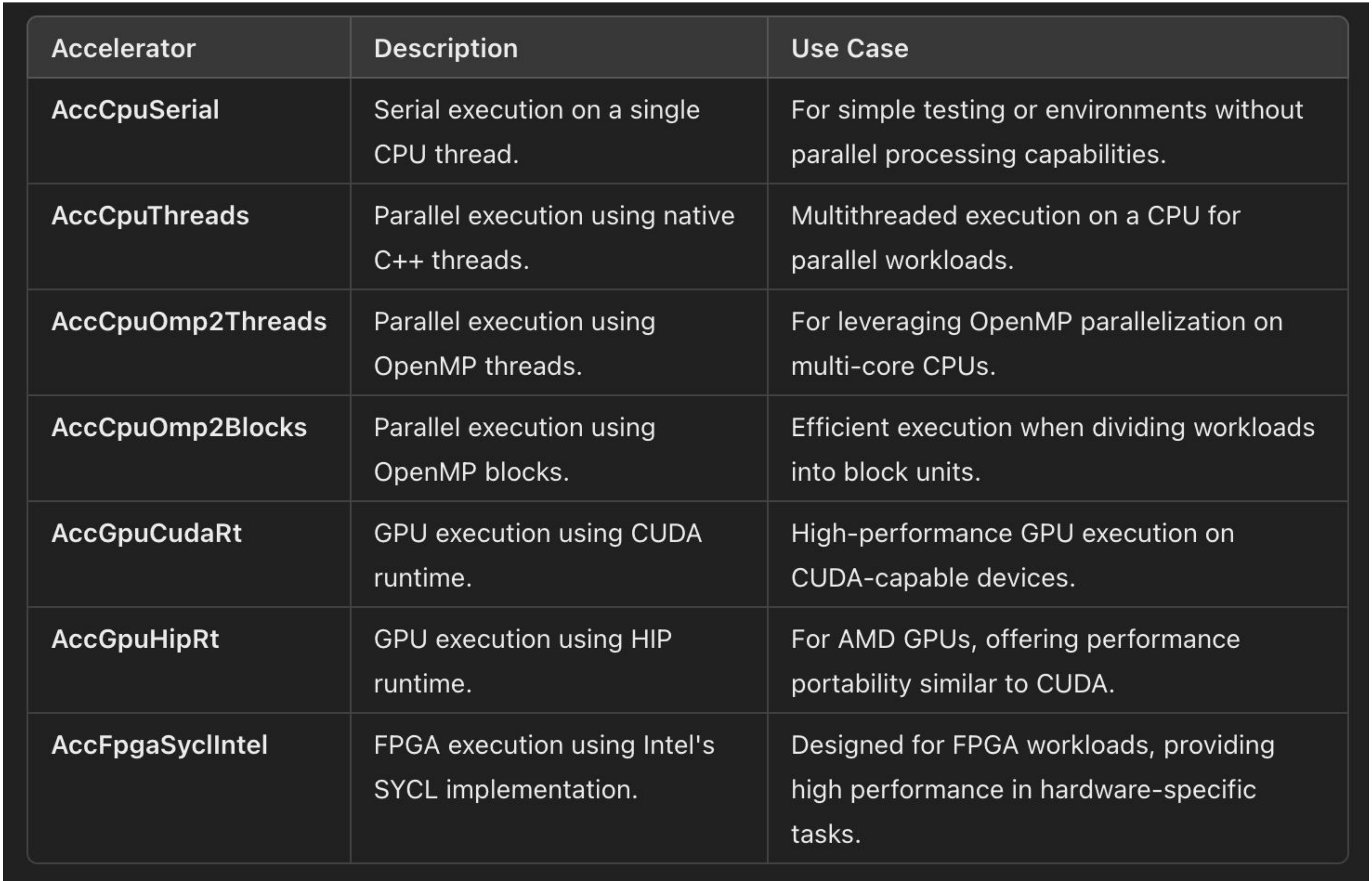
```
struct ClusterKernel_1D {
  template <typename Acc>
  ALPAKA_FN_ACC void operator()(
      Acc const& acc,
      const ChipsSoAFlat& chipsSoA, // Pass ChipsSoA struct
      int chipIdx,
      int* clusterResults) const {

    // Determine which pixel this thread is responsible for
    auto const pixelIdx = alpaka::getIdx<alpaka::Grid, alpaka::Threads>(acc)[0]; // Thread index is the pixel index

    // Ensure the pixel index is valid
    int chipOffset = chipsSoA.getChipOffset(chipIdx);
    int nextChipOffset = chipsSoA.getChipOffset(chipIdx + 1);
    const auto nPixels = nextChipOffset - chipOffset;
    if (pixelIdx >= nPixels) {
      return;
    }

    // Example logic: Set clusterResults based on masking
    clusterResults[pixelIdx] = chipsSoA.isMaskedFlat[chipsSoA.getChipOffset(chipIdx) + pixelIdx] ? 1 : 0;
  }
};
```

Main Accelerators supported by alpaka



Accelerator	Description	Use Case
AccCpuSerial	Serial execution on a single CPU thread.	For simple testing or environments without parallel processing capabilities.
AccCpuThreads	Parallel execution using native C++ threads.	Multithreaded execution on a CPU for parallel workloads.
AccCpuOmp2Threads	Parallel execution using OpenMP threads.	For leveraging OpenMP parallelization on multi-core CPUs.
AccCpuOmp2Blocks	Parallel execution using OpenMP blocks.	Efficient execution when dividing workloads into block units.
AccGpuCudaRt	GPU execution using CUDA runtime.	High-performance GPU execution on CUDA-capable devices.
AccGpuHipRt	GPU execution using HIP runtime.	For AMD GPUs, offering performance portability similar to CUDA.
AccFpgaSyclIntel	FPGA execution using Intel's SYCL implementation.	Designed for FPGA workloads, providing high performance in hardware-specific tasks.

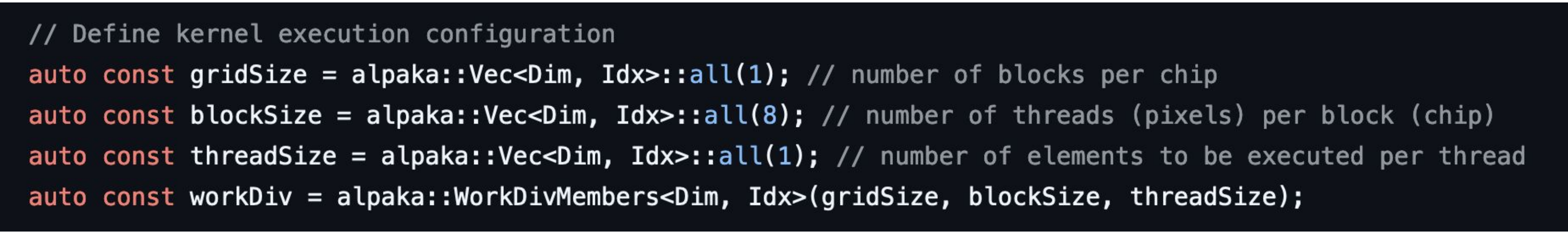
First performance evaluation

On (my personal) **Apple M2** processor (no NVIDIA/ARM GPU), I successfully tested:

1. **Legacy (serial) O2 code**
2. Alpaka execution with **AccCpuSerial**, for development
3. Alpaka execution with **AccCpuThreads to introduce a first parallelism** (one thread per pixel)

Preliminary results:

- **Exact reproduction** of output from legacy code
  - 296462 digits, in 45 RO frames -> 40994 clusters reconstructed
- **execution 2 is 20% faster than execution 1**
- currently investigating different configurations for execution 3:



```
// Define kernel execution configuration
auto const gridSize = alpaka::Vec<Dim, Idx>::all(1); // number of blocks per chip
auto const blockSize = alpaka::Vec<Dim, Idx>::all(8); // number of threads (pixels) per block (chip)
auto const threadSize = alpaka::Vec<Dim, Idx>::all(1); // number of elements to be executed per thread
auto const workDiv = alpaka::WorkDivMembers<Dim, Idx>(gridSize, blockSize, threadSize);
```

Exploit parallelism

- The CPU accelerators only allows for limited parallelization (CPU threads)
- The access to GPU accelerators allows to increase the explore higher dimensions of parallelization.
- Unfortunately, the building of Alice software on lxplus is quite difficult (currently in contact with Giulio Eulisse)

ClusterKernel 2D

```
struct ClusterKernel_2D {
    template <typename Acc>
    ALPAKA_FN_ACC void operator()(
        Acc const& acc,
        const ChipsSoA& chipsSoA, // The ChipsSoA structure containing the isMasked data
        int* clusterResults) const
    {
        // Block index represents the chip
        auto chipIdx = alpaka::getIdx<alpaka::Grid, alpaka::Blocks>(acc)[0];
        // Thread index within block processes pixels in the chip
        auto pixelIdx = alpaka::getIdx<alpaka::Block, alpaka::Threads>(acc)[0];

        // Ensure the chip index is valid
        if (chipIdx >= chipsSoA.isMasked.size()) {
            return;
        }

        // Get the current chip
        auto const& chip = chipsSoA.isMasked[chipIdx];
        auto const nChipPiels = chip.size();
        // Ensure the pixel index is valid
        if (pixelIdx >= nChipPiels) {
            return;
        }

        // Set clusterResults based on masking
        clusterResults[pixelIdx] = chip[pixelIdx] ? 1 : 0;
    }
};
```

Conclusions

1. **Successfully integrated alpaka within AliceO2 framework** (compilation details in the backup): *https://github.com/lecriste/AliceO2/tree/alpaka_its*
2. Defined a first SoA to store pixels informations belonging to all chips in a ROF
3. Designed a kernel to **filter masked pixels before clustering**.
4. Successful execution of alpaka kernel on two CPU accelerators
5. First performance benchmark is promising

A big thanks to **Matteo Concas** for helping a non-Alice member with the initial support in setting up the Alice framework!

Future plans

- Deploy the new framework on an environment equipped with GPUs
  - Introduce GPU accelerators and evaluate performance
- Add data members to the `ChipsSoA`
- Extend kernel to perform more clustering steps by exploiting the augmented `ChipsSoA`
  - Consider performing the clustering across all pixels of all ROFs in a TimeFrame, and not by single ROF
- Once a meaningful speed-up is achieved, perform large scale tests
- Implement feedback received at **Riunione Calcolo of the last ALICE Italia meeting**

Based on the results of the above:

- Expand alpaka integration to other reconstruction modules
- Open to discussion…

CINECA HPC “**Leonardo**”

Some Leonardo nodes have been configured with access to cvmfs

- I prepared a set of instructions to execute a full CMS workflow leveraging GPUs: https://github.com/ICSC-Spoke2-repo/benchmark-heterogeneous-algorithms

Heterogeneous algorithms on ICSC resources

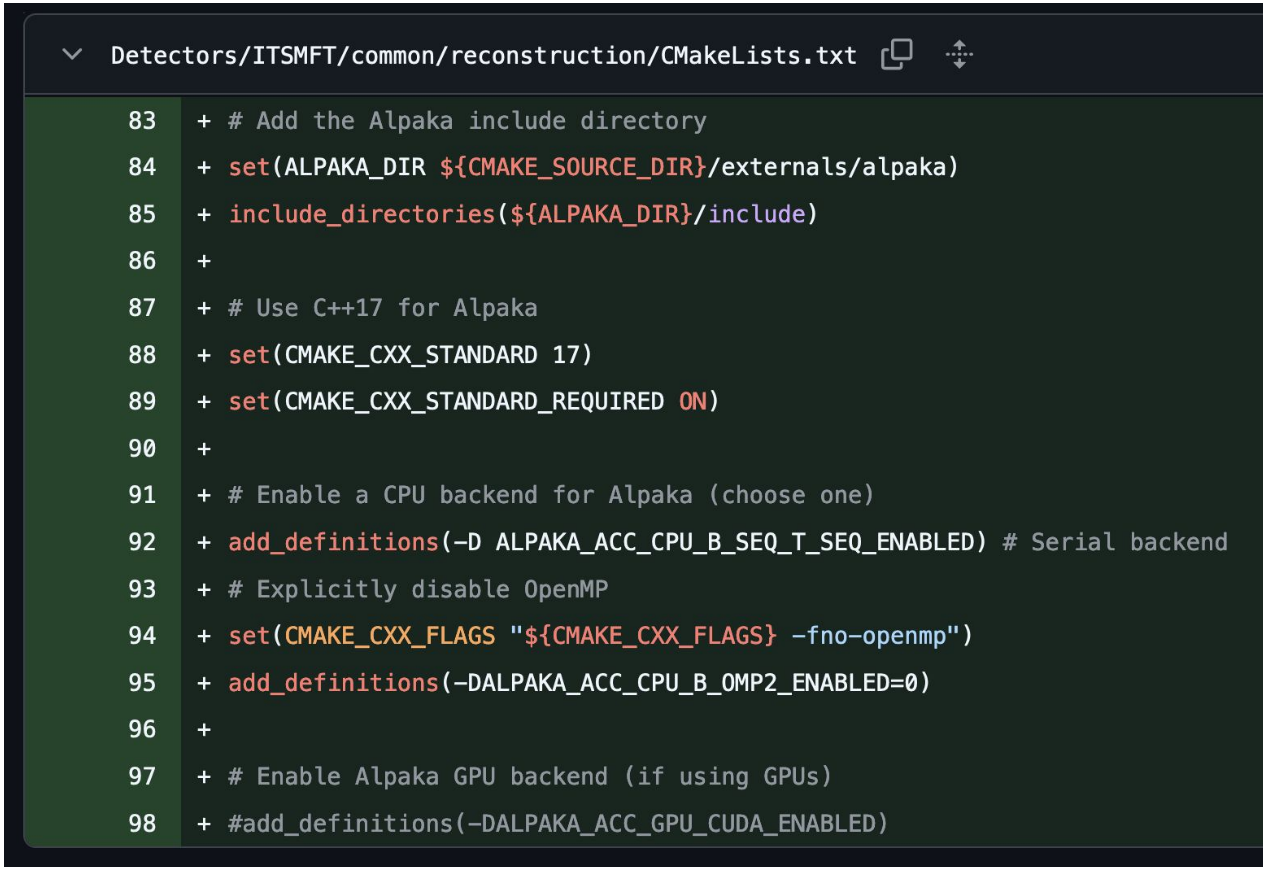
This repository is intended to provide guidance for executing algorithms leveraging heterogeneous resources.

Leonardo HPC

A full CMS workflow leveraging alpaka - e.g. `TTbar_14TeV+2024_Patatrack_FullRecoAlpaka` - can be executed on the CINECA HPC centre "Leonardo" with the following steps:

Backup

Added compilations flags:



Detectors/ITSMFT/common/reconstruction/CMakeLists.txt

```
+ # Add the Alpaka include directory
+ set(ALPAKA_DIR ${CMAKE_SOURCE_DIR}/externals/alpaka)
+ include_directories(${ALPAKA_DIR}/include)
+
+ # Use C++17 for Alpaka
+ set(CMAKE_CXX_STANDARD 17)
+ set(CMAKE_CXX_STANDARD_REQUIRED ON)
+
+ # Enable a CPU backend for Alpaka (choose one)
+ add_definitions(-D ALPAKA_ACC_CPU_B_SEQ_T_SEQ_ENABLED) # Serial backend
+ # Explicitly disable OpenMP
+ set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fno-openmp")
+ add_definitions(-DALPAKA_ACC_CPU_B_OMP2_ENABLED=0)
+
+ # Enable Alpaka GPU backend (if using GPUs)
+ #add_definitions(-DALPAKA_ACC_GPU_CUDA_ENABLED)
```