The SNiPER Offline Software Framework

— *Software for Non-collider Physics Experiments*

J. H. Zou[1] (zoujh@ihep.ac.cn), X. T. Huang[2] (huangxt@sdu.edu.cn), W. D. Li[1] (liwd@ihep.ac.cn)

[1]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, China, [2]Shandong University, Jinan, China

Introduction

SNiPER (the abbreviation of Software for Non-collider Physics ExpeRiments) has been developed based on common requirements from both cosmic ray and nuclear reactor neutrino experiments. Compared to the existing offline software frameworks in the high energy physics domain, the design of SNiPER is more focused on execution efficiency and flexibility.

SNiPER has an open structure. User applications, such as simulation, reconstruction and analysis, are executed as plug-ins based on it. The framework contains a compact kernel for software components management, event execution control, job configuration, common services, etc. Some specific features are attractive to non-collider physics experiments.

Moreover, we are working on a hierarchical parallel computing model, which will support the conjugation of MPI (Message Passing Interface), Multi-Threads and GPGPU (General Purpose Graphic Processing Unit) at different execution stages.

Implementation

Nowadays, mixed programming with multiple languages is practical, so we can choose different technologies for different parts of our software.

- C++ — kernel and functional modules to ensure efficiency
- Python — friendly and flexible user interface

We design and manage SNiPER modularly. Every functional element is implemented as a module, and can be dynamically loaded and configured. Modules are designed as high cohesion units with low couplings between each other. They communicate only through interfaces, as show in Fig.1. We can replace or modify one module without affecting any of the others.

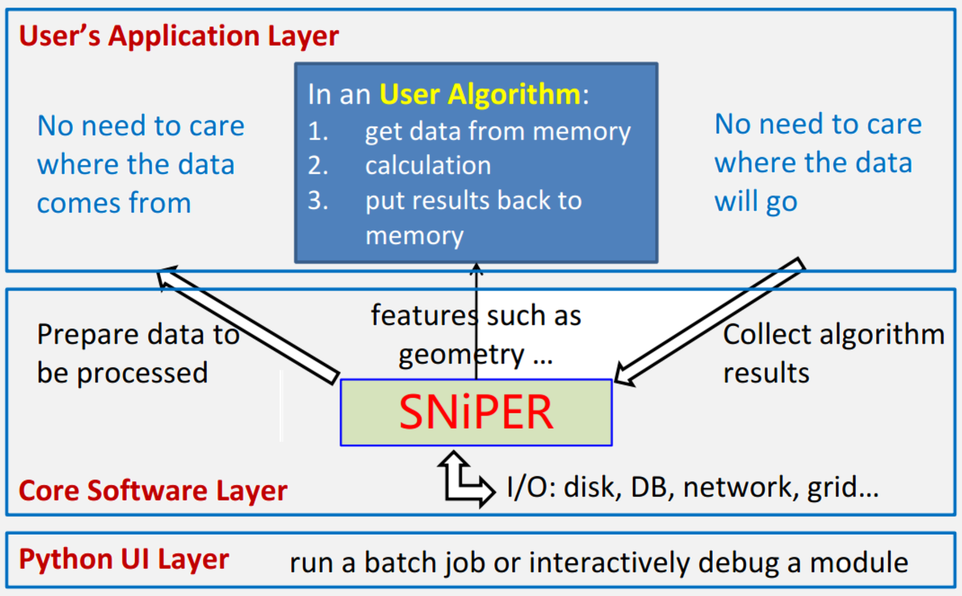

Figure 1: An overview of SNiPER

1. Software Components Management

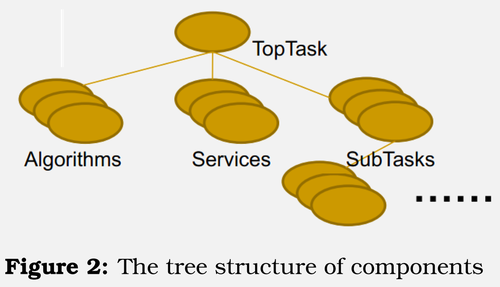

Figure 2: The tree structure of components

In SNiPER, a job is composed of one or more low coupling tasks, and various services and algorithms can be specified in each task. We can achieve some complex goals in a simple way by this mechanism. For example, when there is only single-stream I/O service, we can simply combine several tasks to support multi-stream I/O.

It is recommended to organize all tasks in a tree structure in a job configuration, as show in Fig.2. Each component (such as a task, algorithm, service and so on) is assigned a unique path style string name, and can be retrieved with its absolute or relative path.

2. Event Execution Control

Algorithms and services are plugged and executed dynamically. They can be selected and combined flexibly for different requirements.

The incident mechanism is deployed to trigger some specific procedures in SNiPER. Together with a group of task instances, conditional execution of algorithm subsets can be easily realized, as show in Fig.3.

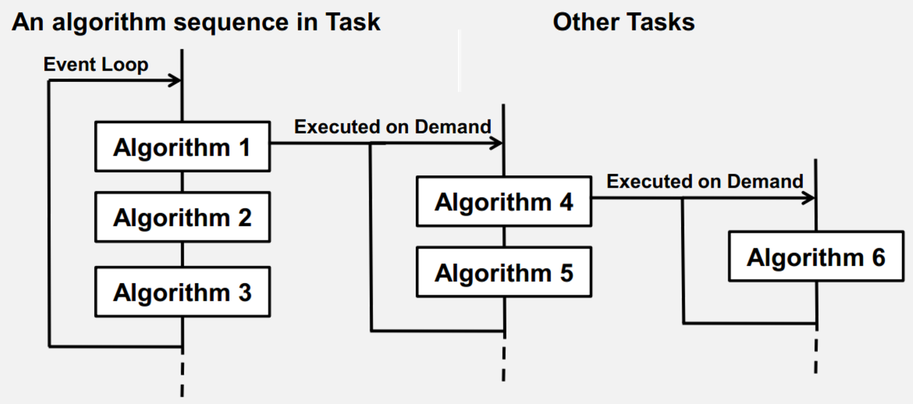

Figure 3: Conditional execution of algorithm subsets

3. Data Memory Management

Both event data model and in-memory data management can be customized by different applications.

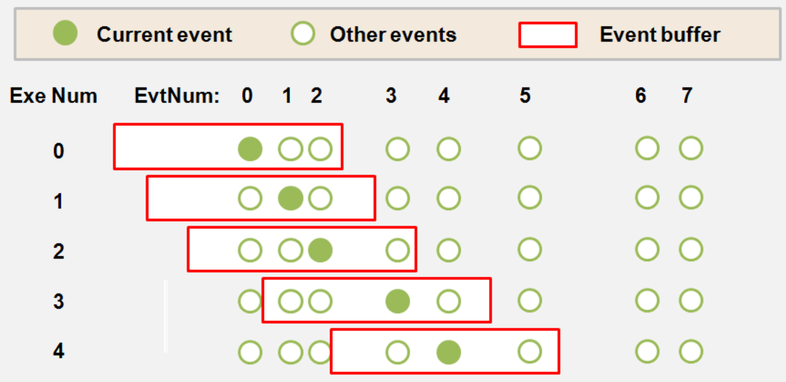

Figure 4: Event buffer with a time window

However, in order to facilitate event correlation analysis (especially for neutrino physics), a FIFO data buffer is used to store adjacent events and a sophisticated method of memory updating is applied, as show in Fig.4. So that accessing to successive events within the user-defined window according to event timestamp becomes possible.

4. Common Services

The framework involves many frequently used functions, such as the logging mechanism, particle property lookup, system resource loading, database access and histogram booking, etc. There are also some frequently used external libraries, such as ROOT and Geant4. All these contents are wrapped in services in SNiPER, which will ease the development procedure.

Plans for Parallel Computing

Parallel computing became pragmatic in recent years. Many different technologies can be used to accelerate our application. We are going to provide a compatible structure to conjugate distributed computing, multi-thread computing and GPGPU computing, as show in Fig.5.

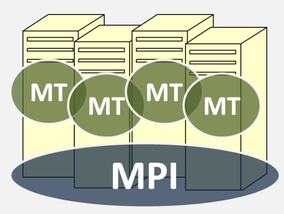

Figure 5: MPI & MT

- MPI can be used to implement a server/clients model for a distributed computing system. So that resources are available across nodes in PC clusters for a single job.
- Multi-thread programming is able to extract the capability of multi-core CPUs. Boost.Thread will be used to simplify this mission.
- GPGPU is powerful to process large amount of simple concurrent calculations. The new framework will also involve CUDA for such algorithms.

Conclusions

Currently a nascent product of SNiPER has been released and is being used by the JUNO and LHAASO experiments. The practices show that the software architecture is universal and expandable. As a general purpose framework, SNiPER can be used by non-collider physics experiments to build their offline data analysis and processing systems. Furthermore, the coming releases will involve some fantastic features for parallel computing.