EPJ Web of Conferences **337**, 01087 (2025) https://doi.org/10.1051/epjconf/202533701087
CHEP 2024

SYCL-based online data processing framework concept for $\overline{\text{P}}$ANDA

Bartosz Soból[1,2,*] and *Grzegorz* Korcyl[1,**]

[1]Department of Information Technologies, Jagiellonian University, Prof. Łojasiewicza 11, Kraków, Poland

[2]Doctoral School of Exact and Natural Sciences, Jagiellonian University, Prof. Łojasiewicza 11, Kraków, Poland

Abstract. The $\overline{\text{P}}$ANDA experiment has been designed to incorporate software triggers and online data processing. Although $\overline{\text{P}}$ANDA may not surpass the largest experiments in terms of raw data rates, designing and developing the processing pipeline and software platform for this purpose is still a challenge. Given the uncertain timeline for $\overline{\text{P}}$ANDA and the constantly evolving landscape of computing hardware, our attention is directed toward ensuring the future-proofness of the solutions we develop.

The PandaR2 is a concept for a framework handling online data processing in heterogeneous and distributed HPC environments. It utilizes the SYCL programming model as the primary technology for parallelization and offloading. Being a new and standalone entity, PandaR2 also interfaces with the $\overline{\text{P}}$ANDA's original ROOT-based simulation and analysis framework - PandaRoot, connecting the best of both worlds.

This contribution aims to present an overview of the PandaR2 SYCL-centric architecture. We will share experiences with SYCL during the codebase design process, particularly highlighting its portability across various hardware platforms and compilers. Additionally, we will showcase the performance results of the initial algorithms implemented in PandaR2.

1 Introduction

1.1 The $\overline{\text{P}}$ANDA experiment

$\overline{\text{P}}$ANDA (or antiProton ANnihilations at DArmstadt) is a hadron physics experiment planned as one of the major parts of the Facility for Antiproton and Ion Research (FAIR) [1]. It will use antiproton beam interactions with hydrogen target to study topics such as charmonium and open charm mesons, search for gluonic excitations, and spectroscopy of charmed and strange bosons [2].

Currently, various detector subsystems of $\overline{\text{P}}$ANDA are being developed and tested as temporary setups within other experiments and are waiting for the final construction phase.

*e-mail: bartosz.sobol@doctoral.uj.edu.pl
**e-mail: grzegorz.korcyl@uj.edu.pl

1.2 Software and online processing

As with many other modern particle physics experiments, $\overline{\text{P}}$ANDA will use a data acquisition system with a so-called software trigger. It will need to digest the raw experimental data incoming from all subsystems at a rate of up to 200 GB/s and process it with low latency using various specialized algorithms [3]. Such a setup requires a specific infrastructure and software solutions supporting operation in a distributed environment and taking advantage of multicore CPUs, GPGPU devices, and potentially FPGA and other accelerators.

The existing simulation, reconstruction, and analysis software stack of $\overline{\text{P}}$ANDA comprises the PandaRoot [4] framework. It is built on top of the FairRoot framework [5], an FAIR in-house developed simulation, reconstruction, and analysis framework for particle physics experiments also used by other experiments inside (CBM [6]) and outside of FAIR (Alice [7]). PandaRoot and FairRoot are both built in C++ programming language and heavily based on ROOT [8] infrastructure. Currently, simulation data produced with PandaRoot is used to develop and evaluate reconstruction and analysis algorithms.

PandaRoot is a classic offline-focused platform developed without support for parallelism and GPU acceleration, making it unsuitable for online processing applications. Adapting it to online processing requirements would be costly and problematic. Hence, the need for a new online-ready and future-proof solution has emerged.

2 Technologies for accelerated online processing

2.1 SYCL

SYCL [9] is an open standard for a high-level programming model for multicore CPUs and compute accelerators. Its primary objective is to offer a fundamental API that guarantees code portability, allowing a single application to utilize various hardware targets, such as CPUs, GPUs, or FPGAs from leading vendors.

The SYCL APIs are based on standard C++17 and don't include any language extensions. They employ a traditional approach, where device code is encapsulated in separate procedures known as kernels. These kernels are submitted to the SYCL device queue as C++ functors. Both host and device code are maintained within a unified codebase. The SYCL compiler is responsible for discerning device code and converting it into a format suitable for the target device.

SYCL's default memory management employs a buffer-accessor model, where buffer objects manage the underlying memory and data ownership. Access to data is facilitated through accessor objects that have pre-defined read-write permissions. By utilizing these accessor properties, the SYCL runtime can construct a dependency graph of kernels and automatically schedule data transfers between the host and accelerators. Additionally, with the introduction of SYCL 2020, there is now an alternative pointer-based memory management system known as Unified Shared Memory (USM). This mechanism is reminiscent of approaches found in other kernel-based programming models, such as CUDA. It allows for explicit memory management, providing fine-grained control in memory-bounded applications.

Potential alternatives to SYCL include several kernel-based programming models, such as Kokkos [10] and Alpaka [11], as well as directive-based models like OpenMP [12]. The former are abstraction layer libraries with base concepts of the API close to SYCL. However, SYCL as a standard can be implemented both as a library [13] and on the compiler level [14], which can benefit performance and lower-level features support. The availability of multiple implementations provides a higher level of confidence in the long-term availability of the solution. Additionally, directive-based models can be used alongside SYCL, especially for less complex problems where their lower verbosity is advantageous.

2.2 FairMQ

FairMQ [15] is a message queuing library developed at GSI/FAIR. It is specifically designed for large-scale data movement in distributed computing environments. The framework provides an abstract transport layer that supports ZeroMQ, shared memory, and potentially RDMA communication. It also includes various utilities, such as logging, configuration management, and a command-line interface, making it highly adaptable for large-scale use. FairMQ has already been successfully implemented in the ALICE experiment.

3 The PandaR2 project

3.1 Overview

The PandaR2 framework has been designed for the purpose of handling online data processing in the $\overline{\text{P}}$ANDA experiment. All experiments included in the FAIR facility will utilize a common high-performance computing (HPC) center located on campus. Therefore, the software solutions used by experiments should be universal and, as far as possible, adaptable to current and future architectures of commodity HPC installations.

Parallel processing is a key feature of PandaR2, with SYCL chosen as the primary programming model to enable seamless execution across both CPUs and GPU accelerators. Additionally, FairMQ is integrated into the framework to enable high-performance messaging between processing stages.

Unlike classic software packages used in particle physics, such as PandaRoot, PandaR2 eliminates dependencies on the ROOT framework in the computing layer. This makes it more suitable for accelerated execution. To maintain compatibility with PandaRoot, PandaR2 includes a middleware layer that allows PandaR2-based tasks to be invoked from PandaRoot, ensuring seamless interoperability between the two frameworks. Furthermore, PandaR2 stays connected to the ROOT ecosystem by supporting dictionaries for essential data classes and providing input and output capabilities with ROOT files.

3.2 System architecture

PandaR2 aims at supporting various system configurations, ranging from simple single-node setups to complex multi-node distributed environments. In a basic configuration shown in Figure 1, a feeder process reads input data from a ROOT file and transmits it via FairMQ to a worker for processing. The processed data is then sent to a sink device for storage.

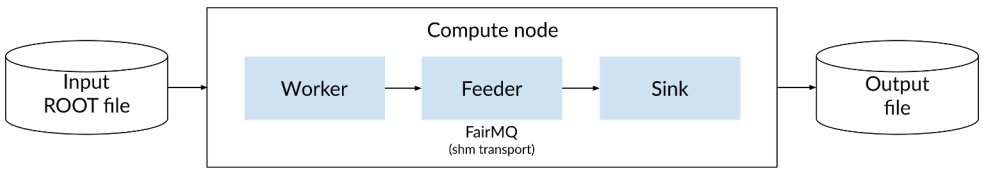
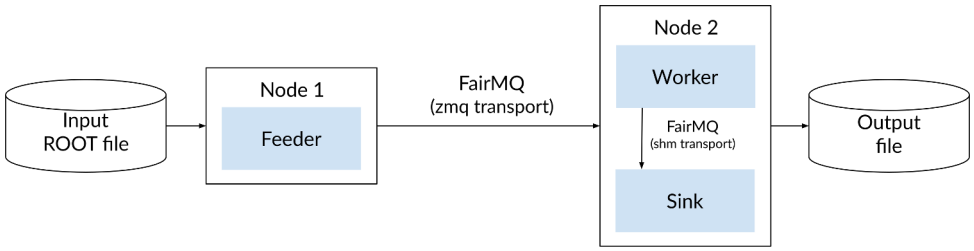
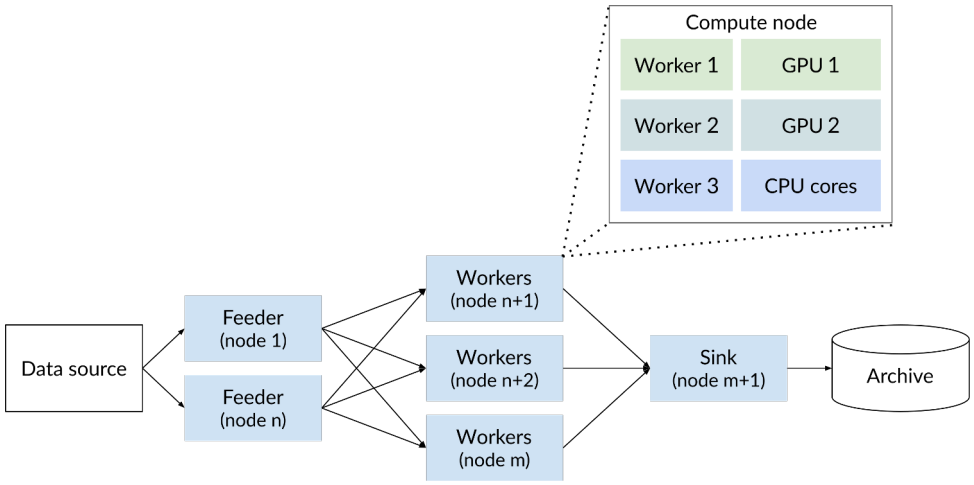


Figure 1: PandaR2 basic single-node deployment

Figure 2 presents examples of distributed setups. A simple linear deployment similar to the previously mentioned basic configuration, but split into two separate compute nodes, is shown in Figure 2a. Figure 2b shows a scheme of a more complex setup. It involves multiple feeders distributing data to several nodes running numerous worker processes, each executing different processing tasks. The results are eventually sent to sink devices for archival.



(a) simple two-node setup



(b) complex multi-node setup

Figure 2: PandaR2 distributed deployments

The ultimate goal is to integrate PandaR2 into the complete $\overline{\text{P}}$ANDA experimental infrastructure, with low-level DAQ servers providing data into preprocessing servers serving as feeders connected to a high-performance compute cluster running PandaR2. A diagram of the potential architecture of such a system is shown in Figure 3.

3.3 Implementation

The PandaR2 platform consists of two main parts: the core framework and a $\overline{\text{P}}$ANDA detector implementation. The framework provides interfaces for task functionalities and implements core and common components, such as data management layer, detector geometry abstractions, etc., many of which are in the early stages of development. Detector implementation includes anything related to the execution of specific tasks for detector subsystems, such as hit finding or particle tracking algorithms for individual $\overline{\text{P}}$ANDA subsystems. It uses PandaR2 core features and forms a modular task library. This high-level architecture of PandaR2 is shown in Figure 4.

The PandaR2 concept assumes deep integration with the SYCL programming model for all tasks that require CPU parallelism or GPU/FPGA acceleration. However, we foresee using the OpenMP [12] model for specific, less complex tasks.

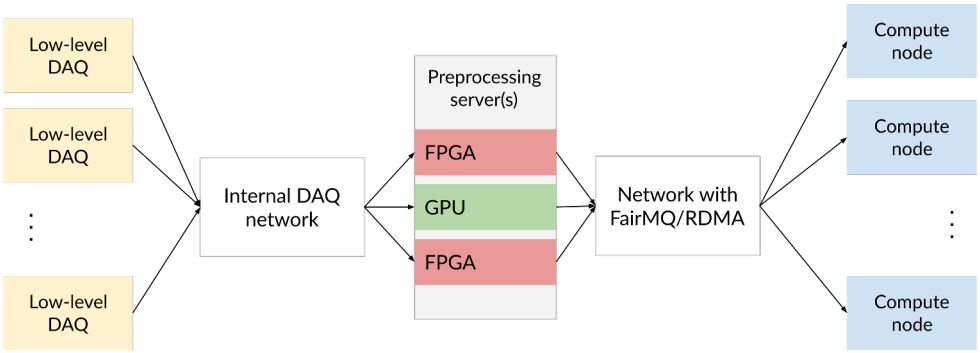


Figure 3: PandaR2 experimental setup connected to DAQ system

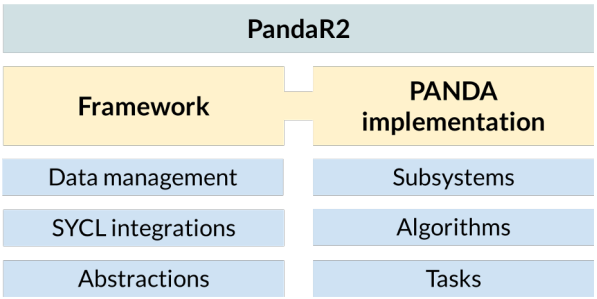


Figure 4: PandaR2 structure

3.3.1 SYCL integration

The SYCL standard introduces a concept of a `queue` associated with a physical or logical device and manages the execution of tasks on that device. Queues are typically constructed given a specific `device` entity or a device `selector`. Device selectors are callables that, given a list of devices, determine which will be managed by the queue.

The first PandaR2-specific features built on top of the SYCL API are a custom queue manager and device selector. The PandaR2 queue manager encapsulates a list of all SYCL queues used in the R2 worker instance. Tasks performed by the worker can request a specific queue with a suitable selector. If necessary, the manager can be configured to hold multiple queues per device or to assign a particular queue to a specific task. Additionally, the custom device selector allows for device selection based on priority *hints* provided for each device type (CPU, GPU, or accelerator) at runtime.

3.3.2 Use of FairMQ

PandaR2 leverages FairMQ for the data transport layer. The design of PandaR2 includes three generic FairMQ `devices` (i.e., FairMQ connected processes): `r2feeder`, `r2sink` and `r2daemon`. While the role of the first two `devices` is represented in their name and described in Section 3.2, the latter is responsible for executing one or multiple data processing tasks. All can be configured for the requirements of a certain deployment, run in multiple instances, and connected into the desired topology.

4 First evaluation

4.1 Experimental setup and execution

For the first evaluation of the PandaR2 system, the experimental setup reflecting the simple two-node setup from Figure 2a has been set up. Two network-connected compute nodes communicated via FairMQ and ran the standard feeder and worker + sink processes.

In the setup, the feeder was reading data from the Monte Carlo-generated simulations of $\overline{\text{P}}$ANDA detector and publishing it to the Worker. For the worker, we used the most established data processing task implemented in PandaR2 - particle track finding algorithm for the Forward Tracker subsystem in $\overline{\text{P}}$ANDA - the `FtsTrackFinder`. Finally, the sink was responsible for saving serialized processing results to a file. The stable execution of the two-node setup was the first step to expanding the development into more complex use cases.

The compute node running the worker was equipped with NVIDIA RTX 4090 and AMD Radeon Pro W7800 GPUs and an AMD EPYC 7F52 CPU. PandaR2 was compiled with GCC 11. AdaptiveCpp version 24.10, built with LLVM 18, was used as an SYCL compiler.

4.2 Performance results

In addition to evaluating the system functionally, further benchmarks of the worker running `FtsTrackFinder` task were conducted. Figure 5 shows the throughput of detector events processed per second on all platforms available during the test.

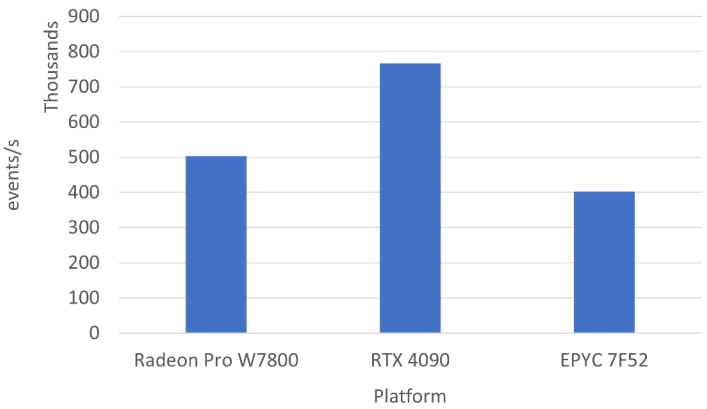


Figure 5: `FtsTrackFinder` task throughput

All runs were performed using AdaptiveCpp `generic` backend and, in case of the AMD EPYC 7F52 CPU, on all 16 cores and 32 threads. The presented results are consistent with the standalone execution of the track reconstruction algorithm outside the PandaR2 environment. An in-depth analysis of the algorithm behind `FtsTrackFinder` and its performance is the subject of another contribution currently under publisher review.

5 Conclusions

The development of PandaR2 has already established its base components, including abstractions, integration with SYCL and FairMQ, and a middleware connection with PandaRoot.

The first tracking algorithms have been successfully implemented, demonstrating the framework's capabilities.

Current efforts focus on expanding the reconstruction pipeline by implementing and optimizing additional algorithms. Another significant development area is mastering FairMQ for more complex deployment topologies and efficient data transfer across distributed nodes. Infrastructure improvements, such as parameter management and system monitoring, are also on the roadmap.

Future developments will include implementing a simulated DAQ system to test and refine data formats and feeder functionality. Establishing an experimental test setup with real hardware, similar to CBM's mCBM, will be an important milestone in validating the framework under realistic conditions.

The development of PandaR2 marks a significant advancement in real-time data processing for the $\overline{\text{P}}$ANDA experiment. By leveraging modern technologies such as SYCL for parallel execution and FairMQ for distributed communication, PandaR2 aims to provide a scalable, efficient, and future-proof solution for processing large-scale experimental data. Continued development efforts will focus on further optimization, algorithmic improvements, and seamless integration into the broader $\overline{\text{P}}$ANDA infrastructure, ensuring its readiness for the experiment's operational phase.

References

[1] Gutbrod, Hans H., International Facility for Antiproton and Ion Research (FAIR) at GSI, Darmstadt., Nuclear Physics A pp. 457–469 (2005). 101016/jnuclphysa200502137

[2] Brinkmann, K. T. and Gianotti, P. and Lehmann, I., Exploring the mysteries of strong interactions: The PANDA experiment, Nucl. Phys. News **16**, 15 (2006), physics/0701090. 10.1080/10506890600579868

[3] A. Malige, G. Korcyl, M. Firlej, T. Fiutowski, M. Idzik, B. Korzeniak, R. Lalik, A. Misiak, A. Molenda, J. Moroń et al., Real-time data processing pipeline for trigger readout board-based data acquisition systems, IEEE Transactions on Nuclear Science **69**, 1765 (2022). 10.1109/TNS.2022.3186157

[4] S. Spataro (for the PANDA collaboration), The PandaRoot framework for simulation, reconstruction and analysis, Journal of Physics Conference Series **331** (2011). http://dx.doi.org/10.1088/1742-6596/331/3/032031

[5] M. Al-Turany, D. Bertini, R. Karabowicz, D. Kresan, P. Malzacher, T. Stockmanns, F. Uhlig, The fairroot framework, Journal of Physics: Conference Series **396**, 022001 (2012). 10.1088/1742-6596/396/2/022001

[6] E. Lavrik, o. Collaboration, Compressed baryonic matter experiment at fair, AIP Conference Proceedings **2163** (2019). 10.1063/1.5130095

[7] Eulisse, Giulio, Rohr, David, The o2 software framework and gpu usage in alice online and offline reconstruction in run 3, EPJ Web of Conf. **295**, 05022 (2024). 10.1051/epjconf/202429505022

[8] R. Brun, F. Rademakers, Root — an object oriented data analysis framework, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **389**, 81 (1997), new Computing Techniques in Physics Research V. https://doi.org/10.1016/S0168-9002(97)00048-X

[9] SYCL Specificcation, Accessed: 02-28-2025, https://www.khronos.org/sycl/

[10] C.R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D.S. Hollman, D. Ibanez et al., Kokkos 3: Programming model extensions

for the exascale era, IEEE Transactions on Parallel and Distributed Systems **33**, 805 (2022). 10.1109/TPDS.2021.3097283

[11] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W.E. Nagel, M. Bussmann, Alpaka - An Abstraction Library for Parallel Kernel Acceleration (IEEE Computer Society, 2016), 1602.08477, http://arxiv.org/abs/1602.08477

[12] L. Dagum, R. Menon, Openmp: An industry-standard api for shared-memory programming, IEEE Comput. Sci. Eng. **5**, 46–55 (1998). 10.1109/99.660313

[13] A. Alpay, SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL, in *IWOCL '20: Proceedings of the International Workshop on OpenCL* (2020), https://doi.org/10.1145/3388333.3388658

[14] oneAPI, Intel® oneAPI DPC++/C++ Compiler, Accessed: 01-20-2025, https://www.intel.com/content/www/us/en/docs/dpcpp-cpp-compiler/developer-guide-reference/2025-0/overview.html

[15] M. Al-Turany, D. Klein, T. Kollegger, A. Rybalchenko, N. Winckler, Fairmq (2025), https://doi.org/10.5281/zenodo.14726773