ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Informatica

PERFORMANCE PORTABILITY OF PHYSICS RECONSTRUCTION ALGORITHMS ON HETEROGENEOUS DEVICES USING THE SYCL ABSTRACTION MODEL

Relatore: prof. **Renzo Davoli**

Correlatori: dr. **Francesco Giacomini – INFN-CNAF**
dr. **Andrea Bocci – CERN**

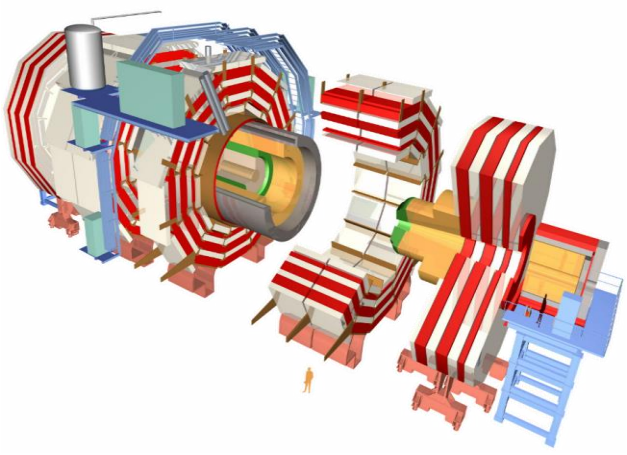
Presentata da:

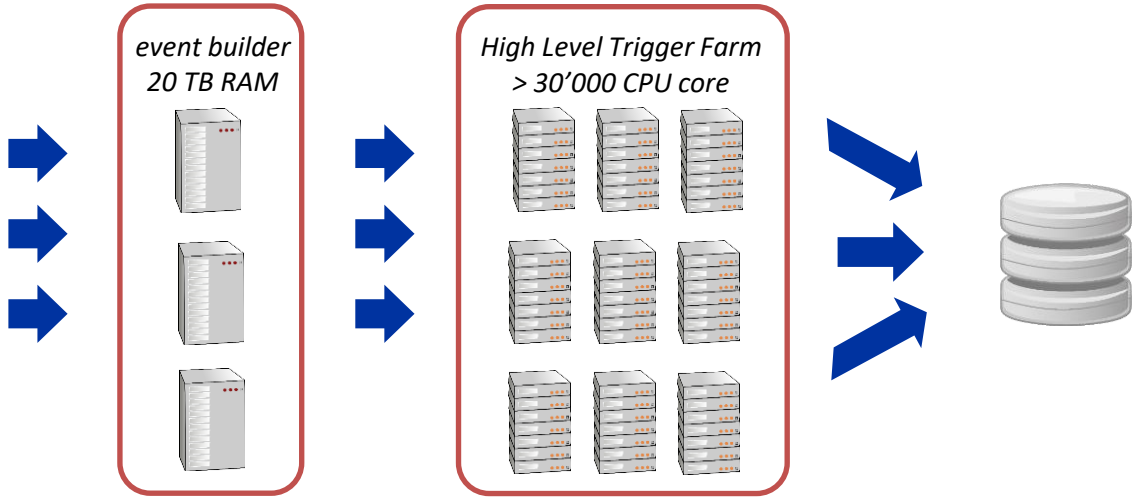
Laura Cappelli

II Sessione – Secondo Appello

A.A. 2019-2020

CMS Trigger & DAQ





Data Acquisition

Many complex layers detect particles produced by a high energy proton-proton collision

30 MHz

Level 1 Trigger

hardware based (FPGA and custom electronic) synchronous with LHC

100 kHz

High Level Trigger

Software based (runs on Event Builder and High Level Trigger farm)

On-demand reconstruction and event selection

1 kHz

Storage Manager

Distributed filesystem

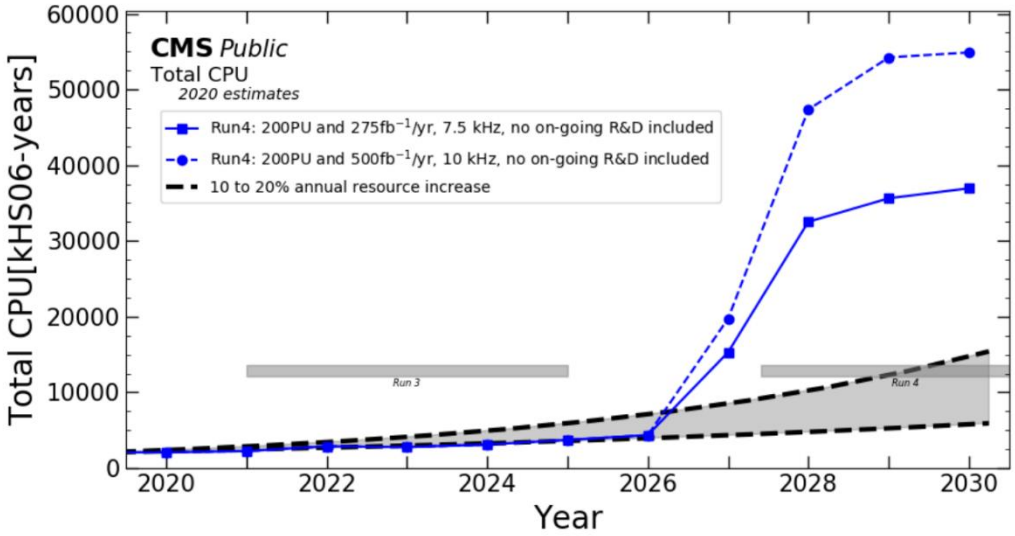
Data are made available for offline analysis

The problem of computing power

- The higher the frequency of collisions, the higher the chance of we observe rare events
  - *High-Luminosity LHC*
  - An update of hardware and software is needed

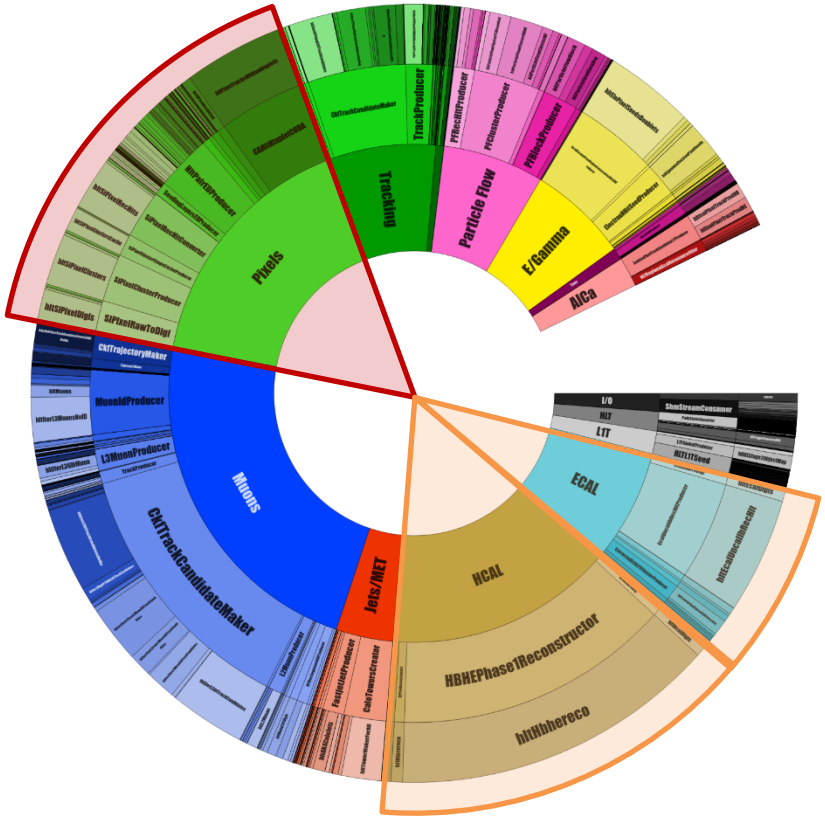
- Notes:
  - Addressing only CPU architectures in not sufficient
  - The highest performance/price ratio is expected to come from **accelerators** (GPU and FPGA)

- We need a new software technology to run code on **heterogeneous architectures** with convenient performance on each of them



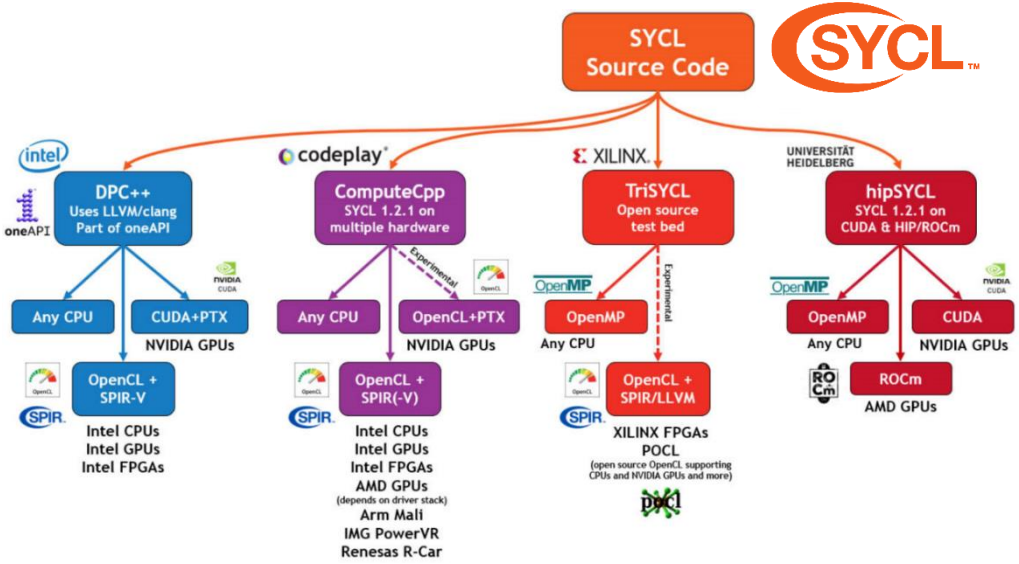
Patatrack

- The Patatrack project has the goal to show that CMS software (CMSSW) could take advantages of accelerators
  - The highlighted slices of the pie chart have been rewritten in CUDA to run on GPUs
- A test code has been created with the purpose to study the performance portability of other technologies (SYCL/oneAPI, Kokkos, Alpaka)
  - CMSSW lite software framework + Pixel modules
  - Avoid the complexity and dependencies of the full CMSSW, while maintaining the same characteristics
  - [Standalone Patatrack pixel tracking]



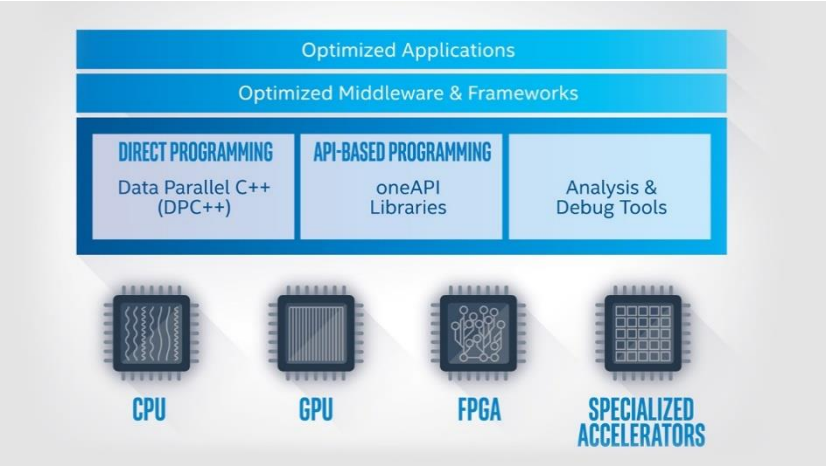
SYCL

- **SYCL** in a Khronos Group standard:
  - It defines a new abstract and high-level programming model for a heterogeneous context
  - *Single-source*
  - Conformant to ISO C++
- SYCL implements and extends the OpenCL execution model
  - It expects one ***host*** and one or more devices
  - It runs on devices some blocks of code called ***kernels***
- Four available implementations
  - **Data Parallel C++**
  - ComputeCPP
  - TriSYCL
  - hipSYCL



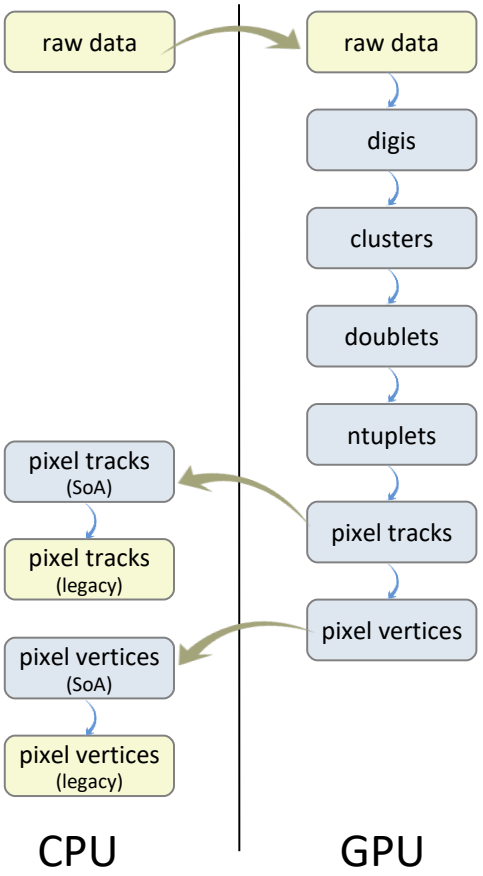
Intel oneAPI and DPC++

- **oneAPI** is a new open programming model developed by Intel
  - Introduce a unified parallel programming model to simplify heterogeneous programming
  - Implement the SYCL standard and extend it with other tools for making oneAPI like CUDA approach
  - A promising technology for portability across different types of accelerators from different vendors



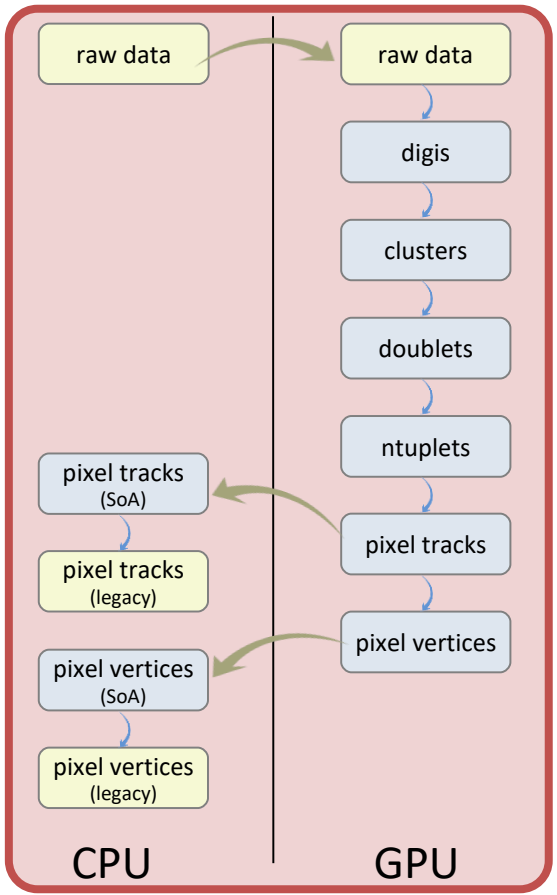
- oneAPI **includes**:
  - The language DPC++
    - Open-source implementation based on LLVM
    - Mostly developed by Intel
    - CodePlay contribution for CUDA support
  - Libraries
  - Tools like Data Parallel Compatibility Tool
  - Level Zero
    - Low-level hardware interface
    - Other backends (OpenCL, CUDA) are supported

SYCL and Pixeltrack Standalone



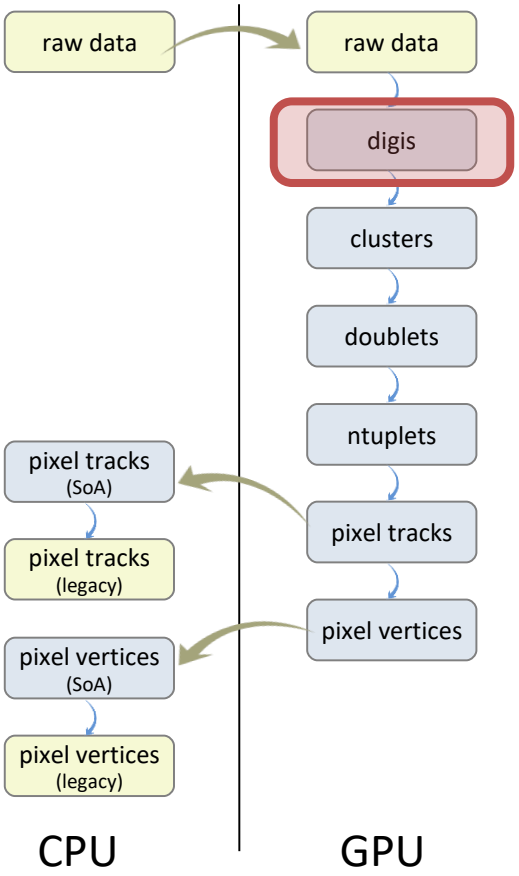
- The full workflow
  - Copy the raw data to the GPU or to other accelerators
  - Run multiple kernels to perform the various steps for the pixel-based tracks and vertices reconstructions
  - Copy only the final results back to the host
- Project goals:
  - Framework and modules conversion from CUDA to SYCL DPC++
  - Test the code on different devices and back-ends
  - Compare the performance of the SYCL “port” to that of the native implementation and verify if they are convenient on each device
- Work done until December:
  - Framework conversion
  - *digis* module conversion
  - *clusters* module conversion

Framework conversion



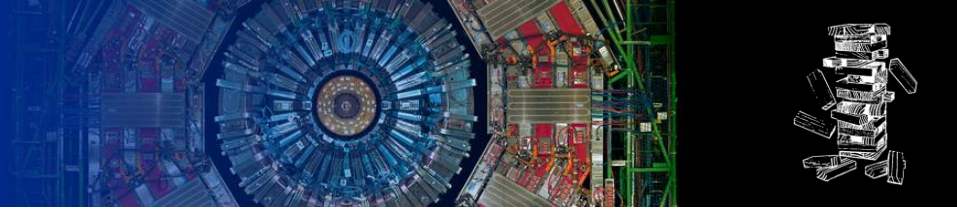
- From native CUDA to SYCL DPC++ with the oneAPI compatibility tool
  - Successfully converted code (80%)
    - CUDA stream → in-order SYCL queues
    - CUDA memory instructions → DPC++ *Unified Shared Memory* instructions
    - CUDA synchronization operations → SYCL synchronization operations
  - Code not converted correctly (20%)
    - Events:
      - cannot create not-ready events associated to devices
      - wrap events into `std::optional<sycl::event>`
    - Memory allocations:
      - CUDA version uses a caching allocator based on CUB to improve performance, not implemented yet for the SYCL version
- DPC++ improves the code design:
  - SYCL queues include device information, CUDA streams don't include it

Digis conversion



- From native CUDA to SYCL DPC++ with the oneAPI compatibility tool
  - Successfully converted code (80%)
  - Code not converted correctly (20%)
    - Events
    - Memory allocations
    - Error handling
    - Print instruction inside a kernel
      - SYCL streams vs `printf`
      - An additional parameter has to be passed to every kernel function

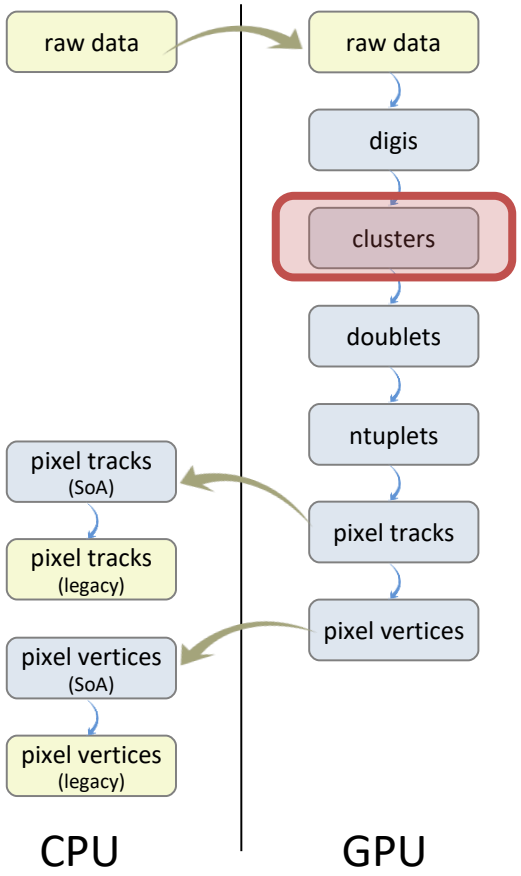
Performance



- The data in this slide don’t show meaningful performance results
  - The focus is on testing the syntax and the correctness of the code
- The values are measured on different devices and back-ends; they are made on Intel DevCloud
  - CPU: Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz, OpenCL driver version 2020.11.10.0.05
  - GPU: Gen9 HD Graphics integrated GPU, OpenCL and Level 0 NEO driver version 20.41.18123
- Each result shows the average time spent to process a single event
  - The values are the average of 10 measures

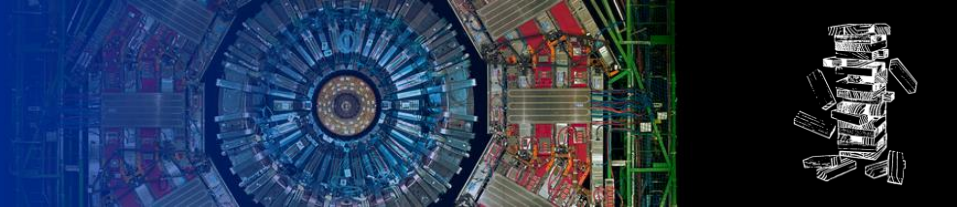
Dispositivo	**CPU nativo**	**CPU**	**GPU**	**GPU**	**SYCL HOST**	**FPGA emu**
Interfaccia di back-end	*Single thread*	*OpenCL*	*OpenCL*	*Level Zero*	-	*OpenCL*
Work groups	1	9	24	24	1	9
Work items per group	1	4096	256	256	1	4096
Tempo di processamento per evento (ms)	**0.45 ± 0.01**	**17.07 ± 0.03**	**2.67 ± 0.01**	**4.75 ± 0.04**	**3.6 ± 0.2**	**13.0 ± 0.7**

Cluster conversion



- From native CUDA to SYCL DPC++ with the oneAPI compatibility tool
  - Successfully converted code (80%)
  - Code not converted correctly (20%)
- Future developments and encountered problems
  - A library version of ***prefixScan***, a part of the code, was recently released
    - Compare the performance of these two versions
  - The subgroup size must be set on compile time
    - It isn't possible to write a parameterized program
    - CPU and GPU have different max subgroup sizes
    - Try template classes with explicit subgroup sizes
  - The client code blocks and leaves the GPU driver in a frozen state
    - reproduced locally and on Intel DevCloud
    - reported to Intel, working to isolate the cause and find a fix or a workaround

Summary



- Achieved goals:
  - The conversion from CUDA to SYCL is possible
  - The same SYCL code can be compiled and run on heterogeneous devices
- Difficulties encountered during the project:
  - The Data Parallel C++ compiler is under development
    - The documentation is poor or absent
    - The implementation isn't stable
- Conclusions:
  - We are still testing the syntax and the correctness of the code
  - It's too early to give judgments about the performance
- Future developments:
  - Complete the code conversion
  - Optimize the software
  - Compare the performance portability between SYCL, Kokkos and Alpaka versions of Pixeltrack Standalone