The Application of SNiPER to the JUNO Simulation Postprint

Authors: Tao Lin, Jiaheng Zou, Weidong Li, Ziyan Deng, Xiao Fang, Guofu Cao, Xingtao Huang, Zhengyun You

Date: 2017-11-10T00:00:00+00:00

Abstract

The JUNO (Jiangmen Underground Neutrino Observatory) is a multipurpose neutrino experiment which is designed to determine neutrino mass hierarchy and precisely measure oscillation parameters. As one of the important systems, the JUNO offline software is being developed using the SNiPER software. In this proceeding, we focus on the requirements of JUNO simulation and present the working solution based on the SNiPER.

The JUNO simulation framework is in charge of managing event data, detector geometries and materials, physics processes, simulation truth information etc. It glues physics generator, detector simulation and electronics simulation modules together to achieve a full simulation chain. In the implementation of the framework, many attractive characteristics of the SNiPER have been used, such as dynamic loading, flexible flow control, multiple event management and Python binding. Furthermore, additional efforts have been made to make both detector and electronics simulation flexible enough to accommodate and optimize different detector designs.

For the Geant4-based detector simulation, each sub-detector component is implemented as a SNiPER tool which is a dynamically loadable and configurable plugin. So it is possible to select the detector configuration at runtime. The framework provides the event loop to drive the detector simulation and interacts with the Geant4 which is implemented as a passive service. All levels of user actions are wrapped into different customizable tools, so that user functions can be easily extended by just adding new tools. The electronics simulation has been implemented by following an event driven scheme. The SNiPER task component is used to simulate data processing steps in the electronics modules. The electronics and trigger are synchronized by triggered events containing possible physics signals.

The JUNO simulation software has been released and is being used by the JUNO collaboration to do detector design optimization, event reconstruction algorithm development and physics sensitivity studies.

Full Text

The Application of SNiPER to the JUNO Simulation

Tao Lin[1], Jiaheng Zou[1], Weidong Li[1], Ziyan Deng[1], Xiao Fang[2], Guofu Cao[1], Xingtao Huang[3], and Zhengyun You
(On Behalf of the JUNO Collaboration)

[1]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, China
[2]Sichuan University, Chengdu, China
[3]Shandong University, Jinan, China
Sun Yat-sen University, Guangzhou, China
E-mail: lintao@ihep.ac.cn

Abstract

The Jiangmen Underground Neutrino Observatory (JUNO) is a multipurpose neutrino experiment designed to determine the neutrino mass hierarchy and precisely measure oscillation parameters. As a critical component of the experiment, the JUNO offline software is being developed using the SNiPER framework. In this proceeding, we focus on the requirements of JUNO simulation and present a working solution based on SNiPER.

The JUNO simulation framework manages event data, detector geometries and materials, physics processes, simulation truth information, and more. It integrates physics generator, detector simulation, and electronics simulation modules to achieve a complete simulation chain. In implementing this framework, we have leveraged many attractive features of SNiPER, including dynamic loading, flexible flow control, multiple event management, and Python bindings. Furthermore, additional efforts have been made to ensure both detector and electronics simulation are flexible enough to accommodate and optimize different detector designs.

For the Geant4-based detector simulation, each sub-detector component is implemented as a SNiPER tool—a dynamically loadable and configurable plugin—allowing runtime selection of detector configurations. The framework provides an event loop to drive the detector simulation and interacts with Geant4, which is implemented as a passive service. All levels of user actions are wrapped into different customizable tools, enabling easy extension of user functions by simply adding new tools. The electronics simulation follows an event-driven scheme, using SNiPER' s task component to simulate data processing steps in electronics modules. The electronics and trigger systems are synchronized through triggered events containing potential physics signals.

The JUNO simulation software has been released and is currently being used by the collaboration for detector design optimization, event reconstruction algorithm development, and physics sensitivity studies.

1. Introduction

JUNO [**?**, **?**] is a multi-purpose neutrino experiment designed to measure the neutrino mass hierarchy, precisely measure oscillation parameters, and detect astrophysical and geological neutrinos. It will be located in southern China, approximately 53 km from the Yangjiang and Taishan nuclear power plants.

Figure 1 [Figure 1: see original paper] shows a schematic view of the JUNO detector. The central detector (CD), the innermost component, consists of a spherical acrylic vessel containing 20 kt of liquid scintillator (LS) and is used for neutrino detection. Photomultiplier tubes (PMTs) surround the acrylic vessel to collect light from the LS. A water pool surrounding the CD provides shielding against radioactive backgrounds, with PMTs in the water pool detecting Cherenkov light to veto cosmic ray muon events. A top tracker located above the water pool also measures and vetoes muons.

The raw data must be processed offline and reconstructed for subsequent physics analysis. Since simulation data and raw data share the same event data model, simulation data can be processed identically to real data. Figure 2 [Figure 2: see original paper] illustrates the current data processing steps, implemented as different algorithms following the SNiPER framework [**?**]. Each algorithm reads event data [**?**] from a data buffer, performs corresponding calculations, and writes results back to the buffer. A ROOT I/O service [**?**] handles event data persistence between the data buffer and disk. A detector geometry service uses GDML [**?**] and ROOT for detector geometry description, providing unified access for algorithms.

In the workflow, physics generators produce kinematic information for primary particles, which is stored in GenEvent objects using the HepMC format [**?**]. Other formats such as HepEvt and GENIE [**?**] are converted to HepMC via corresponding tools. The detector simulation algorithm then accesses these GenEvent objects to begin tracking. Hits containing charge and time information are generated in sensitive detectors and saved in SimEvent objects. Subsequently, the electronics simulation algorithm reads these SimEvent objects and performs digitization, generating ElecEvent objects containing waveform information. These waveforms are processed by a PMT calibration algorithm, which saves CalibEvent objects. The event reconstruction algorithm reads CalibEvent objects, performs reconstruction, and stores RecEvent objects. Physicists can then perform any physics analysis using RecEvent objects.

As part of the offline data processing software [**?**], reliable Monte Carlo (MC) simulation software plays a crucial role in detector parameter optimization and physics studies. A simulation framework based on the SNiPER framework has been developed, consisting of physics generator, detector simulation, and elec-

tronics simulation modules. It integrates seamlessly with SNiPER and leverages its capabilities.

2. Detector Simulation

The JUNO detector simulation software, built on Geant4 [**?**, **?**], was originally developed as a standalone application. To integrate it into the framework, we designed several interface classes to connect SNiPER and Geant4 without requiring extensive modifications to user code. SNiPER' s extensibility simplifies management of physics generators, detector geometry construction, user actions, and other components. Additionally, tools are used to configure parameters via Python scripts rather than Geant4 macro files.

Figure 3 [Figure 3: see original paper] presents the class diagram of the detector simulation framework, which serves as a bridge between Geant4 and SNiPER. It comprises integration components, a configurable user interface, geometry management, and modularized user actions. This middleware layer makes detector simulation a seamless component of the offline software.

Both SNiPER and Geant4 have their own event loop control, necessitating adaptation of Geant4' s event processing workflow into SNiPER. To adopt SNiPER' s event loop, an algorithm named DetSimAlg is invoked on an event-by-event basis within SNiPER. DetSimAlg then invokes a service called G4SvcRunManager, derived from G4RunManager, to initialize simulation and start tracking. To decouple the run manager from user code such as detector construction and physics lists, we implemented a factory class called DetSimFactory that constructs user code components and passes them to G4SvcRunManager.

Physics generator interfaces, physics processes, and detector components are all configurable through lightweight tools rather than Geant4 macro files, maintaining a consistent user interface across algorithms via Python scripts. For compatibility with Geant4 macro files, a command-line option can specify input macro files, enabling users to visualize detectors by providing a visualization macro file.

Geometry management addresses the challenge of multiple detector design options. To ensure consistency and flexibility, we propose a detector element as a high-level concept representing one detector component. For example, a central detector is a detector element containing only LS, container, and buffer material –PMTs are not placed within this element. This allows different central detector options to be interchanged while keeping PMT arrangements unchanged, which is also useful when different PMT types are employed. These PMTs are managed uniformly. In addition to real detector components, a GDML-based detector component can construct any geometry and materials by parsing an input GDML file.

A separate interface controls the placement of detector elements, allowing calibration units to be placed into the central detector at runtime through corre-

sponding configurations.

User actions are essential for developers to access MC truth information during detector simulation. To modularize these actions, we organize them into a list of lightweight tools that can access all necessary information supplied by Geant4. A manager invoked by different levels of user actions dispatches calculations to these tools according to their registration order. Users can load corresponding tools as needed, and developers can create new tools without modifying existing ones. Several "official" tools are loaded by default, such as writers for the event data model and geometry.

3. Electronics Simulation

An inverse beta decay (IBD, $\bar{\nu}_e + p \rightarrow e^+ + n$) event consists of a prompt signal from the positron and a delayed signal from neutron capture on hydrogen. Electronics simulation must split such physics events into two readouts while also mixing detector backgrounds.

To handle both event splitting and mixing, we propose a "pull" mode workflow. As shown in Figure 4 [Figure 4: see original paper], electronics simulation begins with a readout algorithm rather than an event mixing algorithm. Initially, the readout algorithm requires a trigger from the trigger buffer to create a readout event. If the trigger buffer is empty, the trigger simulation algorithm executes. This trigger simulation requires sufficient pulses to generate an event trigger. If the pulse buffer lacks enough pulses, a PMT simulation algorithm produces pulses from the hit buffer until enough pulses are available within a time window. An unpacking algorithm unpacks hits from different events into the hit buffer, while an event mixing algorithm places events into the event buffer. Once all buffers contain the necessary data, the readout algorithm digitizes waveforms and saves them to the readout event. This process completes the simulation of one event. Finally, the global timestamp is updated to synchronize the data buffers. For memory optimization, circular buffers are used in some algorithms, such as the waveform algorithm.

In this implementation, each algorithm is associated with a task. These tasks execute passively, invoked only when data in the corresponding buffer is insufficient. SNiPER' s incident mechanism facilitates communication between algorithms and tasks. When a task is invoked, its registered algorithms execute once, and the task' s input, output, and buffer services are updated. To mix events from different files, tasks containing only input and buffer services are set up dynamically. Users can also assign label names to identify data types and associate event rates. Based on these rates, an event mixing algorithm invokes corresponding tasks, accesses their buffers, and places events into its own buffer. An unpacking algorithm then sorts all hits chronologically and mixes them, achieving hit-level event mixing.

4. Performance Measurements

We used IBD events for performance studies, measuring detector simulation, electronics simulation without mixing, and electronics simulation with mixing. For each category, we prepared 20 jobs with 1000 readout events per job. Jobs were submitted to a dedicated blade server with an Intel Xeon E5-4620 v2 @ 2.60GHz CPU and 132 GB of memory. A daemon process monitored each job' s CPU and memory usage, saving results to files. To eliminate interference, the daemon updated results every 5 seconds.

Due to readout splitting of IBD events, one SimEvent object produces two ElecEvent objects. For an electronics simulation job, 1000 readout objects are generated from fewer than 500 IBD events. Hit-level event mixing is performed only in electronics simulation. Input SimEvent objects were prepared before event mixing. For testing, we used 1 Hz of IBD events with 6 Hz of U-238 events, 6 Hz of Th-232 events, and 2 Hz of K-40 events. Other configurations remained identical to previous measurements.

Table 1 summarizes the simulation software performance. Detector simulation is both time- and memory-intensive due to the propagation of thousands of optical photons. CPU utilization demonstrates that detector simulation software remains efficient even when integrated with SNiPER. For high-energy cosmic ray muon events that generate millions of optical photons in the liquid scintillator, various methods have been studied to accelerate simulation [**?**].

Electronics simulation is memory- and I/O-intensive. Despite optimizations to the data structure, thousands of waveforms sampled at 1 GHz still consume significant memory and disk space. Event mixing requires less time and disk usage due to the lower energies of background particles, with most readout objects containing only background. However, event mixing must load events randomly from multiple files, suppressing CPU utilization due to I/O latency.

5. Conclusions

This proceeding demonstrates how SNiPER is applied in JUNO simulation software, including physics generator, detector simulation, and electronics simulation modules. By introducing several middleware classes, we integrated SNiPER and Geant4 to simplify geometry and user action management. Using SNiPER' s incident mechanism, electronics simulation implements a "pull" workflow that naturally supports hit-level mixing. Performance measurements show the current simulation software meets requirements.

However, challenges remain, such as simulating huge numbers of optical photons and PMTs. We are studying concurrent computing using GPUs and Phi coprocessors to accelerate light propagation simulation in the large liquid scintillator detector.

Acknowledgements

This work is supported by the Joint Large-Scale Scientific Facility Funds of the NSFC and CAS (U1532258), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA10010900), and the National Natural Science Foundation of China (11575224, 11405279, 11675275).

References

[1] An F et al. (JUNO) 2016 J. Phys. G43 030401 (Preprint 1507.05613)
[2] Djurcic Z et al. (JUNO) 2015 (Preprint 1508.07166)
[3] Zou J H, Huang X T, Li W D, Lin T, Li T, Zhang K, Deng Z Y and Cao G F 2015 J. Phys. Conf. Ser. 664
[4] Li T, Xia X, Huang X T, Zou J H, Li W D, Lin T, Zhang K and Deng Z Y (Preprint 1702.04100)
[5] Brun R and Rademakers F 1997 Nucl. Instrum. Meth. A389 81–86
[6] Chytracek R, McCormick J, Pokorski W and Santin G 2006 IEEE Trans. Nucl. Sci. 53 2892
[7] Dobbs M and Hansen J B 2001 Comput. Phys. Commun. 134 41–46
[8] Andreopoulos C et al. 2010 Nucl. Instrum. Meth. A614 87–104 (Preprint 0905.2517)
[9] Huang X T, Li T, Zou J H, Lin T, Li W D, Deng Z Y and Cao G F 2016 PoS ICHEP2016 1051 URL http://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=282
[10] Agostinelli S et al. (GEANT4) 2003 Nucl. Instrum. Meth. A506 250–303
[11] Allison J et al. 2006 IEEE Trans. Nucl. Sci. 53 270
[12] Lin T, Deng Z Y, Li W D, Cao G F, You Z Y and Li X Y 2016 Chin. Phys. C40 086201 (Preprint 1602.00056)

Note: Figure translations are in progress. See original paper for figures.