V. Kůrková et al. (Eds.): ITAT 2014 with selected papers from Znalosti 2014, CEUR Workshop Proceedings Vol. 1214, pp. 97–101
http://ceur-ws.org/Vol-1214, Series ISSN 1613-0073, © 2014 F. Zavoral, J. Yaghob, D. Bednárek, M. Kruliš

Contemplating Efficiency of the ROOT File Format for Data Intensive Simulations in Particle Physics

Filip Zavoral, Jakub Yaghob, David Bednárek, and Martin Kruliš

Parallel Architectures/Algorithms/Applications Research Group
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, Prague, Czech Republic
{zavoral,yaghob,bednarek,krulis}@ksi.mff.cuni.cz

Abstract: A vast majority of nuclear and particle physicists in the world are currently using the ROOT framework (developed in CERN) as a software platform for simulations and data evaluations. Some of the simulations and experiments performed in this domain are very data intensive and they need to be designed with particular emphasis on the processing performance. We have analyzed the efficiency of the Background Mixer component employed in particle simulations of the Belle II project and identified, that the ROOT file format is unacceptably inefficient. The structure of the file and the API presented in ROOT framework prevents efficient data retrieval for parallel processing. We propose a data format which overcomes these limitations and which is much more suitable for high performance computing. We also demonstrate how to integrate this novel format into processing pipeline of the Belle II experiments and present its initial evaluation.

Keywords: particle physics, experiments, ROOT, data format, HPC, Belle II, performance

1 Introduction

Many empirical sciences, especially physics, have been increasingly dependent on the computer science support. In this paper, we focus on a particular problem that have been observed by the particle physicists working on Belle II experiment [1]. Our objective was to improve performance of a Background Mixer component, which is used for particle simulations related to the experiment. We have detected that the bottleneck was the ROOT file format, which is being used for storing the experimental and simulated data, and its API. In this paper, we propose a novel format, which is expected to improve the performance of the Background Mixer module significantly.

The paper is organized as follows. Section 2 briefly introduces the Belle II experiment, its hardware and software support, and the related problems of the data processing. Section 3 addresses the performance issues of accessing objects in the ROOT files. Our new data format (including implementation and integration details) is proposed in Section 4 and Section 5 concludes our work.

2 Technical Background

2.1 Belle II and Basf2

Belle II [2] is a project headed by Ko Enerugi Kasukoki Kenkyu Kiko (KEK) centre in Tsukuba, Japan, that studies the physical process called Charge Parity violation (CP violation). Such process is responsible for breaking the matter-antimatter symmetry, which causes the dominance of matter over antimatter in our universe. The particular task of the project is to search for new sources of CP-violation that could generate the observed asymmetry.

Since the data flow generated by the KEK particle detector is enormous (up to 1.8 GB/s), a distributed computing model comprising high-performance grid sites located all over the globe has been adopted [1] (Figure 1) in order to ensure better scalability, redundancy, and sharing hardware resources of experiment participants.

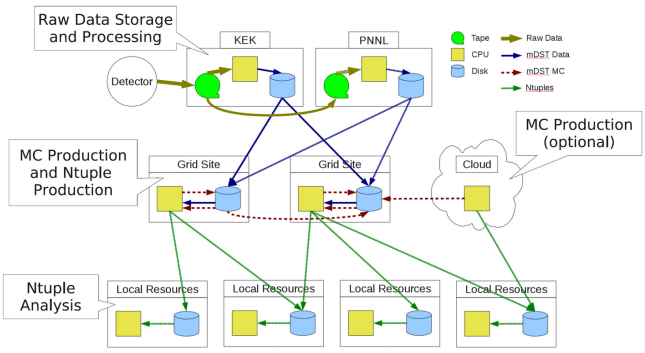

Figure 1: Belle II computing model

A software framework BASF2 (Belle II Analysis Framework) [3] is being developed for the experimental and data evaluation purposes of the Belle II experiment. It is designed to massively process the data using parallel event processing on the multi-core CPUs. The software is written in C++ to meet the requirements of high performance processing and to interface well with other tools used in particle physics. Python scripting language is used for the configuration scripts.

BASF2 is based on a software bus architecture. A large scale application is designed as a set of pluggable modules [4], each of which serves as a building block of the application. The modules are assembled in an execution

pipeline (which the framework denotes *path*), where output of each module is interconnected with input of the subsequent module. The data are passed between the modules as ROOT objects (described in Section 3).

2.2 Background Mixer

One of the essential parts of the experiment is an accurate simulation of the background signal. The background is generated by Monte Carlo method and the produced dataset is mixed with other simulated events. The BASF2 module responsible for this task is called Background Mixer [5].

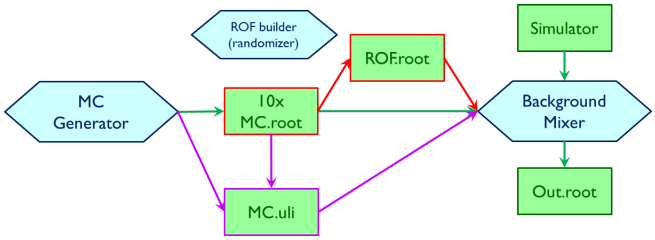

Figure 2: Belle II background mixer module

The presimulated background is added as *SimHits* (simulated events on particular detectors) to the already existing SimHits from the main signal. Both signals are then mixed together, giving a realistic result. A simplified scheme of the background mixing process is depicted in Figure 2.

2.3 Structure of Data Processing

Each BASF2 module has standardized component interface which is mainly responsible for controlling the input and output data streams. When the module path is executed, all relevant datasets are initialized, i.e., their ROOT objects are fetched in the main memory for onward processing.

The input data for the Background Mixer are generated by three module types: *Generator*, *Component*, and *Detector*. The Generator and Component generate main particles and background particles respectively. The Detector module simulates the particle detector and generates SimHit events as they would have been generated by the real hardware. Various types of generators, components, and detectors exist and the Background Mixer input ROOT file keeps the types of the modules that were used for generating the data.

The Background Mixer input file contains three main data streams – *mcParticles*, *mcSimHits*, and *mcPartRels*. The mcParticles stream represents the particles generated by Generator and Component modules. The mcSimHits stream holds the SimHits produced by the Detector. The msPartRels stream provides the n-to-m relation between particles and SimHits (i.e., which SimHits were produced by which particles). The data file usually holds thousands of SimHits per each particle in the stream.

When the Background Mixer is initialized, it receives a list of generated files with background data. Each file was generated with a different combination of Generator, Component, and Detector modules. The Background Mixer then works as a filter of the main event stream. For each particle event in the main stream it loads a frame of particles and SimHits from each of the input files and adds them to the main stream.

2.4 ROOT File Format

ROOT [6] is an object-oriented C++ framework conceived in the high-energy physics community. It was designed for storing and analyzing large volumes of data. It uses a proprietary ROOT file format for data representation. Basically any C++ class/structure can be serialized into a ROOT file in a machine-independent compressed binary representation.

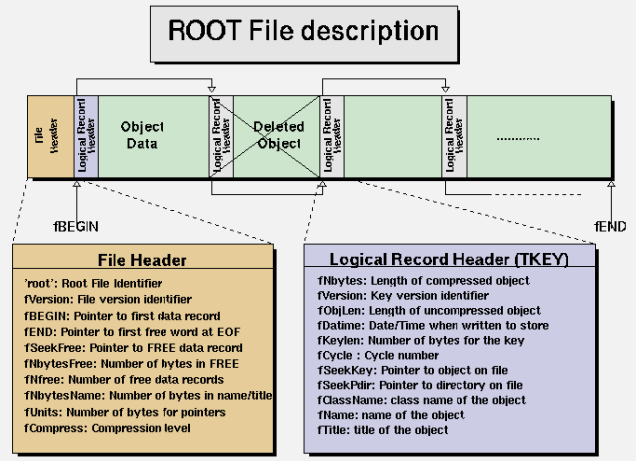

Figure 3: Structure of a ROOT file

Internally, the ROOT files are organized as *TTrees* - containers optimized for I/O and memory usage. A TTree consists of branches (as depicted in Figure 4 and Figure 3) which can contain complete serialized object of a specified class or which can be split up into sub-branches containing individual data members of the original object. The splitting can be done recursively. Branch-based storage is an example of column-wise (vertical) storage.

Much more detailed description of the ROOT file format can be found in [6] and [7].

3 ROOT Format Efficiency

The ROOT file format and associated ROOT libraries have exhibited poor I/O performance in some situations, especially in case of the Background Mixer component.

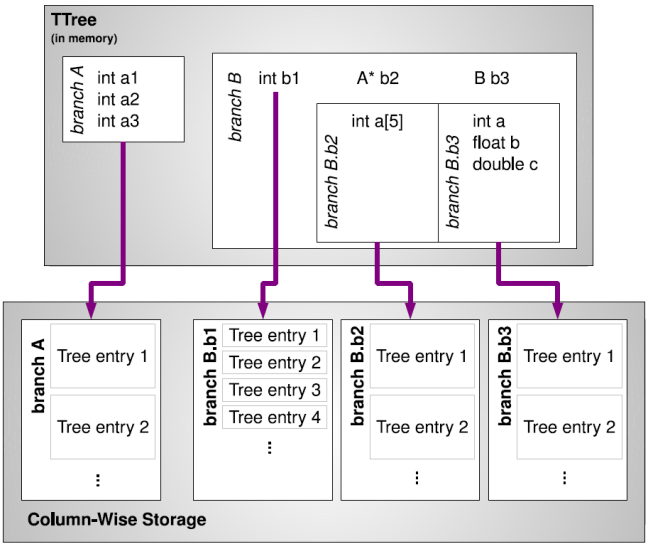

Figure 4: Internal tree structure of a ROOT file

Figure 5 shows how a ROOT file is accessed when its logical structure is read sequentially. The file consists of 216 logical entries which were read in the same order as when they were written. The vertical axis represents the progress of the processing of the logical entries. The horizontal axis represents the data in the binary file image (approx. 1.8 MB large). The graph marks, which portions of the binary image were accessed (I/O operations) when each logical entry was processed.

The figure indicates that the processing of the very first entry induces access to approximately 75% of the file data. However, the processing of subsequent 70 entries were satisfied from this data cached in memory. Some of the following entries induced additional reading of the file, including minor portions that were already read. Nevertheless, the amount of repetitive read operations is almost negligible and some parts of the file were never accessed.

Figure 6 displays the effect of random access to the logical entries of the same file. In this case, entries were processed in randomly permuted order. Almost every event induced significant access to the file. The accessed areas have similar size as in the sequential case; however, there is almost no effect of caching. Consequently, the total disk traffic was 53 MB (i.e. about 30 times the size of the file), which significantly exceeds the 1.7 MB, which were read in the sequential case.

Unfortunately, a random access pattern is currently employed by the Background Mixer since it is mixing data from several independent files. The ROOT API permits the user to handle multiple files at once, but internal limitations and function invariants prevent efficient data caching. According to reports generated by the cluster task management framework, the computational workload uses only 25% of the assigned CPU, which is unacceptably low.

The cost of data processing is a substantial part of the Belle II project expenses. Significant improvement of the efficiency would enable more extensive and influential experiments.

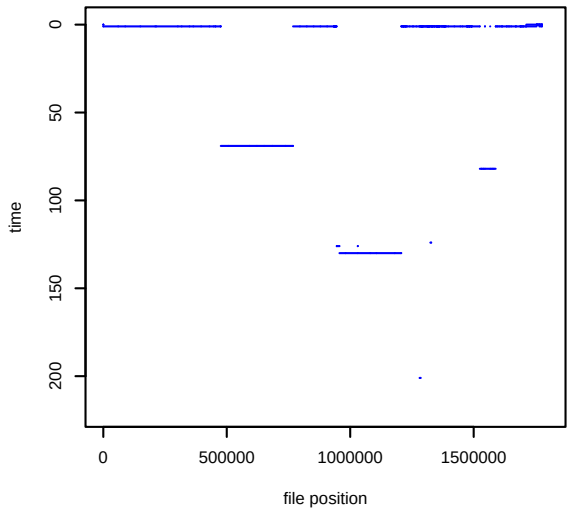

Figure 5: Access pattern to a ROOT binary format in case of serial processing

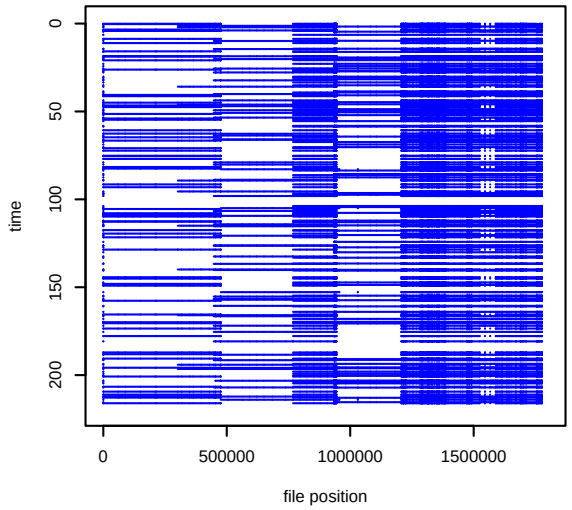

Figure 6: Access pattern to a ROOT binary format in case of random processing

4 Proposed Solution

The ROOT format is tightly coupled with many essential parts of the ROOT framework, so it would be quite impractical to significantly alter its structure or API. To solve the problems observed in the Background Mixer, we propose a novel *GFT* data format, where GFT stands for *GraFT* as we have grafted this new format from the ROOT format. It is designed specifically to preserve all the benefits of the ROOT format and to optimize data I/O operations in many scenarios, especially in case of the Ground Mixer.

4.1 Format Description

The original logical structure of a ROOT file comprised only three data streams (Section 2.3). We have decided to make the format more general, so it can possibly contain arbitrary number of data streams.

Since we are describing binary file format, technical details regarding representation of atomic values have to be observed. All numbers are stored in IEEE 754 representation with little-endian byte ordering. Strings are stored as zero-terminated sequences of 8-bit chars (C string convention). In the remainder of this paper we denote basic integer types as *iXX* (signed) and *uXX* (unsigned), where XX is the number of bits (e.g., *u64* stands for 64-bit unsigned integer).

One GFT file is produced by a combination of three modules: Generator, Component, and Detector (as described in Section 2.3). It is possible to have multiple files generated with the same combination of these three modules, but data in one file has to be generated by a fixed combination.

Figure 7: Structure of GFT file

A GFT file (Figure 7) contains one *main header* and a sequence of HOT (Hierarchial Object Table) data structures. Each HOT represents one set (originally called *frame*) of data, which is mixed with one event from the main stream.

u16	hver	Header version
u32	hsize	Header size
u32	idxoffs	Index offset
str	gcd[3]	Generator, component, detector names
i16	nstreams	Number of streams
str	strname[nstreams]	Streams names
i16	nstrfld[nstreams]	Number of fields in streams
i16	tstrfld[sum(nstrfld)]	Types of fields in streams

Figure 8: Main header of GFT file

The GFT main header (Figure 8) comprise global identification entries followed by the description of data streams stored in the file. Each stream is defined by its name, number of data fields in the stream record, and the data type of the data values in the stream.

The header can optionally contain a HOT index, which can be used for random access to the HOT data structures. If the index is present, its offset from the beginning of the file is stored in the *idxoffs* entry.

Figure 9: Schema of the HOT data structure

Each HOT data structure (Figure 9) contains data for one frame. The data from the streams are restructured into a column-oriented format, which is much more suitable for modern computing architectures including GPGPU. Each data column is stored in a COLD (COLumn Data) structure and represents one data field of one stream.

u16	hver	Header version
u16	hsize	Header size
u64	strnrec[nstreams]	Number of records for each stream
u64	fldoffs[sum(nstrfld)]	Relative offsets of COLD for each stream field

Figure 10: Header of the HOT data structure

The header of the HOT structure (Figure 10) contains only the number of records in each data stream and an index to the COLD data structures. The index is implemented as a sequence of file offsets, which are relative to the beginning of the HOT data structure.

u16	hver	Header version
u16	hsize	Header size
i16	ftype	Field type (copy of tstrfld[x])
i16	hdsize	Header data size
u8	hdata[]	Header specific data (compression table, …)

Figure 11: COLD header of GFT

The COLD data structure comprises a COLD header (Figure 11) and a data payload in column-oriented format. The column data representation permits using specific compression techniques (e.g., delta encoding combined with RLE compression), which is an important component of big data processing. Some forms of compression require additional data (like the compression dictionary), so we have introduced header-specific data (entries *hdsize* and *hdata*) to each COLD structure. The column-data payload is stored directly after the header as a vector of values of given basic data type. The vector is properly aligned to the size of the data type of its items.

4.2 Implementation Details

We have implemented a prototype library for manipulation with data in GFT files. The library has two interfaces (*gft-reader* and *gft-writer*), which are responsible for reading and writing the data respectively. There is currently no

API for direct data manipulation, since the expected usage does not modify the data once they are generated.

The metadata contained in the headers (main header, HOT headers, and COLD headers) are sufficiently small, thus the API caches them for the entire time the file is opened. The payload data are read and written in the granularity of HOT structures. A HOT structure usually takes kilobytes or tens of kilobytes, thus the read/write operations have acceptable overhead.

Current implementation uses standard synchronous I/O of the operating system for reading and writing the file data, since it was easiest for implementation and our first objective was to create a proof of concept only. In the future implementations, we are planning to experiment with the asynchronous I/O and memory mapped files.

Our preliminary experiments indicate that the reading overhead is below 10%, which is much more acceptable than 3000% overhead observed for the ROOT files.

4.3 Integration in Belle Framework

The integration of the proposed format into the processing chain of BASF2 is quite straightforward. We have identified two places where the proposed format could be used.

First, when the background are generated, they will be directly written to the GFT format by the MCGenerator module. The module has to be modified to use *gft-writer* instead of the original ROOT file API. Since the existing pregenerated data for the Background Mixer are intended for the evaluation purposes only, they may be discarded and generated again (no conversion tool is required).

Second, the Background Mixer has to be modified to use GFT format instead ROOT format. The `template <class SIMHITS> class Belle2::background::Generator` has to be reimplemented using *gft-reader*. The performance improvement of this modification should be twofold. The data stored in the GFT format will be accessed almost sequentially and read only once. Furthermore, an additional speedup caused by non-interleaving access to the ROOT library is expected, since the ROOT API will be used solely for accessing the main particle stream.

5 Conclusion

The analysis of the Belle II Background Mixer data flows showed that the underlying ROOT file format is unacceptably inefficient for the purposes of combining data from multiple files together. We have proposed a novel data format GTF which is more suitable for efficient data stream processing and adopts some design features of high performance data formats. At present, the reader and writer APIs of the proposed GFT format are implemented in a form of library and our preliminary performance experiments indicate, that the new format has much lower overhead then ROOT format.

In the future work, we are going to integrate the background processing chain and thoroughly evaluate its efficiency. We expect a significant speedup of the particle simulations and better utilization of the Belle II hardware infrastructure.

Acknowledgment

This paper was supported by Czech Science Foundation (GACR) projects P103/13/08195 and P103/14/14292P and by SVV-2014-260100.

References

[1] T. Kuhr, “Computing at Belle II,” *Journal of Physics: Conference Series*, vol. 396, 2012.

[2] Ko Enerugi Kasokuki Kenkyu Kiko (KEK) - High Energy Accelerator Research Organization. (2014) Belle II Project. [Online]. Available: http://belle2.kek.jp/

[3] R. Itoh, S. Lee, N.Katayama, S.Mineo, A.Moll, T.Kuhr, and M.Heck, “Implementation of Parallel Processing in the Basf2 Framework for Belle II,” *Journal of Physics: Conf. Ser.*, vol. 396, 2012.

[4] D. Y. Kim, “The Software Library of the Coming Belle II Experiment and its Simulation Package,” in *Proceedings of the 2013 IEEE Nuclear Science Symposium*. IEEE, 2013.

[5] P. Kvasnicka, “Background mixing status and plans,” in *15th OM of the Belle II Collaboration, Blacksburgh*. KEK, 2013.

[6] R Brun et al. (2014) Root. [Online]. Available: http://root.cern.ch/

[7] I. Antcheva et al, “ROOT — A C++ Framework for Petabyte Data Storage, Statistical Analysis and Visualization,” *Computer Physics Communications*, vol. 182, no. 6, pp. 1384 – 1385, 2011.