HIPO Data Format Performance Compared to ROOT

Gagik Gavalian[a], Derek Glazier[1]

[a] *Jefferson Lab, Newport News, VA, USA*
[b] *University of Glasgow, Glasgow, UK*

Abstract

In this paper, we present studies of the performance of High-Performance Output (HIPO) [1] data format in writing and reading scientific data from Nuclear Physics experiments in CLAS12 [2]. The performance of HIPO is compared to widely used in the high-energy physics community data format of ROOT [3] files. Our studies show that HIPO reader and writer codes in C++ and Java outperform ROOT by a significant margin.

1. Introduction

Modern High Energy and Nuclear Physics experiments produce an ever-increasing amount of data. This represents challenges in data processing categorization and monitoring. Data collected from experimental setups undergo several iterations of processing data to produce an output that can be used for physics analysis. It is important to use a data storage format able to read and write a large amount of data fast.

2. High Performance Output Data Format

In Nuclear Physics experiments data is stored in "event" representing one interaction of the beam with the target particle. Each event is a record containing variable-length collections of data structures (called banks) each representing processed data from different detector components. In CLAS12 the reconstructed particles stored in such structures are used for the final physics analysis. The event from the data processing stage is processed by data classifying algorithm to produce reduced outputs for each physics group containing different physics event topologies. In CLAS12 experiments the data selection and reduction process (called data trains) is run on a regular basis to provide users with updated data samples to analyze. The data reduction process is primarily reading data, categorizing a writing several data streams, and data both data reading and writing performance is important. The High-Performance Output (HIPO) data format was developed at Jefferson Lab for CLAS12 detector data processing applications. It is an event-based data container with an indexed entry map that allows fast random access to any event in the file with low latency. Two methods of compression are implemented by default for HIPO files: LZ4 and GZIP, for all the tests in this paper we used LZ4 compression due to its high rate of compression and decompression. The HIPO format is used to store raw data information from experimental data acquisition, then processed with a data processing application to produce final physics data for analysis in the same file format. CLAS12 data processing

program is using Service Oriented Architecture (SOA) [4] implemented in Java [5]. The reconstructed data is analyzed using a ROOT-based analysis framework (clas12root) in C++. HIPO supports C++ and Java (native) for reading files and Python. In this paper, we test the performance of HIPO for both libraries (C++ and Java) and compare them to the performance of ROOT Trees, which is the most commonly used data format in High Energy and Nuclear Physics community for storing and analyzing experimental physics data.

3. Benchmark Code

To test the performance of each data format we used a file that contains only one data structure describing the particles reconstructed from experimental data. The data contains 12 columns with particle momentum and vertex components and some auxiliary variables such as : particle id, charge, beta, status, and particle id χ^2. Each entry contains a variable number of rows. The test consists of reading each entry in the tree then looping through a number of particles in each entry and calculating some quantity which involves all the columns in the bank and filling a histogram. The size of the file in each format (HIPO and ROOT) is approximately 2GB, produced using the LZ4 compression algorithm.

The listing above shows ROOT code used to analyze the data using an improved implementation of ROOT Tree [6] provided to the authors through private communications. The code was translated to Java (almost exactly) to benchmark the performance of the HIPO/Java library and was tested using GraalVM 17 platform. The benchmarks were run on different platforms including M1, AMD, and Intel/Xeon architectures. We also run benchmarks in two different scenarios where the data was read from NFS disks (AMD and Intel/Xeon tests) as well as reading the data from the local SSD hard drive (all MacBook tests). Several runs were performed with a warmup cycle, then an average of 10 tests was taken to produce comparative plots.

4. Reading Benchmarks

To benchmark file reading performance we used a file from production data that was filtered to contain only one structure (bank) that contains information on reconstructed particles in the CLAS12 detector containing $7M$ events and a size of 2.1 Gb. The HIPO file was converted to a ROOT file using code provided by the ROOT team and analyses are run using the code described in the "Benchmark Code" section. The benchmark is run on different platforms and different environments to emulate standard everyday data processing scenarios. The platforms used are:

- MacBook Pro 16, Apple M1 Max (reading from SSD)
- AMD EPYC 7502 32-Core Processor (NFS mounted luster)
- Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz (NFS mounted luster)

On MacBook computers, the data is read from a local SSD drive. The AMD and Intel/Xeon machines used in these benchmarks are part of the Jlab computing cluster and the

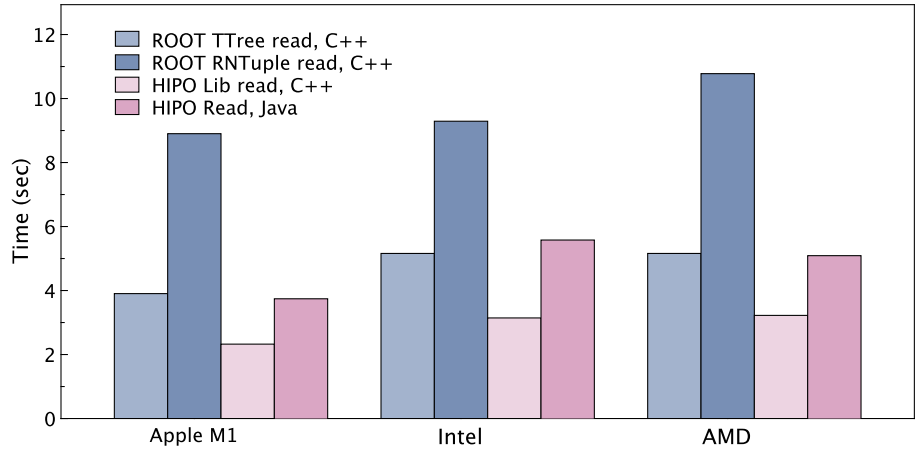

Figure 1: Reading times for ROOT and HIPO compared for different platforms for C++ and Java. RNTuple is used for ROOT with LZ4 compression.

data is read from NFS-mounted disks. The results of benchmarks are shown in Figure 1, where the execution time for each benchmark is shown in seconds.

Few observations can be made from the Figure. The analyses in Java are always faster than code running in C++ using ROOT I/O. The HIPO C++ library outperforms ROOT I/O significantly, peculiarly the difference in performance is architecture-dependent. On M1 and AMD architecture the HIPO is about 25%-30% faster than ROOT, on Intel architecture the ROOT is more than twice slower. If we consider AMD and Intel tests are done in the same environment of Jlab computer cluster in the same conditions of reading the files from NFS disks, it's surprising that HIPO performance is similar ($\pm 2\%$) while the ROOT I/O performance changes dramatically (almost twice) going from AMD to Intel Xeon. Interestingly Java library also performs the same on AMD and Intel architectures. These results are very surprising and hard to explain.

5. Writing Benchmarks

The writing speed of the data format is important in different workflows of the data analysis process, especially in I/O bound workflows. In CLAS12 we frequently process the data through data "trains", where we create different data samples, depending on the reconstructed event topology, for different experimental groups to analyze. The entire reconstructed data set is processed and several (typically of the order of 14-18) separate data sets are written to the disk. In these kinds of applications, the data writing speed is also important since there is very little processing done by "trains" other than checking if the event matches some topology criteria and the majority of processing time is spent on reading and writing.

We also run some tests to compare the performance of the ROOT I/O against HIPO running in C++ and Java. The results can be seen in Figure 2.

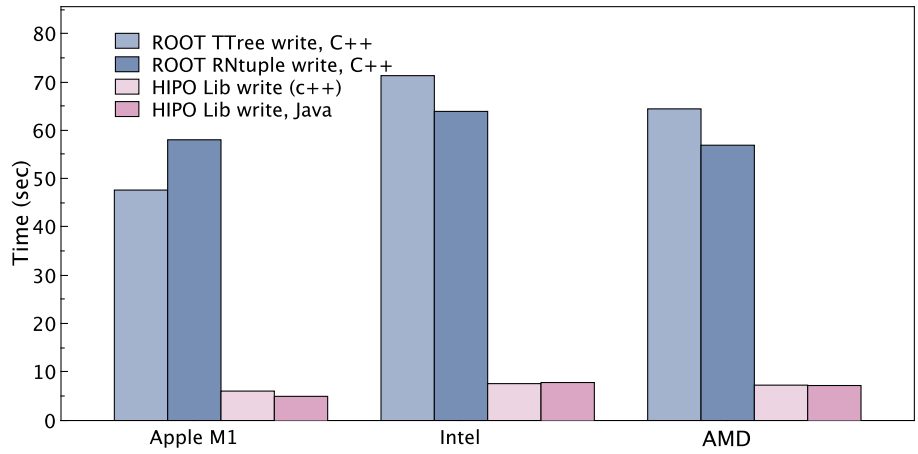

Figure 2: Writing times for ROOT and HIPO compared for different platforms for C++ and Java. RNTuple is used for ROOT with LZ4 compression.

The results show that the MacBook M1 has the best writing speeds due to its very fast SSD. However, in all of the tests HIPO C++ and HIPO Java writers outperform ROOT writers significantly. The surprise of this benchmark is that the writing speeds of ROOT files on AMD and Intel Xeon architectures are not very different, as it was for reading speeds. Both data formats use the same LZ4 compression algorithms, and in general writing speed will depend on the level of compression requested from the library. For both samples, we used highest level of compression and we ended up with similar file sizes for both data formats, the ROOT file being slightly larger 2,119 MB vs 2,033 MB in the HiPO data format.

6. Discussion

In this article, we compared the performance of the ROOT tree and HIPO data format. In our study, we read the entire bank (all branches in ROOT) from both files and used all the variables to perform some calculations. In this scenario, the HIPO outperforms the ROOT tree by about 30% in reading speed. The Java implementation of HIPO also outperforms the ROOT tree in reading speed. The HIPO data format is also ~ 8 times faster in writing speed in both implementations of C++ and Java. At the early stages of CLAS12 software development, the data was converted to ROOT for final analysis, which leads to significant empty computation cycles, and leads to data structures that are slower to process. With the development of CLAS12ROOT package (in C++) the HIPO files can be easily read into the ROOT environment and analyzed using data analysis and visualization tools available in ROOT.

It is important to note that for data visualization when only one variable is read from the tuple-like structure and plotted, the ROOT tree outperforms the HIPO tree. The HIPO data format is primarily designed to read a collection of data and do analysis and for that purpose its performance is adequate to be used for CLAS12 experimental needs in all stages

of data processing, starting from data acquisition, reconstruction code, and final physics analysis.

References

[1] G.Gavalian, "High Performance Output Data Format." `https://github.com/gavalian/hipo`.

[2] V. Burkert *et al.*, "The CLAS12 Spectrometer at Jefferson Laboratory," *Nucl. Instrum. Meth. A*, vol. 959, p. 163419, 2020.

[3] R. Brun and F. Rademakers, "ROOT: An object oriented data analysis framework," *Nucl. Instrum. Meth. A*, vol. 389, pp. 81–86, 1997.

[4] V. Gyurjyan, D. Abbott, J. Carbonneau, G. Gilfoyle, D. Heddle, G. Heyes, S. Paul, C. Timmer, D. Weygand, and E. Wolin, "CLARA: A contemporary approach to physics data processing," *J. Phys. Conf. Ser.*, vol. 331, p. 032013, 2011.

[5] V. Ziegler *et al.*, "The CLAS12 software framework and event reconstruction," *Nucl. Instrum. Meth. A*, vol. 959, p. 163472, 2020.

[6] J. Blomer, P. Canal, A. Naumann, and D. Piparo, "Evolution of the ROOT Tree I/O," *EPJ Web Conf.*, vol. 245, p. 02030, 2020.