# Status of the parallelized JUNO simulation software

*Tao* Lin[1,*], *Jiaheng* Zou[1,**], *Weidong* Li[1,2], *Ziyan* Deng[1,2], *Guofu* Cao[1,2], *Xingtao* Huang[3], and *Zhengyun* You[4](on behalf of JUNO collaboration)

[1]Institute of High Energy Physics, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]Shandong University, Jinan, China
[4]Sun Yat-Sen University, Guangzhou, China

**Abstract.** The Jiangmen Underground Neutrino Observatory (JUNO) is a multi-purpose neutrino experiment. It consists of a central detector, a water pool and a tracker placed on top. The central detector, which is used for neutrino detection, consists of a 20 kt liquid scintillator target and about 18,000 20-inch photomultiplier tubes (PMTs) to detect scintillation photons.
Simulation software is an important part of the JUNO offline software. To speed up the simulation, a parallelized simulation framework has been developed based on the SNiPER framework and Geant4 version 10. The SNiPER task components are in charge of the event loop, which can run in sequential mode, Intel TBB mode and other modes. Based on SNiPER, the simulation framework and its underlying parallel libraries have been decoupled. However, parallelized simulation of correlated events is a challenge. In order to keep the correct event order, a component called global buffer is developed in SNiPER.
In this paper, an overview of the parallelized JUNO simulation framework is presented. The global buffer is used in the parallelized event correlation simulation. An event generator produces events with timestamps in sequential mode. These events are put into the global buffer and processed by the detector simulation algorithms in different tasks. After simulation, the events are saved into ROOT files with a ROOT I/O service running in a dedicated thread. Finally, the software performance is presented.

## 1 Introduction

The JUNO [1, 2] experiment, located in southern China, aims to determine the neutrino mass hierarchy. It is about 53 km away from the Yangjiang and Taishan nuclear power plants. To detect neutrinos, a central detector is filled with 20 kt liquid scintillator (LS) target. When a neutrino interacts with LS, it is captured by a proton and then a positron ($e^+$) and a neutron (n) are produced, which is called inverse-beta decay (IBD). Then the positron and the neutron deposit energy in the LS and scintillation photons are produced. These photons propagate in the LS and are collected by the surrounding photomultiplier tubes (PMTs) in the water. Figure 1 shows the schematic view of the JUNO detector, where the innermost part is the central detector which is surrounded by a water Cerenkov detector. The water is used to

[*]e-mail: lintao@ihep.ac.cn
[**]e-mail: zoujh@ihep.ac.cn

suppress radioactivity backgrounds from PMTs, rocks and so on. Equipped with PMTs, the water Cerenkov detector can also measure cosmic ray muons. On the top of the water Cerenkov detector, there is a detector called the top tracker, which can also detect muons.

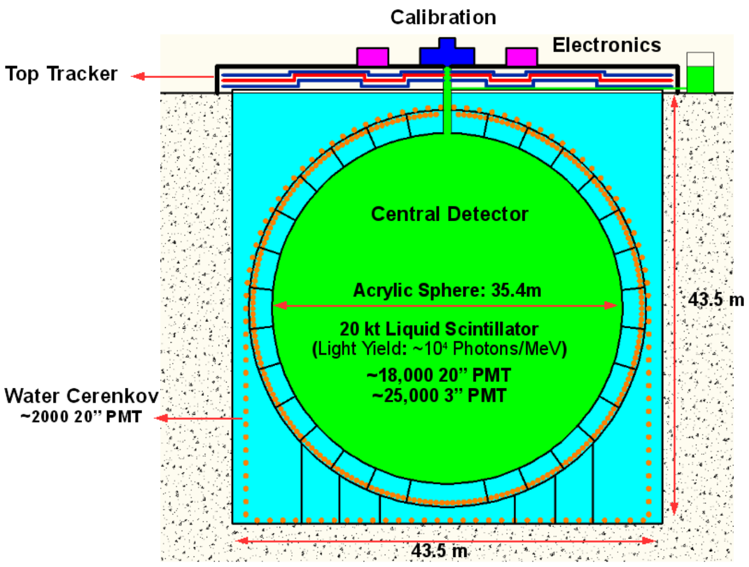

**Figure 1.** Schematic view of the JUNO detector.

As part of the offline JUNO software [3], the detector simulation software is important for detector design, reconstruction algorithm development and physics studies. To fulfill the requirements of Monte Carlo (MC) production, a detector simulation framework [4] is developed based on SNiPER [5] and Geant4 [6, 7]. The framework can support different types of physics generators by converting the output into a HepMC [8] based event data model. Then the framework loads the HepMC event object and converts it to a Geant4 object, so that Geant4 can start the simulation. A ROOT [9] based event data model, called `SimEvent`, is created and filled with hit objects and then put into an event data buffer. The framework saves the event objects into a ROOT file automatically [10].

In recent years, a multi-threading model has been adopted by several data processing frameworks and libraries, such as Gaudi [11], Geant4 [12] and ROOT. The LHC experiments such as ATLAS and CMS experiments adopt this model [13–17]. One benefit of a multi-threading model compared with a multi-processing model is the shared memory. For example, the full geometry of JUNO detector is complicated, requiring several hundred MB of physical memory. If the geometry object is shared by multiple threads, the memory usage is reduced significantly. Hence a parallelized detector simulation framework for JUNO was designed and a prototype was implemented based on SNiPER-MT [18].

One challenge for the framework is handling events with time correlation. The order should be preserved for these physics events. For example, a physics generator simulating a supernova burst will generate a series of events with time correlation. When simulating these events using the multi-threading model, the order of events should be kept the same. The supernova burst can be simulated in a dedicated thread, however, the number of events for each supernova burst may differ. The multi-threaded job will finish only when the last event is done, which causes other CPUs to idle. To improve the CPU utilization of one job, a lock-free global buffer is introduced in SNiPER-MT. Using the global buffer, the physics generator produces HepMC objects on demand for Geant4 to later retrieve and simulate events using different threads.

This section introduces the motivation and challenge of the parallelized simulation for the JUNO experiment. The remainder of this paper is as follows, section 2 presents the design and implementation of the parallelized simulation framework to handle time correlation events. The performance measurement and conclusion are shown in section 3 and section 4 respectively.

## 2 Parallelized simulation software

The SNiPER framework adopts Intel Threading Building Blocks (TBB) [19] to implement a parallelized version, which is called SNiPER Muster (Multiple SNiPER Task Scheduler) [20]. SNiPER Muster controls the mapping a SNiPER `Task` component to an Intel TBB worker. A SNiPER `Task` component is a lightweight manager of algorithms, services and tools. When a SNiPER `Task` component is dispatched to an Intel TBB worker, the algorithms registered in the `Task` component are executed. By default, each SNiPER `Task` component can be configured with its own local buffer and corresponding input and output services. Therefore most of the user code is not affected when migrating from a serial version to a parallelized version.

In order to keep most of the Geant4 user code unchanged during parallelization, the simulation framework retains the same interfaces. These interfaces are divided into two categories: one for initialization and another for event simulation. The design of the simulation software framework is shown in figure 2. A global task is invoked during initialization to control the detector construction and physics list. The global task is then shared by other workers, so that a slave run manager can access detector geometries managed by the master run manager. Then the worker tasks start to simulate events in parallel. Each worker task has its own instances of Geant4 user actions, such as event action, tracking action and stepping action, which are user hooks to collect data from Geant4 during event processing.

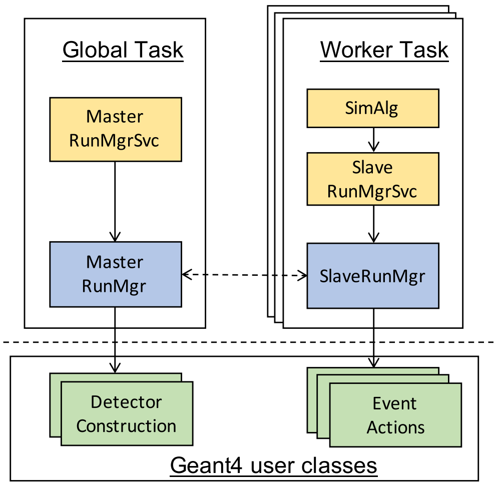

**Figure 2.** Components of the simulation software.

For each worker task, a physics generator algorithm, a detector simulation algorithm and a ROOT I/O service are registered. An event object is created by the physics generator algorithm and then put into the local buffer of the task. The local buffer is shared by the components registered in the task, so the detector simulation algorithm and the ROOT I/O service can access the events. After converting the `HepMC` object to the Geant4 object, the detector simulation algorithm simulates the event using Geant4 and generates a collection of hit objects for the PMTs. Then the ROOT I/O service saves the event into a ROOT file. These algorithms and services are all thread-safe. Each worker task has its own copies of these algorithms and services, as shown in figure 3. The events in the different local buffers are not shared.

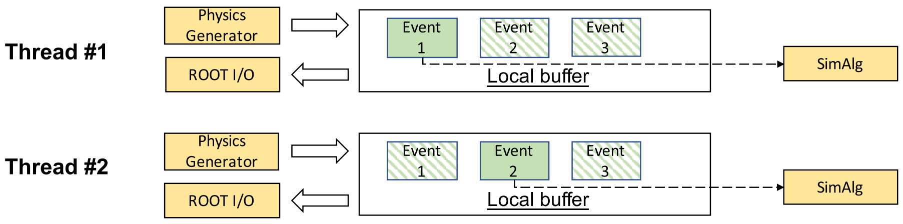

**Figure 3.** Parallelized simulation using local buffers.

Since the worker tasks are configured with their own ROOT I/O services, these events are not saved into the same file. Even though there are different copies of ROOT `TFile` and `TTree` objects, using ROOT 5 will crash the software during simulation. Several locks are needed to protect these objects. To avoid locking explicitly, ROOT 6 is then used in the simulation software. In order to enable thread safety in ROOT 6, a method called `ROOT::EnableThreadSafety()` is invoked during initialization of the global task.

Due to multiple output files per job, a `TChain` object is used to load them for analysis. However, it becomes difficult to manage files if each thread has its own output. An experimental ROOT component called `TBufferMerger` is used to merge output files automatically to reduce the number of files. This feature is used in events without correlation, such as events generating using particle guns. For events with correlation, a SNiPER component called global buffer is used.

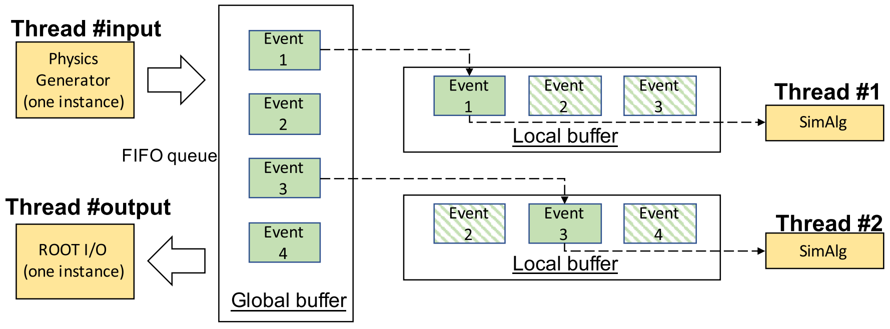

**Figure 4.** Schematic view of the parallelized simulation using global buffer.

As shown in Figure 4, two additional threads are created: one for generating events and another for saving events. The global buffer is a FIFO queue, whose capacity is configurable.

The events with timestamps are produced by the physics generator and then filled into the global buffer. If the buffer is full, the physics generator is blocked until there is free space. For the worker task, only a detector simulation algorithm and a local buffer are configured and registered. The local buffer asks the global buffer to return event objects, which are not simulated yet. Then the simulation algorithm loads an event from the local buffer and starts simulation of this event. After event generation, the ROOT I/O service saves the events in the same order. A flag is used to indicate whether the event is finished or not. The ROOT I/O service can only get an event from the global buffer when it is finished. If the event is not ready, the thread with the ROOT I/O service is suspended. If the event is ready, then the thread is woken up and the ROOT I/O service saves the event.

Using the global buffer, the parallelized simulation of a supernova burst is possible. The supernova physics generator produces events in order and pushes them into the global buffer. When the global buffer is full, the generation is suspended. All the worker threads retrieve their own events and do the detector simulation. Then the events are marked as done and saved by the ROOT I/O service. It reduces the time to simulate a supernova burst, while only one ROOT file is generated with correct event order.

## 3 Performance

The performance measurements are run on a blade server with Intel Xeon CPU E5-2680 v3 @ 2.5 GHz and 64 GB of memory. The Geant4 version is 10.04.p02, which fixed the mutex used in the material property tables of Geant4. The ROOT version is 6.12.06, which enables thread safe support. Each job is configured with 1000 events of 2.2 MeV $\gamma$s, generated at the detector center. Jobs are repeated three times for each measurement and the average timing for the simulation is calculated. The initialization time is not included. The studies of performance measurements include two parts: one is simulation with local buffers only and another is simulation with a global buffer.

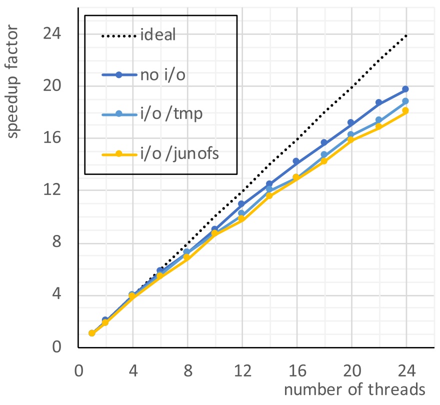

**Figure 5.** Performance of simulation all using local buffers. "/tmp" is a local file system and "/junofs" is a network shared file system.

Figure 5 shows the performances, where the simulation software is configured without any global buffer. Three cases are considered. The first case is using local buffers without any I/O, which can eliminate I/O interference. The second case is using local buffers with

I/O, where the output is saved on the local disk of the blade server. The third case is same as the second one, but the output is saved into a network file system. As expected, there is an impact on the speedup when the I/O is considered. Comparing between the local disk and the network disk, it is also expected that the better performance of the file system, the better speedup.

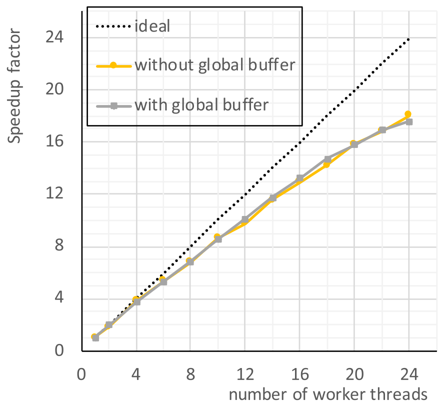

**Figure 6.** Performance of simulation using a global buffer.

As shown in figure 6, the performance of the simulation with a global buffer is close to linear speedup. The extra two threads are not included in the number of worker tasks. As a comparison, the third case without the global buffer is also shown. The global buffer and a dedicated thread with a ROOT I/O service does not affect the simulation timing. The performance of the lock free global buffer is as expected. Due to two dedicated threads for event generation and output, the speedup drops a little when all of the cores are occupied by worker tasks.

## 4 Conclusion

In this paper, the handling of events with correlation in the parallelized JUNO simulation software is shown. By using the new SNiPER component global buffer, events are generated using a dedicated thread with a physics generator algorithm and then put into the global buffer and simulated by the parallelized detector simulation algorithms. These events are saved into ROOT files using an extra thread with a ROOT I/O service. The result shows an almost linear speedup with respect to the number of worker threads.

In the current parallelized simulation, the global buffer will block the output service when the current event is not done. One possible problem is that it will block the whole buffer if only this event is not completed. The physics generator is blocked and the worker tasks are all idle. Currently, simulating events in the same energy ranges can avoid such problems. In the future, a global buffer which can be dynamically expanded will be implemented. If the buffer is full and the first event is not finished, the software will detect the problem and expand the capacity, so the physics generator and the worker tasks can continue.

This work is supported by 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, National Natural Science Foundation of China (11575224, 11675275, 11805223).

## References

[1] Z. Djurcic et al. (JUNO) (2015), 1508.07166
[2] F. An et al. (JUNO), J. Phys. **G43**, 030401 (2016), 1507.05613
[3] X.T. Huang, T. Li, J.H. Zou, T. Lin, W.D. Li, Z.Y. Deng, G.F. Cao, PoS **ICHEP2016**, 1051 (2016)
[4] T. Lin, J. Zou, W. Li, Z. Deng, X. Fang, G. Cao, X. Huang, Z. You (JUNO), J. Phys. Conf. Ser. **898**, 042029 (2017), 1702.05275
[5] J.H. Zou, X.T. Huang, W.D. Li, T. Lin, T. Li, K. Zhang, Z.Y. Deng, G.F. Cao, J. Phys. Conf. Ser. **664**, 072053 (2015)
[6] S. Agostinelli et al. (GEANT4), Nucl. Instrum. Meth. **A506**, 250 (2003)
[7] J. Allison et al., IEEE Trans. Nucl. Sci. **53**, 270 (2006)
[8] M. Dobbs, J.B. Hansen, Comput. Phys. Commun. **134**, 41 (2001)
[9] R. Brun, F. Rademakers, Nucl. Instrum. Meth. **A389**, 81 (1997)
[10] T. Li, X. Xia, X. Huang, J. Zou, W. Li, T. Lin, K. Zhang, Z. Deng, Chin. Phys. **C41**, 066201 (2017), 1702.04100
[11] M. Clemencic, B. Hegner, C. Leggett, J. Phys. Conf. Ser. **898**, 042044 (2017)
[12] A. Dotti, M. Asai, G. Barrand, I. Hrivnacova, K. Murakami, pp. 1–2 (2015), 1605.01792
[13] S. Farrell, P. Calafiura, C. Leggett, V. Tsulaia, A. Dotti (ATLAS), J. Phys. Conf. Ser. **898**, 042012 (2017)
[14] P. Calafiura, W. Lampl, C. Leggett, D. Malon, G. Stewart, B. Wynne, J. Phys. Conf. Ser. **664**, 072031 (2015)
[15] G.A. Stewart et al. (ATLAS), J. Phys. Conf. Ser. **762**, 012024 (2016)
[16] E. Sexton-Kennedy, P. Gartung, C.D. Jones, D. Lange, J. Phys. Conf. Ser. **608**, 012034 (2015)
[17] P. van Gemmeren, S. Binet, P. Calafiura, W. Lavrijsen, D. Malon, V. Tsulaia (ATLAS), J. Phys. Conf. Ser. **396**, 022054 (2012)
[18] T. Lin, J. Zou, W. Li, Z. Deng, G. Cao, X. Huang, Z. You (JUNO), J. Phys. Conf. Ser. **1085**, 032048 (2018), 1710.07150
[19] *Intel threading building blocks*, https://www.threadingbuildingblocks.org/, [Online; accessed 19-March-2018]
[20] J.H. Zou, T. Lin, W.D. Li, X.T. Huang, T. Li, Z.Y. Deng, G.F. Cao, Z.Y. You, J. Phys. Conf. Ser. **1085**, 032009 (2018)