Computing and Software for Big Science (2025) 9:3
https://doi.org/10.1007/s41781-025-00131-w

RESEARCH

Parallelized Event Data Management System Based on MT-SNiPER Framework and PODIO

Qianqian Shi[1,2] · Teng Li[1,2] · Xingtao Huang[1,2]

Received: 14 September 2024 / Accepted: 20 January 2025

Abstract
Software framework serves as a skeleton for the offline data processing software for many high energy physics (HEP) experiments. The event data management, including the event data model (EDM), transient event store and data input/output, implements the core functionalities of the framework, and has a great impact on the performance of the entire offline software. Future HEP experiments are generating increasingly large amounts of data, bringing challenges to offline data processing. To address this issue, a common event data management system that supports efficient parallelized data processing applications has been developed based on Software for Non-collider Physics ExpeRiments (SNiPER) common software framework as well as PODIO, a common EDM toolkit for future HEP experiments. In this paper, the implementation of a parallelized event data management (PEDM) system is introduced, including the integration with MT-SNiPER and PODIO, as well as the implementation of GlobalStore to support multi-threaded event processing. Finally, the application and performance evaluation of the data management system in OSCAR (offline software of Super Tau Charm Facility) is presented.

Keywords Offline data processing · SNiPER · High energy physics · Event data model

Introduction

Offline data processing software, such as Athena [1] for ATLAS [2] and BOSS [3] for BESIII [4], is usually a critical component of modern high energy physics (HEP) experiments, responsible for building the offline data processing chain including detector simulation, calibration, reconstruction, and data analysis. The underlying framework, such as Gaudi [5], builds the foundation of the offline software, providing basic functionalities like event loop control, algorithm scheduling, event data management, detector description management, and some key common services.

The challenges in non-collider experiments such as nuclear reactor neutrino and cosmic ray experiments involve handling events with rare physics signals and correlation analysis between events within certain time windows, which are pretty different from collider experiments [6]. Learning the concepts of algorithm and service from Gaudi, and considering the needs of non-collider experiments, Chinese developers implemented the SNiPER framework [7]. Now SNiPER has been adopted by several experiments, including Jiangmen Underground Neutrino Observatory (JUNO) [8], Large High Altitude Air Shower Observatory (LHAASO) [9], Neutrinoless double beta decay experiment (nEXO) [10]. SNiPER is very lightweight and easy-to-use. It supports flexible event processing sequences and custom data management. As a general-purpose offline software framework, SNiPER is also suitable for collider experiments [11], offering customizability, extensibility, and inherent advantages in parallel computing. Therefore, we chose SNiPER as the underlying framework for the parallelized event data management (PEDM) system.

The size of experimental data used for physics research is sharply increasing (see Table 1), which is

✉ Teng Li
tengli@sdu.edu.cn

✉ Xingtao Huang
huangxt@sdu.edu.cn

Qianqian Shi
shiqq@mail.sdu.edu.cn

1 Institute of Frontier and Interdisciplinary Science, Shandong University, 72 Binhai Road, Qingdao 266237, Shandong, China

2 Key Laboratory of Particle Physics and Particle Irradiation (MOE), Shandong University, 72 Binhai Road, Qingdao 266237, Shandong, China

Table 1 The amount of raw data generated by particle physics experiments each year

Experiment	Operation time	Raw data (PB)
BESIII	2008–2030	0.2
LHAASO	2018–2030	9
JUNO	2021–2030	2
STCF	–	300

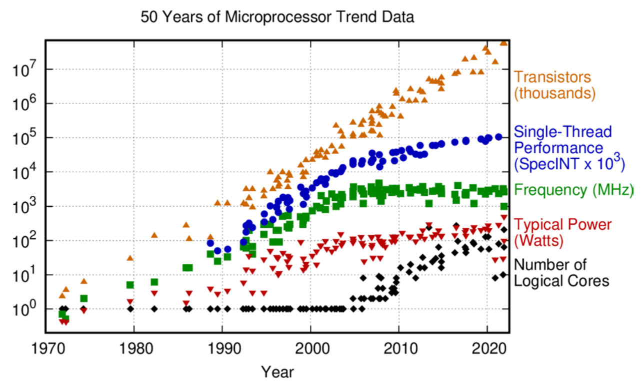


Fig. 1 Microprocessor trend data since 1970 [16]. The figure shows that the amount of transistors increases year over year, but the growth rate of single-thread performance is gradually slowing down and approaching a plateau. Moreover, the rising number of CPU cores indicates the potential benefits of utilizing multi-core processing to enhance performance

a big challenge for offline data processing software. In order to take advantage of new computing resources, such as coprocessors, graphics processing units (GPUs), and field programmable gate array (FPGA), high-performance computing techniques must be considered and adopted to speedup data processing when designing and developing an offline software system for new generation experiments.

Recent developments in microprocessor technology have underlined the trends (Fig. 1) that have been identified since the mid-2000s: while the density of transistors on CPUs has continued to rise more or less exponentially (Moore's law [12]), clock speeds have plateaued. The growth of single-threaded performance is gradually slowing down more than 15 years ago. To compensate for this, manufacturers put more cores on the chip. It is natural to apply parallel computing to take advantage of their high performance fully. In recent years, a multi-threading model has been adopted by the HEP community to develop several data processing frameworks and libraries. For instance, CMS has already completed its transition to a concurrent framework [13]. GaudiHive [14], an extension of Gaudi, has been a successful example in the HEP field, enabling concurrency through multi-threading. In 2017, a multi-threaded version of SNiPER was developed, MT-SNiPER [15], which further enhanced processing performance.

Event data model (EDM) is the core of offline data processing software framework. It defines the structure of event data in memory and data files and implements the relationship between data objects at different processing stages. The PODIO toolkit [17] provides an easy way to generate a performant implementation of an EDM from a high-level description in YAML format. One of the key ideas of PODIO is to use plain-old-data (POD) type as much as feasible, encompassing structures composed of basic types, such as integers, floats and arrays of these basic types. Thanks to their simplicity, efficiency, and ease of management, POD types enable parallelization. PODIO was chosen as the toolkit for the definition of EDM in the PEDM system. Since POD types have simple internal structures and memory layouts, they are highly suitable for sharing and transferring data between threads.

This paper aims to describe the PEDM system developed based on SNiPER and PODIO. "Key Components and Design" section outlines the key components and the design of serial and parallelized event data management system. "Implementation of the Parallelized Event Data Management System" section provides a detailed description of the implementation of the PEDM system. "Application and Performance" section shows the performance of applying PEDM system to the offline software of Super Tau Charm Facility (OSCAR) [18]. Finally, "Conclusions" section summarizes the status and gives an outlook for future development.

Key Components and Design

In offline data processing software, the EDM is responsible for defining the structure of event data in memory and data files, and implementing the relationship between data objects in different processing stages. The event data management system manages event data in memory, provides interfaces for user applications, and handles data input/output (I/O). Therefore, the EDM and the event data management system greatly influence the function and performance of the experiment software.

SNiPER Framework and PODIO

In the SNiPER framework, a parallelized solution based on Intel Threading Building Blocks (TBB) [19], MT-SNiPER, is proposed. The *Task* is one of the most essential components in SNiPER. It is responsible for controlling the event loop, functioning as the "*Application Manager*" in Gaudi [20]. As shown in Fig. 2, each *Task* contains a set of *Algorithms*, *Services*, and *Sub-Tasks*. Furthermore,

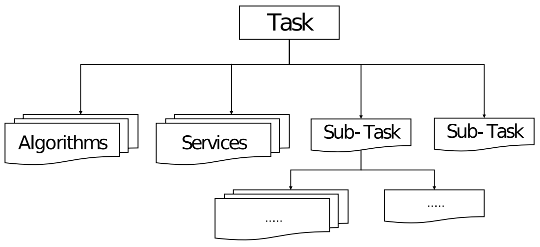


Fig. 2 Components tree in a SNiPER job. Each task comprises a set of Algorithms, Services, and Sub-Tasks

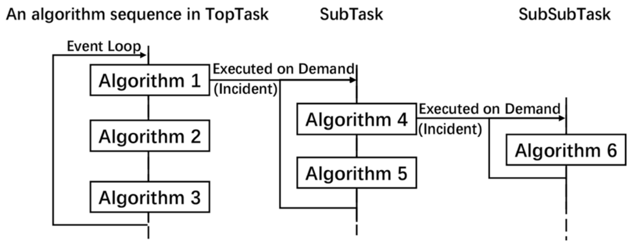


Fig. 3 Conditional execution of algorithm subsets. Incident mechanism [21] is implemented to enhance communication between tasks. A task can be triggered by an incident on demand

SNiPER considers the need to use new parallel computing techniques in data processing. It supports multiple *Tasks*, which can be naturally mapped to threads in parallelized computing. Figure 3 shows the execution sequence of multiple *Tasks*, with each *Task* managing its *Algorithms* which are executed sequentially within a single *Task*.

Multiple SNiPER Task Scheduler (Muster) [22] was implemented in MT-SNiPER to support multi-threading. Muster builds the mapping of SNiPER *Task* to Intel TBB task and breaks the traditional event loop. Multiple instances of *Tasks* are not executed directly by Muster; instead, Muster creates corresponding TBB-based workers to execute the *Tasks*. As shown in Fig. 4, Muster spawns a set of workers and starts them in different threads. Then, different events are dispatched to different workers. In each worker, it is a copy of the SNiPER *Task* that performs like a serial job. With this approach, the existing *Algorithms* and *Services* in serial mode can be almost seamlessly migrated to parallel mode.

The event data management system of SNiPER is shown in Fig. 5, including *Input System*, *Output System* and *Data Store*. *Data Store* is responsible for handling event data in memory and managing its lifecycle. Each *Task* instance has its own *Data Store*.

As the *Data Store* is used to manage general event data, we have chosen the *EventStore* of PODIO to fulfill the functions of *Data Store*.

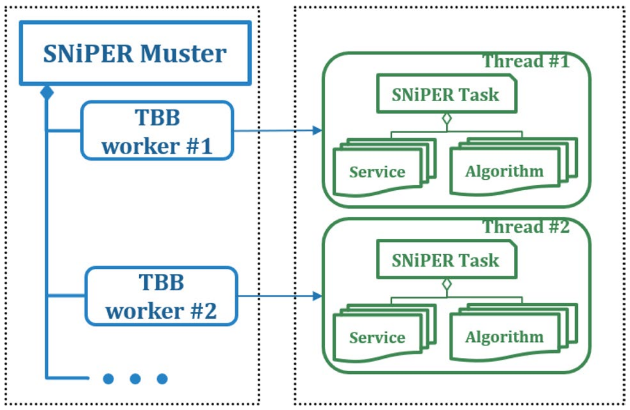


Fig. 4 SNiPER Muster builds the mapping of a SNiPER Task to an Intel TBB task

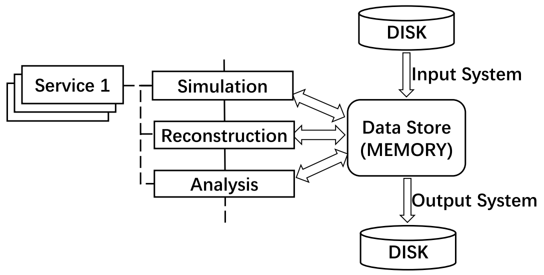


Fig. 5 Diagram of the event data management system and its interfaces to applications

The event data management system of PODIO mainly consists of three parts: the *EventStore*, the *ROOTReader*, and the *ROOTWriter*. The *ROOTReader* and *ROOTWriter* manage data input and output. The *EventStore* stores and manages the event data in memory and also provides the interfaces for *ROOTReader* and *ROOTWriter*. By integrating the event data management systems of SNiPER with PODIO, an event data management system for serial data processing has been implemented in offline software.

Multi-threaded programming can effectively utilize multi-core CPU resources and allow different threads of the same task to run simultaneously. As mentioned earlier, Muster supports parallelized process control, but PODIO only holds data objects for one single event. It was necessary to redesign a *EventStore* to cache multiple event data simultaneously. Therefore, we designed the PEDM system by integrating Muster with PODIO to support parallelized data processing.

Design of Serial Event Data Management System

In the implementation of serial data processing, *PodioDataSvc*, *PodioInputSvc*, *PodioOutputSvc* and *DataHandle* are developed to integrate the PODIO with SNiPER. The *PodioDataSvc* serves as the wrapper of *EventStore*, and there is only one *EventStore* in a job. The *EventStore* in PODIO handles transient data, whose lifetime is managed by *PodioDataSvc*. Therefore, only one event can be processed at a time, and it needs to be promptly created and deleted to ensure the proper functioning of the program. The sequence diagram for serial event data management is shown in Fig. 6. In one task, a pair of incidents are named "*BeginEvtHdl*" and "*EndEvtHdl*" to implement the input and output of event data. They are automatically triggered at the beginning and end of an event. Through the *PodioDataSvc*, *PodioInputSvc* is invoked by "*BeginEvtHdl*" to convert persistent data from the input files to transient event data in the *EventStore*, while *PodioOutputSvc* is invoked by "*EndEvtHdl*" to store the event data from the *EventStore* into the output files.

Design of the Parallelized Event Data Management System

PODIO favors composition over inheritance and uses POD types to generate thread-safe and efficient C++ code for the EDM. On the other hand, the *EventStore* was never intended to support such use cases and has exceeded its original purpose as an example implementation for a transient event store [23]. Therefore, we redesigned the *EventStore* to support caching multiple events.

Due to the flexibility of data management and the configuration of dynamic processes, it is critical to ensure memory safety. Implementing security measures such as identity recognition, locking, and unlocking was essential to guarantee effective data processing and the consistency of data reading and writing. While boosting the rate of offline data processing, this method ensures the reliability of results. A new memory management system, named *GlobalStore* (as illustrated in Fig. 7), has been designed and developed to support parallelized data processing fully. In *GlobalStore*,

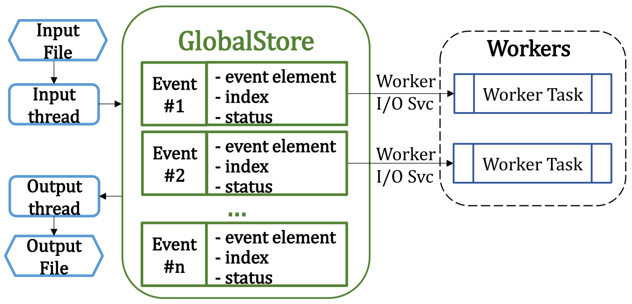


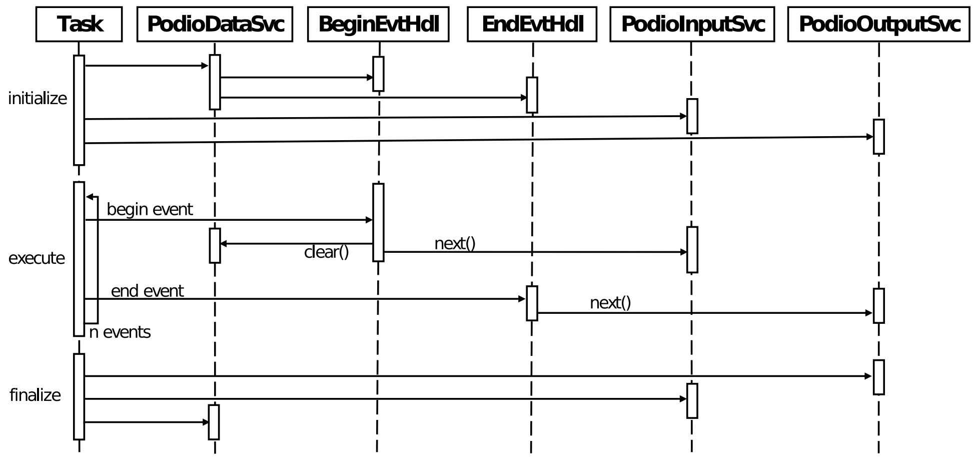
Fig. 7 The design of event data management for parallel computing. Each Task can be configured with its own local buffer and corresponding input and output services

the structure named "*event element*" is defined to manage most of the information that is handled by the *EventStore*. *GlobalStore* also stores the index and status to identify the event. The index is used to label the event, and the status is needed to ensure that the event is processed only once.

Data races arise when multiple threads simultaneously attempt to read/write data from/to the same file using ROOT I/O. To avoid this situation, decoupling the I/O functions from the *EventStore* and allocating input and output to two dedicated threads is necessary. This method significantly improves CPU utilization and reduces the task processing time, as described in "Application and Performance" section.

To efficiently deliver data to workers (see Fig. 8), the *PodioLocalDataSvc* has been developed to access event data objects using *DataHandles*. An abstract interface class, *IPodioDataSvc*, has been defined to facilitate the sharing and reusing of a set of data management services between serial and parallelized event data management systems. Both *PodioDataSvc* and *PodioLocalDataSvc* are inherited from *IPodioDataSvc*, and these services are implemented to ensure thread safety. The workers independently process events in parallel. Two new I/O services, namely the worker input service (*WInputSvc*)

Fig. 6 The sequence diagram for serial event data management. Tasks control the operation of SNiPER. The lifecycle of a Task covers the entire SNiPER job



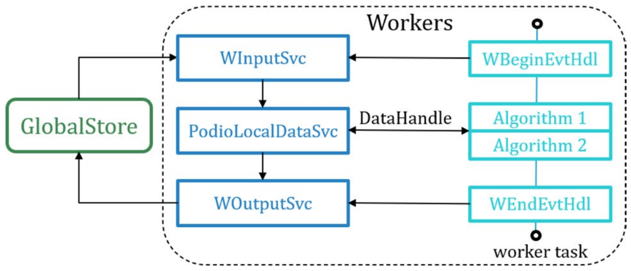


Fig. 8 The data processing procedure of the worker. A worker is functioning as to a serial SNiPER application

and the worker output service (*WOutputSvc*), have been developed to deliver event data between *GlobalStore* and workers. They are triggered by two new incidents, "*WBeginEvtHdl*" and "*WEndEvtHdl*". With this design, the intra-event and inter-event parallelism can be supported as described below:

- The first technique employs thread synchronization mechanisms, including condition variables and mutex, to facilitate complex data exchange during data processing. After a worker thread acquires an event, it is locked by the mutex and released upon data processing completion, thus preventing a data race. Condition variables are used to coordinate synchronization between threads and ensure that data are not processed redundantly. Each worker is limited to the scope of a SNiPER Task. It is the same as a serial SNiPER job in most cases.
- The second technique, lazy loading, dynamically loads data to conserve memory and computational resources. Only the data that are accessed are loaded. As a well-known technique, lazy loading has already been applied in some HEP frameworks [24].

Some options are offered to users for switching between serial and parallel modes based on their preferred job configuration. Users have the freedom to define the number of worker threads.

Implementation of the Parallelized Event Data Management System

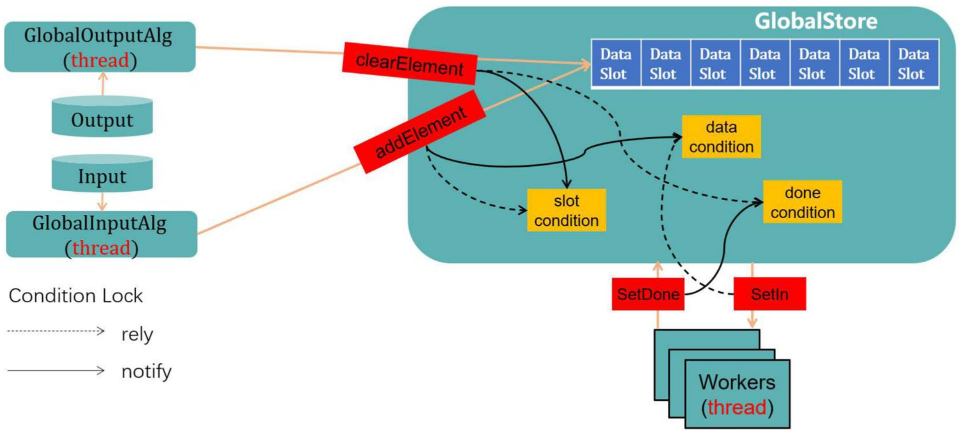
For caching more than one event, the *GlobalStore* implements multiple data slots to store the "*event elements*" as well as the status and ID corresponding to each event. The information corresponding to a specific event can be easily obtained through its event ID, and each event comprises three states: "*Ready*", "*Occupied*", or "*Done*". The states are utilized to guarantee that each event is processed exactly once. Several condition variables are used in the *GlobalStore* to ensure the safety of data exchange between threads.

The parallelized event data processing procedure is shown in Fig. 9. Event data input and output are handled by two dedicated threads to achieve multi-threaded data processing, thereby improving CPU utilization. The *GlobalInputAlg* (input thread) reads event objects sequentially and sends them to *GlobalStore*. The *GlobalOutputAlg* (output thread) writes event objects into root files as persistent data when all events are completed.

After an event is filled into the *GlobalStore* by the input thread, it is marked as "*Ready*". The primary role of the Muster scheduler is to create and manage worker threads. As soon as the conditions are met, the worker thread immediately acquires and locks the event, marking it as "*Occupied*".

Customized I/O services have been implemented in the PEDM system to facilitate data exchange between the *GlobalStore* and the workers. The input service gets event data directly from data slots, eliminating the necessity to read from a file. Each worker is assigned to a thread-local SNiPER *Task*, which is equivalent to a serial SNiPER job as described in "Design of Serial Event Data Management System" section. In the local thread, we have developed the *PodioLocalDataSvc* to handle the transient event

Fig. 9 The implementation of the PEDM system



data efficiently. It utilizes a pair of incidents named *WBeginEvtHdl* and *WEndEvtHdl* to trigger the input and output services. Additionally, the *PodioLocalDataSvc* offers the capability to retrieve existing event data objects from memory or register a new event data object in memory. This design enables efficient data management and manipulation within the worker.

The algorithms responsible for processing event data are defined within the workers. Once an event has been processed, the output service assigns it a "*Done*" status without immediately writing it out. After all events have been processed, the output thread sequentially writes the marked events into the root file as persistent data. It is important to note that, to ensure optimal utilization of data slots, the *GlobalStore* promptly clears these processed events to accommodate incoming events and prepare for the next event loop.

This implementation ensures that the processing of each event is entirely independent of other events. Events processed in worker threads can be written out without waiting for previously processed events in other threads, thereby reducing thread waiting time and improving CPU utilization.

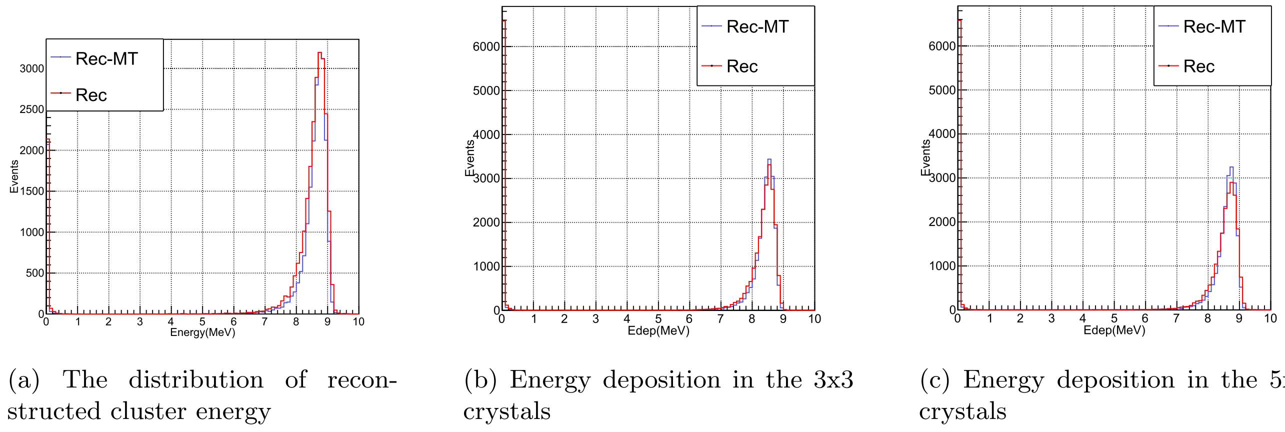
Application and Performance

The Super Tau Charm Facility (STCF) [25] is a new-generation facility of electron–positron collider operating at center-of-mass energies of 2 to 7 GeV. STCF will play a leading role in the tau, charm, and hadron physics of HEP intensity frontier in the world.

The STCF will produce several hundreds of petabytes (PB) of scientific data annually. To address this challenge, OSCAR is designed based on SNiPER [26] and partially based on Key4hep. OSCAR is developed to facilitate the offline data processing tasks for the STCF experiment, including the production of Monte Carlo simulation data, calibration, and reconstruction of collected data, as well as helping physicists to conduct physics analysis [27]. The event data management system is a fundamental component for event data transfers and communications between OSCAR based applications in offline data processing. Applying the PEDM system to OSCAR can further enhance the performance of OSCAR.

In our performance study, 20,000 events of single e^- with the energy of 5 GeV were generated at the collision point in STCF. We executed full detector simulation and Electromagnetic calorimeter (EMC) reconstruction algorithms of different events in different threads and compared the reconstructed EMC information obtained in single-threaded mode with that obtained in multi-threaded mode to validate the functionality of the system. The energy of reconstructed EMC clusters is the most crucial feature of EMC. From Fig. 10, we can see that the energy distributions and the energy deposition distribution within the 3×3 and 5×5 crystals are largely consistent in both modes, demonstrating the effective functionality of the PEDM system.

To measure the speedup, each job was repeated three times to calculate the average time and then the average time was divided by the serial time. The speedup ratio is plotted against the number of threads in Fig. 11. It can be observed that when the number of threads is less than 5, the speedup ratio exhibits a good linear behavior. However, beyond that point, factors such as thread scheduling cause the speedup ratio to gradually deviate from the ideal line. This is because the workload per task is not substantial enough. Our performance study indicates that the PEDM system can reduce data processing time and meet the requirements to handle large amounts of data.



(a) The distribution of reconstructed cluster energy
(b) Energy deposition in the 3x3 crystals
(c) Energy deposition in the 5x5 crystals

Fig. 10 The physics distribution of reconstruction results after simulating in two modes. The distributions are largely consistent between multi-threading (blue) and single-threading (red)

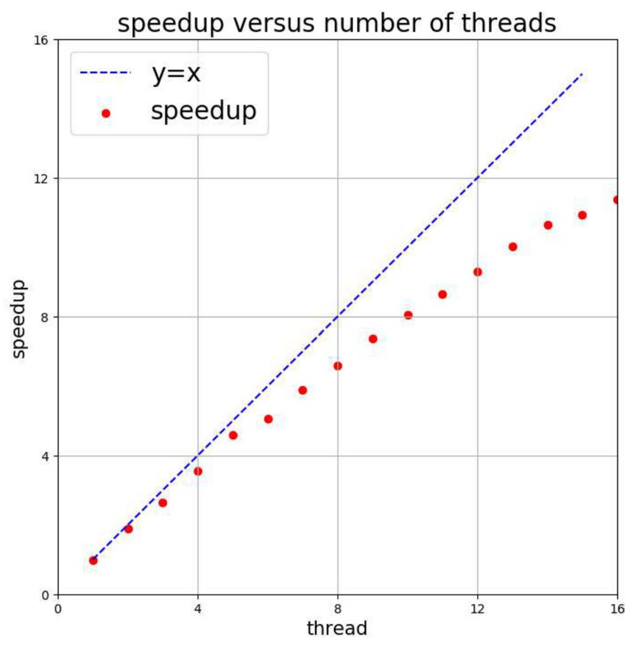


Fig. 11 Speedup ratio versus number of worker threads. The speedup ratio is close to the ideal value with less than 5 threads

Data availability The data used for performance validation in this study are generated from the simulation and reconstruction software of OSCAR (offline software of Super Tau Charm Facility) v2.3.0. Restrictions apply to the availability of softwares, therefore they are not publicly available. Data are available from the authors, [Xingtao Huang, Teng Li], upon reasonable request.

Declarations

Competing interests The authors declare no competing interests.

Conclusions

SNiPER is a simple and lightweight software framework that has been used in several HEP experiments. To meet the requirements of parallel computing, a common event data management system is redesigned by consistently integrating MT-SNiPER and PODIO. Furthermore, the PEDM system has been designed and developed to support parallelized event data management and processing well. Its application and performance study in OSCAR has shown that the PEDM system speeds up the event data processing with the advantage of concurrent event processing. At the same time, due to its good generality and flexibility in the design and implementation, other HEP experiments can also easily adopt it to implement their parallel data processing.

Acknowledgements This work was performed with the support of the National Natural Science Foundation of China (Nos.: 12025502, 12341504, 12105158), National Key Research and Development Program of China (Grant No.: 2021YFA0718403).

Author contributions Conceptualization: [Huang Xingtao], [Li Teng]; methodology: [Huang Xingtao], [Li Teng]; funding acquisition: [Huang Xingtao], [Li Teng]; writing—original draft preparation: [Shi Qianqian]; writing—review and editing: [Huang Xingtao], [Li Teng], [Shi Qianqian]; investigation: [Shi Qianqian]; software: [Shi Qianqian], [Li Teng].

Funding This study was funded by the National Natural Science Foundation of China (grant nos. 12105158, 12025502); National Key Research and Development Program of China (2021YFA0718403).

References

1. Calafiura P, Marino M, Leggett C, Lavrijsen W, Quarrie DR (2005) The Athena control framework in production, new developments and lessons learned. In: 14th international conference on computing in high-energy and nuclear physics. pp 456–458 . https://api.semanticscholar.org/CorpusID:73681202
2. Aad G, Abat E, Abdallah J et al (2008) The atlas experiment at the CERN large hadron collider. J Instrum 3(08):08003. https://doi.org/10.1088/1748-0221/3/08/S08003
3. Li W-D, Mao Y-J, Wang Y-F (2009) The bes-iii detector and offline software. Int J Mod Phys A 24S1:9–21. https://doi.org/10.1142/S0217751X09046424
4. Ablikim M, An ZH et al (2010) Design and construction of the BESIII detector. Nucl Instrum Methods Phys Res Sect A Accelerators Spectrometers Detectors Assoc Equip 614(3):345–399. https://doi.org/10.1016/j.nima.2009.12.050
5. Barrand G et al (2001) Gaudi—a software architecture and framework for building hep data processing applications. Comput Phys Commun 140:45–55. https://doi.org/10.1016/S0010-4655(01)00254-5
6. Yang Y (2023) JUNO collaboration: parallel processing in data analysis of the Juno experiment. J Phys Conf Ser 2438(1):012057. https://doi.org/10.1088/1742-6596/2438/1/012057
7. Zou JH, Huang X, Li W, Lin T, Li T, Zhang K, Deng ZY, Cao GF (2015) Sniper: an offline software framework for non-collider physics experiments. J Phys Conf Ser 664(7):072053. https://doi.org/10.1088/1742-6596/664/7/072053
8. Huang X, Li T, Zou J, Lin T, Li W, Deng Z, Cao G (2017) Offline data processing software for the Juno experiment. PoS ICHEP2016. https://doi.org/10.22323/1.282.1051
9. Cao Z (2010) A future project at Tibet: the large high altitude air shower observatory (LHAASO). Chin Phys C 34:249–252. https://doi.org/10.1088/1674-1137/34/2/018
10. Kharusi SA et al (2018) nEXO pre-conceptual design report

11. Li T, Huang W, Huang X, Ai X, Li H, Liu D (2024) Offline data processing software for the super tau charm facility. EPJ Web Conf 295:03025. https://doi.org/10.1051/epjconf/202429503025
12. Moore GE (1998) Cramming more components onto integrated circuits. Proc IEEE 86(1):82–85. https://doi.org/10.1109/JPROC.1998.658762
13. Jones CD, Sexton-Kennedy E (2014) Stitched together: transitioning CMS to a hierarchical threaded framework. J Phys Conf Ser 513(2):022034. https://doi.org/10.1088/1742-6596/513/2/022034
14. Clemencic M, Hegner B, Mato P, Piparo D (2014) Introducing concurrency in the Gaudi data processing framework. J Phys Conf Ser 513:022013. https://doi.org/10.1088/1742-6596/513/2/022013
15. Zou JH, Lin T, Li WD, Huang XT, Li T, Deng ZY, Cao GF, You ZY (2018) Parallel computing of sniper based on intel TBB. J Phys Conf Ser 1085(3):032009. https://doi.org/10.1088/1742-6596/1085/3/032009
16. Karl Rupp: microprocessor trend data. https://github.com/karlrupp/microprocessor-trend-data/tree/master/50yrs
17. Gaede F, Ganis G, Hegner B, Helsens C, Madlener T, Sailer A, Stewart GA, Volkl V, Wang J (2021) Edm4hep and podio—the event data model of the key4hep project and its implementation. EPJ Web Conf 251:03026. https://doi.org/10.1051/epjconf/202125103026
18. Huang W.H, Li H, Zhou H, Li T, Li QY, Huang XT (2023) Design and development of the core software for stcf offline data processing. JINST **18**(03), 03004 https://doi.org/10.1088/1748-0221/18/03/P03004arXiv:2211.03137 [physics.ins-det]
19. Robison AD (2011) In: Padua D (ed) Intel®Threading building blocks (TBB), pp. 955–964. Springer, Boston, MA . https://doi.org/10.1007/978-0-387-09766-4_51
20. Clemencic M, Corti G, Easo S, Jones CR, Miglioranzi S, Pappagallo M, Robbe P (2011) The LHCB simulation application, gauss: design, evolution and experience. J Phys Conf Ser 331:032023. https://doi.org/10.1088/1742-6596/331/3/032023
21. Gaudi framework services. https://gaudi-framework.readthedocs.io/en/latest/old/GDG_Services.html
22. Zou J, Lin T, Li W, Huang X, Deng Z, Cao G, You Z (2019) The event buffer management for mt-sniper. EPJ Web Conf 214:05026. https://doi.org/10.1051/epjconf/201921405026
23. Fernandez Declara P, Gaede F, Ganis G, Hegner B, Helsens C, Madlener T, Sailer A, Stewart G.A, Volkl V (2023) Of frames and schema evolution—the newest features of podio. In: 21th international workshop on advanced computing and analysis techniques in physics research: AI meets reality. https://arxiv.org/abs/2312.08199
24. Eté R, Gaede F, Benda J, Grasland H (2020) Marlinmt—parallelising the marlin framework. EPJ Web Conf
25. Achasov M et al (2023) STCF conceptual design report (volume 1): physics & detector. Front Phys 19(1):14701. https://doi.org/10.1007/s11467-023-1333-z
26. Ai X, Huang X, Li T, Qi B, Qin X (2024) Design and development of STCF offline software. Mod Phys Lett A. https://doi.org/10.1142/S0217732324400066
27. Huang WH, Li T, Li QY, Li H, Liu D, Huang XT (2023) Offline software framework for the super tau charm facility. J Phys Conf Ser 2438(1):012054. https://doi.org/10.1088/1742-6596/2438/1/012054