Implementation of parallel processing in the \textit{basf2} framework for Belle II

Ryosuke Itoh\textsuperscript{1} and Soohyung Lee\textsuperscript{2} with
N.Katayama\textsuperscript{3}, S.Mineo\textsuperscript{4}, A.Moll\textsuperscript{5}, T.Kuhr\textsuperscript{6}, and M.Heck\textsuperscript{6}

Institute of Particle and Nuclear Physics, KEK\textsuperscript{1}, Department of Physics, Korea University\textsuperscript{2},
Kavli IPMU\textsuperscript{3}, U. of Tokyo\textsuperscript{4}, MPI\textsuperscript{5}, KIT\textsuperscript{6}

Abstract.

Recent PC servers are equipped with multi-core CPUs and it is desired to utilize the full processing power of them for the data analysis in large scale HEP experiments. A software framework \textit{basf2} is being developed for the use in the Belle II experiment, a new generation B-factory experiment at KEK, and the parallel event processing to utilize the multi-core CPUs is in its design for the use in the massive data production. The details of the implementation of event parallel processing in the \textit{basf2} framework are discussed with the report of preliminary performance study in the realistic use on a 32 core PC server.

1. Introduction

High energy experiments these days, such as the Belle II experiment at KEK, yield a huge amount of data for the cutting-edge search for the evidence of New Physics. They require a large CPU power for the data processing. Meanwhile, the multi-core CPU becomes a commodity hardware today which can provide the processing power. However, to utilize the multi-core, it is necessary to implement a mechanism of parallel processing in the application program.

The event-by-event parallel processing utilizing the multi-cores is included in the original design of the analysis software framework for Belle II experiment, \textit{basf2} \cite{1} so as to use the framework for the massive production applications. This contribution describes the detail of the implementation of parallel processing in \textit{basf2}. The results of the preliminary study of parallel processing performance using a PC server equipped with 32 cores are also reported.

2. SuperKEKB and Belle II

SuperKEKB/Belle II is a new-generation B-factory experiment in Japan which is being constructed at KEK. A new accelerator SuperKEKB is built by upgrading existing KEKB aiming at a luminosity of more than 40 times higher. The new detector system, Belle II, is designed as shown in Figure 1 to cope with a high hit rate up to a few MHz by the wide use of the pixelized detection device which yields event data of a few hundred kB at a Level 1 trigger rate up to 30kHz. The data flow rate at the storage is estimated to be about 1GB/sec after the HLT reduction.
A common analysis framework named basf2 is developed for the wide use in the Belle II experiment from the Monte Carlo and DST productions, the user analysis, to the real time applications in DAQ such as the HLT framework. It has a software bus architecture so that a large scale application is realized by plugging a set of “modules” into the framework each of which serves as a building block of the application as shown in Fig. 2. The execution of the modules is controlled by a “path” where the processing order of modules is defined, with a conditional branch to other paths by looking at the output of the module.

The data passed between modules are ROOT[2] objects and are managed by “DataStore” object manager. It registers the ROOT TObject and TClonesArray in the event-by-event transient list on the memory so that they can be referred to globally from any modules.

The persistence of DataStore objects in the storage device is supported by input and output modules of basf2. For now, two different I/O modules are implemented. One is the modules to store/retrieve objects in ROOT TBranches written in a TFile which is called ROOT I/O. The
other is the I/O packages called Sequential ROOT I/O, where the objects in DataStore in one event are streamed by TMessage into a byte stream record and stored as sequential records in a file. Since the performance of the Sequential ROOT I/O modules is much better than that use ROOT TFile, they are used for the study of the parallel processing performance described in later sections.

For the execution control of basf2, Python is used as the command line interpreter. The framework functions are modularized and hooked to Python functions so that the framework can be used as tools in a wider data processing chain described in Python combined with other packages such as PyROOT.

4. Design of parallel processing in basf2
The parallel event processing in basf2 is designed to satisfy following requirements:

• A trivial event-by-event parallel processing is adopted.
• The parallel processing is implemented to use multiple Linux processes (event processes) forked after the initialization. They are supposed to run in different CPU cores by the Linux SMP kernel. The memory context at the initialization is inherited to each event process, while actual event processing in each event process does not share the memory context.
• The switching between the normal (i.e. without parallel processing) and parallel processing modes is supposed to be transparent to users. No need to to modify any part of user codes for the parallel processing.
• The same I/O modules can be used both in normal and parallel processing modes.

In order to satisfy these requirements, a simple expansion of the module execution in multiple event processes is not enough. When the parallel processing is performed, the event data read by a single input module have to be distributed to multiple event processes where the actual processing is performed, and the processing results have to be collected to a single output module. To realize this data flow in the parallel processing, the “partial” parallel processing of modules is implemented.

5. Implementation: Partial Parallel Processing
Fig. 3 shows how the module path chain is expanded for the parallel processing in basf2. In the normal processing mode, modules placed in a path are executed in a single process. Each module has a property to flag whether the module is capable of parallel processing or not. When parallel processing is turned on, the module property is examined in turn from the beginning of the path. The modules without the parallel processing property is placed in the “input path”. When a module with the property is found, it is registered in a separate “parallel path”. The examination is repeated until a module without the property is found. The modules hereafter are placed in a separate “output path”. The modules in the parallel path is executed in multiple processes while the modules in the input and output paths are executed in different single processes. All the processes are forked out after the initialization of each module is done.

6. Object Passing between processes
The event processing is started from the first module in the input path running in the “input process”. After the last module in the input path is executed, the objects in the event in DataStore are streamed into a byte stream record using TMessage and placed in a ring buffer. The ring buffer is implemented using the Linux IPC shared memory which is accessible by outside processes. Multiple event processes are hooked to the ring buffer and each process picks up one record whenever available. The record is then decoded to objects and restored in DataStore in the event process so that the consecutive processing of modules in “parallel path” is performed.
Figure 3. Implementation of partial parallel processing in *basf2*. The modules plugged in a path is expanded in the input, parallel, and output paths. The modules in parallel path are executed in multiple processes, while those in input and output path are in the corresponding single process. The streamed event objects are passed between the processes through the ring buffer.

for different events in parallel. The processing output are collected in the output process by the reversed sequence via a separate ring buffer for the post-processing by the modules in the output path. The event passing mediated by the ring buffer ensures the load balancing of the parallel processing resource automatically.

7. Performance Test
The performance of the parallel processing implemented in *basf2* is studied using two different benchmarks as below. A total elapsed time to process 10,000 events is measured for different number of event processes and the relative increase in the performance is measured.

The PC server used for the performance test is equipped with 4 chips of Intel Xeon X7550(2GHz) providing 32 cores with a memory of 65GB. The server is operated with a 64bit Scientific Linux 5.5 (kernel 2.6.18).

7.1. Event Generation + Simulation
In this benchmark, Monte Carlo events are generated by the event generator module in the input process, and the Belle II detector simulation is performed in the multiple processes by passing the generated event objects through the ring buffer. The resulting objects are collected in the output process and stored in a file using the Sequential ROOT I/O package.

The processing time per event is measured to be 0.66sec with a single core yielding an output of ∼500kB/event. The maximum output flow rate is measured to be 23.2MB/sec by the parallel
processing with 28 event processes. The increase of the relative performance is shown in fig. 4-a) as a function of number of event processes. A linearity of the performance increase is well kept up to 28 processes.

7.2. Event Reconstruction
The output of the previous benchmark is used as the input in this benchmark. The data file is read by the input module in the input process and events are distributed to parallel processing paths. A full event reconstruction of the Belle II detector is performed in each event process by a chain of various reconstruction modules, and the results are collected in the output path and stored in a file. The Sequential ROOT I/O package is used both for the input and output.

The processing time per event is measured to be 0.29sec with a single core yielding an output of ∼700kB/event. The maximum output flow rate is measured to be 48.5MB/sec in the parallel processing with 28 event processes. The increase of the relative performance is shown in fig. 4-b) as a function of number of event processes. A linearity of the performance increase is well kept up to 28 processes also in this case.

7.3. Bottleneck in absolute performance
Although the parallel processing performance is kept for two realistic benchmarks described above, there are two sources of performance bottleneck.

(i) I/O bandwidth:
The maximum I/O bandwidth of the PC server is a major bottleneck of the total performance. Currently there are two implementations of I/O subsystem for basf2 (ROOT I/O and Sequential ROOT I/O) as described in previous section. The performance of them is measured by reading the output file of the benchmark 1 and the results are 8.9MB/sec for ROOT I/O and 253.8MB/sec for Sequential ROOT I/O. The I/O performance for ROOT I/O is not tuned yet and it is expected to be improved.
Object streaming overhead:

In this implementation of the parallel processing, it is necessary to stream and destream objects for the ring buffer in order to pass objects between processes. Since the overhead for streaming by ROOT is considered to be non-negligible, it can be another source of the performance bottleneck. The streaming performance heavily depends on the objects to be passed. The case is studied for the objects contained in the output of the benchmark 1, consisting of 13 TObjects and 12 TClonesArrays. The total data flow rate from input to output using Sequential ROOT I/O package is measured for the cases with and without the parallel processing. In the case with the parallel processing (case a), the inputs objects are once transferred to the event process via the first ring buffer, and then copied to the output process through another ring buffer. Therefore, two object streaming/destreaming procedures are in the middle of data transfer. For the case without the parallel processing (case b), the input objects are directly copied to output over DataStore in a single process. The flow rate measured for case a) is 112.4Mb/sec while 253.3MB/sec for case b). The twice intervention by the object streaming for ring buffer deteriorates the data flow performance by a factor of 2 in this case, however, the measured performance is still good enough for the realistic application such as the benchmark 2.

8. Application to Belle II HLT

The object passing method implemented for the parallel processing is applicable to the object transfer between different PC nodes connected over network, by replacing the ring buffer with a network socket connection. By using both of them, the parallel event processing utilizing multi-core CPUs and network-connected PC servers for the parallel processing becomes possible.

This technique is used in the design of Belle II High Level Trigger (HLT) as shown in Fig. 5. The detail is reported in other presentation by S.Lee in the contribution 390 to this conference[3].

9. Summary

The implementation of the parallel processing in the Belle II analysis framework basf2 is required for the use in the massive data production in order to utilize the full processing power of the multi-core CPU. It is implemented in the concept of “partial parallel processing” so that the execution of the partial portion of the module chain is performed in multiple processes in parallel

Figure 5. The parallel event processing using basf2 in Belle II HLT.
while the input and output processing is managed in single processes. The object transfer between processes is implemented by streaming and destreaming objects passed over the ring buffer implemented using the Linux IPC. The parallel processing performance is studied using a 32 core server using two realistic benchmarks, and a good linearity up to 28 parallel processes is confirmed for both cases. The same object passing technique is extended to the network-connected PCs to be used in the Belle II HLT.

[1] A.Moll, “The Software Framework of the Belle II Experiment”, J.Phys.: Conf. Ser. 331, 032024 (2011).
[2] R.Brun, et al, http://root.cern.ch.
[3] S.Lee and R.Itoh, “Belle II High Level Trigger at SuperKEKB”, contribution 390 to this conference (2012).