Parallelized JUNO simulation software based on SNiPER

T Lin[1], J H Zou[1], W D Li[1], Z Y Deng[1], G F Cao[1], X T Huang[2] and Z Y You[3]

(On Behalf of the JUNO Collaboration)

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

E-mail: lintao@ihep.ac.cn, zoujh@ihep.ac.cn

Abstract. The Jiangmen Underground Neutrino Observatory (JUNO) is a neutrino experiment to determine the neutrino mass hierarchy. It has a central detector used for neutrino detection, which consists of a spherical acrylic vessel containing a 20 kt liquid scintillator (LS) and about 18,000 20-inch photomultiplier tubes (PMTs) to collect light from LS. Around the central detector, there is a water pool to provide shielding from background radioactivity. The outer water pool is also equipped with about 2000 PMTs to measure cosmic ray muons by detecting Cherenkov light.

As one of the important parts in JUNO offline software, the serial simulation framework is developed based on SNiPER. It manages physics generator, detector simulation, event mixing and digitization. However, Geant4 based detector simulation of such a large detector is time-consuming and challenging. It is necessary to take full advantages of parallel computing to speed up simulation. Starting from version 10.0, Geant4 supports event-level parallelism. Even though this is based on pthread, it can be extended with other libraries such as Intel TBB and MPI. Therefore it is possible to parallelize JUNO simulation framework by integrating Geant4 and SNiPER.

In this paper, our progress in developing parallelized simulation software is presented. The SNiPER framework can run in sequential mode, Intel TBB mode or other modes. The SNiPER task component is in charge of the event loop, which is like a simplified application manager. Two types of tasks are introduced in the simulation framework, one is a global task and another is a worker task. The global task will run only once to initialize detector geometry and physics processes before any other tasks spawned. Later it is accessed by other tasks passively. The worker tasks will be spawned after the global task is done. In each worker task, a Geant4 run manager is invoked to do the real simulation. In this way the simulation framework and the underlying TBB have been decoupled. Finally, the software performance of parallelized JUNO simulation software is also presented.

1. Introduction

JUNO [1,2] is an underground neutrino experiment to measure the neutrino mass hierarchy and oscillation parameters. It will be located in southern China, about 53 km away from two nuclear power plants. Figure 1 shows the schematic view of the JUNO detector. To detect neutrinos,

the innermost part, which is called the central detector, is filled with a 20 kt liquid scintillator (LS). When neutrinos react with the LS, light is produced and then collected by the surrounding photomultiplier tubes (PMTs). To suppress background radioactivity from PMTs and rocks, water surrounds the spherical acrylic vessel. The outer water pool is also equipped with PMTs to veto cosmic ray muon events by detecting Cherenkov light. On the top of the water pool, a top tracker is used to measure muons.

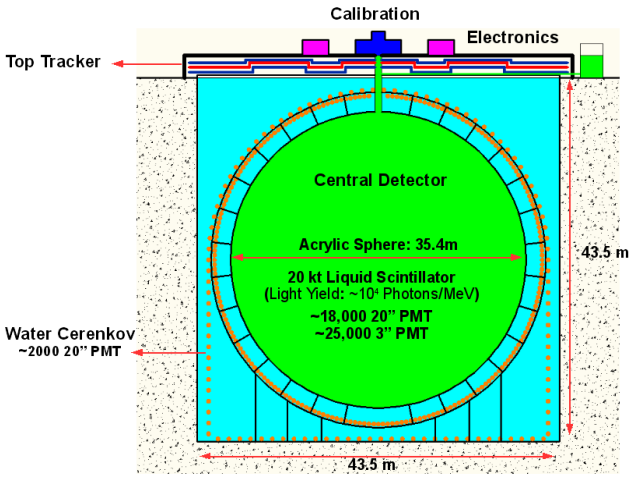

Figure 1. Schematic view of JUNO detector

Offline software is an important part of the JUNO experiment. Simulation software is used for detector design and optimization, algorithm tuning and physics studies. PMT waveform reconstruction is used to calibrate charge and time for each PMT. Reconstruction algorithms are used to reconstruct events and get physics objects for physics analysis. These algorithms are built upon a software framework called SNiPER [3], and compose a full chain of data processing [4], as shown in Figure 2.

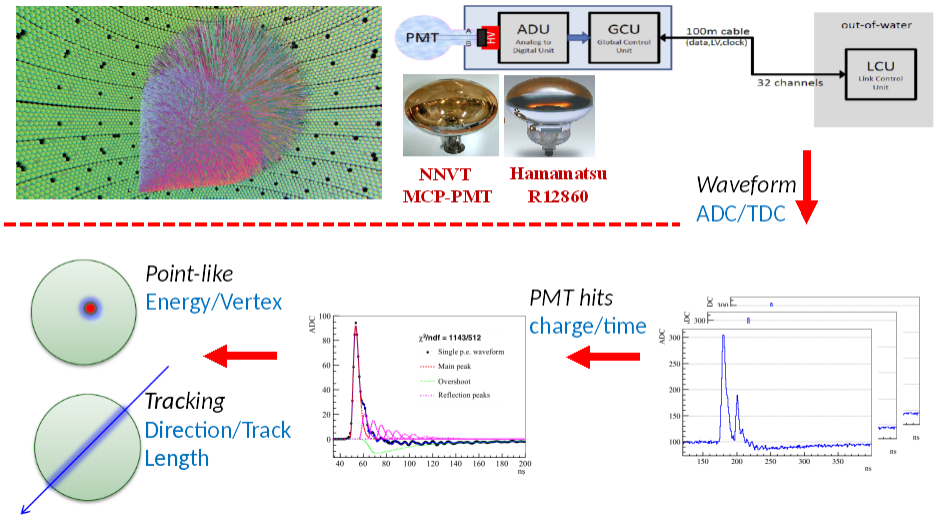

Figure 2. Offline data processing chain

2. Towards parallelized detector simulation framework

A serial detector simulation framework [5] is implemented to integrate SNiPER and Geant4 [6,7]. Figure 3 shows the core class diagram. It is lightweight with only a few classes in the design. Simulation software can be easily migrated from a standalone application into simulation framework. However, simulation of a large detector is not only time consuming, but requires considerable memory and I/O resources as well. After the construction of the full geometry, the physical memory allocation is about 700 MB. More memory is required during the execution of event loop. Especially to the cosmic ray muons, millions of optical photons are produced and propagated, which is a challenge to the memory. Meanwhile, the waveforms are generated with 1 GHz sampling rate and written to files during digitization simulation, which is I/O consuming.

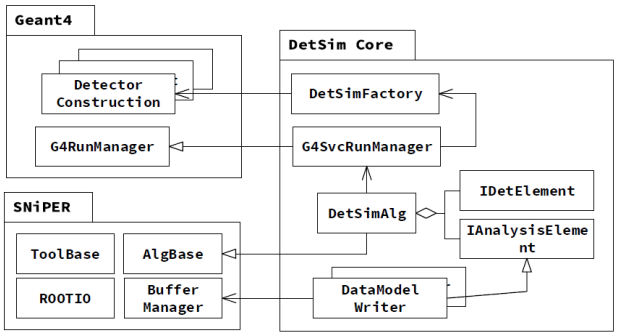

Figure 3. Design of detector simulation framework

A parallelized simulation software is necessary for Monte-Carlo production in the future. It is foreseeable that more and more cores will be integrated in a simple chip, such as the Intel Knights Landing (KNL) processor. But it is not economical to equivalently increase the amount of memory. The memory can be a bottleneck in this situation. According to the results of ATLAS simulation on the Intel KNL, less than 200 workers are used in multi-process jobs due to the limited memory, while 288 threads are used in multi-threaded jobs [8]. In order to use computing and memory resources efficiently, the underlying frameworks adopt the multi-threading model for both the ATLAS and CMS experiments [9–11].

In the SNiPER framework, a parallelized solution based on Intel Threading Building Blocks (TBB) [12] is proposed. Intel TBB supports task-based programming, which simplifies the thinking of parallel computing. In SNiPER, `Task` is an important component, which is like a lightweight application manager. In sequential mode, a `Task` is configured by the user and then invokes registered algorithms. To support parallel computing, SNiPER Muster (Multiple SNiPER Task Scheduler) is designed and implemented to integrate the SNiPER `Task` component and Intel TBB worker. Multiple instances of `Task` components are created before execution. The SNiPER Muster does not execute these `Task`s directly. It creates corresponding TBB-based workers to execute the `Task` component. For better performance, each worker object is recycled rather than allocated for each event, which is called *contiuation* in the Intel TBB terminologies. At the beginning of each recycling, the worker will takes over an idle SNiPER `Task` component in priority order. As shown in Figure 4, the first worker takes over an I/O `Task`, and it will not execute a regular `Task` until the next recycling.

Starting from version 10.0, Geant4 supports multi-threaded simulation [13]. Events are simulated in different threads. With the evolution of Geant4, it is possible to run simulation with Intel TBB, MPI and so on [14]. A parallelized simulation framework is developed to integrate the latest Geant4 and SNiPER Muster, which takes full advantages of parallel computing. Based

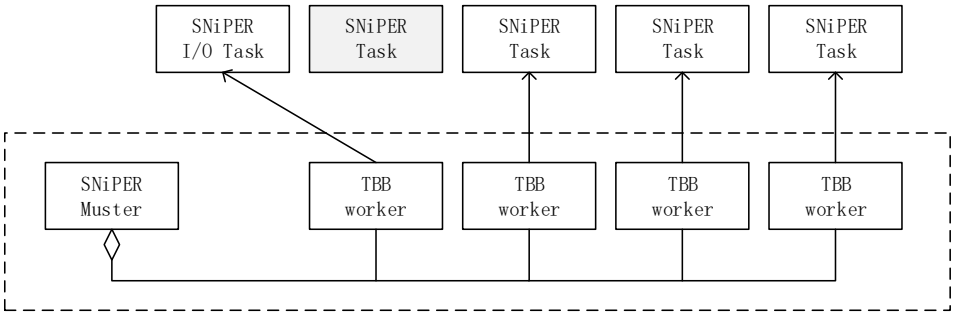

Figure 4. SNiPER Muster

on SNiPER Muster, the simulation framework is decoupled from the underlying Intel TBB, and becomes more flexible and simpler.

3. SNiPER Muster based simulation framework

The parallelized simulation framework developed is based on SNiPER Muster. Several new classes are introduced, compared with the serial simulation framework. The event loop is still controlled by SNiPER instead of Geant4, however, the control flow is rather different from the serial simulation framework. As shown in Figure 5, the simulation framework consists of a global task and several worker tasks. The global task is in charge of initialization of the detector geometry and physics lists. A customized master run manager is derived from Geant4's `G4MTRunManager` to take control of the simulation. A corresponding service is created, so that SNiPER can initialize run manager automatically.

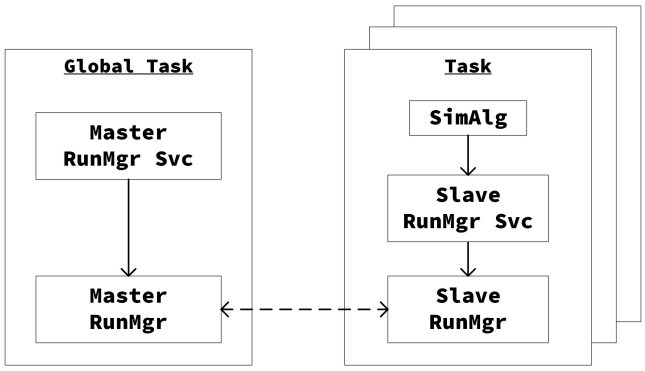

Figure 5. Design of parallelized simulation framework

After the initialization of the global task, SNiPER Muster starts the worker tasks. Each worker manages a simulation algorithm and a slave run manager service. The simulation algorithm can invoke the slave run manager via the corresponding service. The slave run manager is derived from Geant4's `G4WorkerRunManager`, which breaks the original event loop and simulates one event each call. Instead of executing a fixed number of events in each worker, the execution of worker is controlled by SNiPER Muster. If there are events to be simulated, the worker will invoke the simulation algorithm. Otherwise, the worker will be stopped. Hence, this event dispatching procedure can make full usage of all available CPUs.

4. Performance measurements

In these performance measurements, 1000 events of single γs with an energy of 2.2 MeV are generated at detector center. The simulation runs on a blade server with Intel Xeon CPU E5-2680 v3 @ 2.5 GHz and 64 GB of memory. The operating system is Scientific Linux 6.5 with GCC 4.9.4. The Geant4 version is 10.03.p01, built with `global-dynamic` TLS (Thread Local Storage) model. The physics lists `G4EmStandardPhysics` and `G4OpticalPhysics` are used in simulation. An average of three measurements is used for each case. In order to eliminate I/O interference, the events are not saved into ROOT files.

Figure 6 shows speedup ratio versus number of threads. The dotted line represents the ideal result. The gray line with full square represents the result without any optimization. Up to four threads, the speedup is good. However, when increasing the number of threads to 24, the speedup ratio is stuck around 8.

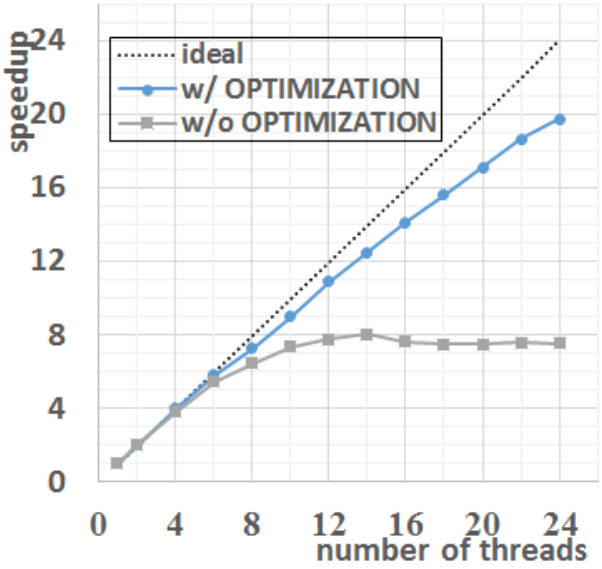

Figure 6. Speedup versus number of threads

The Intel's VTune Amplifier profiler is used to analyze this performance issue. We find that the hotspot comes from a mutex used in Geant4's `G4MaterialPropertiesTable`. This mutex is used when each optical photon calculates its group velocity. The problem is that there are a lot of optical photons in JUNO simulation, so this mutex is accessed frequently in order to get the velocity. When there are multiple threads accessing the mutex, the lock becomes a bottleneck. The optimization is moving this part into initialization stage. When refractive index is set, the group velocity is precalculated during initialization. As shown in Figure 6, the blue line with full circle represents the result with optimization. After this optimization, better performance scaling is achieved.

5. Conclusions

In this paper, we show how JUNO simulation software is parallelized based on SNiPER Muster. By introducing several new classes, Geant4 10 is integrated into SNiPER Muster. The global task initializes detector geometry and physics processes while the worker is invoked by SNiPER Muster to simulate one event. Events are dispatched dynamically to different workers, which

improves CPU utilization. The software performance studies show that there are performance issues due to a mutex in material properties table. After an optimization, the simulation software achieves a close to linear speedup, which fulfills requirements.

To speedup events such as cosmic ray muons, we are investigating how to use track-level parallelism in the simulation framework. The idea is to use MPI to dispatch tracks from the master node to different worker nodes. In the future, MPI and TBB can be applied simultaneously in different levels of the JUNO parallelized simulation software.

Acknowledgments

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, 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** 072053
[4] 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 https://pos.sissa.it/282/1051
[5] Lin T, Zou J, Li W, Deng Z, Fang X, Cao G, Huang X and You Z (JUNO) 2017 *22nd International Conference on Computing in High Energy and Nuclear Physics (CHEP 2016) San Francisco, CA, October 14-16, 2016* (*Preprint* 1702.05275) URL http://inspirehep.net/record/1514060/files/arXiv:1702.05275.pdf
[6] Agostinelli S *et al.* (GEANT4) 2003 *Nucl. Instrum. Meth.* **A506** 250–303
[7] Allison J *et al.* 2006 *IEEE Trans. Nucl. Sci.* **53** 270
[8] Farrell S, Calafiura P, Leggett C, Tsulaia V and Dotti A (ATLAS) 2017 *J. Phys. Conf. Ser.* **898** 042012
[9] Calafiura P, Lampl W, Leggett C, Malon D, Stewart G and Wynne B 2015 *J. Phys. Conf. Ser.* **664** 072031
[10] Stewart G A *et al.* (ATLAS) 2016 *J. Phys. Conf. Ser.* **762** 012024
[11] Sexton-Kennedy E, Gartung P, Jones C D and Lange D 2015 *J. Phys. Conf. Ser.* **608** 012034
[12] Intel threading building blocks https://www.threadingbuildingblocks.org/ [Online; accessed 19-March-2018]
[13] Allison J *et al.* 2016 *Nucl. Instrum. Meth.* **A835** 186–225
[14] Dotti A, Asai M, Barrand G, Hrivnacova I and Murakami K 2016 *Proceedings, 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2015): San Diego, California, United States* p 7581867 (*Preprint* 1605.01792) URL http://inspirehep.net/record/1456144/files/arXiv:1605.01792.pdf