ANALYSIS OF THE EFFECTIVENESS OF VARIOUS METHODS FOR PARALLELIZING DATA PROCESSING IMPLEMENTED IN THE ROOT PACKAGE

T.M. Solovjeva

Joint Institute for Nuclear Research, 6 Joliot-Curie St., Dubna, Moscow Region,141980, Russia

E-mail: tanysha@jinr.ru

The ROOT software package is currently being upgraded in several ways to improve data processing performance. This paper will consider several tools implemented in this framework for calculations on modern heterogeneous computing architectures. PROOF (Parallel ROOT Package Extension) divides common work into small chunks, i.e. packets. The size of the first packet used for calibration, the minimum and maximum set size of the packet, the degree of data structuring affect the speed of their processing. When processing large amounts of data, the read and write speed can be crucial. The new asynchronous file merge feature in the TBufferMerger class allows writing data in parallel from multiple streams to a single output file. Our calculations show a good scalability of the macro execution time depending on the number of processor cores used.

Keywords: ROOT, PROOF, parallelization

Tatiana Solovjeva

1. Introduction

The ROOT [1] software package plays a central role in the analysis of high-energy physics data. At the dawn of its development, ROOT was formed as a single-threaded application. However with an increase in the amount of data processed and the development of computing technology, it became obvious that the framework needed modernization be able to take advantage of modern computing architectures. At present, multiprocessor computing systems, which are a set of fairly powerful computers with distributed memory and distributed control, are widespread. For their effective use, it is necessary to take into account the peculiarities of data structuring and the sequence of stages of their processing for each specific analysis. This article explores various parallelization techniques implemented in the ROOT package to improve data processing performance. Performance tests were carried out on the HybriLIT heterogeneous cluster [2] of the Meshcherykov Laboratory of Information Technologies of the Joint Institute for Nuclear Research.

2. Data structuring for processing

PROOF [3], Parallel ROOT Facility, is a special ROOT tool that uses natural parallelism of data structures located in files of a special format and providing direct access to any individual value. In our previous work [4], we compared the efficiency of data processing using PROOF and a special class for threading, as well as the OpenMP technology. It showed the high performance of PROOF, if the analysis contained a large number of mathematical operations and was performed on a large amount of data. The present article will study the question of whether the PROOF performance depends on the way the data file is organized, and also consider the possibilities of speeding up data writing, provided this process is parallelized.

The calculations were performed using the ROOT 6.13 and PROOF-Lite versions. For their operation on the HybriLIT cluster, six CPU cores were reserved. PROOF was running at different values of the number of workers. As long as the number of workers started does not exceed the number of cores, one worker process completely occupied the processor core. If, when starting PROOF, more workers than the number of reserved cores were called, then naturally; several workers were sharing one processor core. The acceleration factor (the ratio of the execution time of a macro in a non-parallel version and the time of a macro in a parallel version) was actively growing with an increase in workers from two to six, then its value stabilized. When starting PROOF with more than 12 workers, the acceleration rate decreased smoothly. The data shown in the graphs was obtained with workers equal to six.

To represent data in ROOT, a special class TTree was developed, it was optimized to reduce disk space and increase the data access rate. All variables are presented in the form of leaves, which are combined into branches, and the collection of branches forms a tree. The tree storage approach is good for parallel architectures since each branch can be read independently of the other. The tree can have different structures. We were interested in the question of how the organization of data in a tree affected the effectiveness of using PROOF. We considered a simple tree with a list of variables, a tree with structures and a tree with class objects.

All of our trees contained twelve variables structured in different ways. The simple tree had twelve branches, where a floating-point variable acted as a leaf. In the second case, the tree had three branches, the first two of which contained three leaves each, combined into structures, and the third branch was a structure of six float variables. In the third case, the tree had one branch that contained an object consisting of all twelve variables. The data was contained in ten files, each of which was approximately 1.6 GB in size. We performed two analyzes. In the first of them, data was read from all branches, they were processed and the result was written to a new file. In the second analysis, the data of only two variables was read, and in the case of the tree with a structure, we considered two options for reading, namely, in the first, two variables belonging to the same structure were read and processed, and in the second case, the variables belonged to different structures. The processing included calculations of various physical quantities commonly used in different types of physical analysis.

Figure 1 shows the acceleration factor of the macro execution depending on the amount of processed data on the left for the first analysis, on the right for the second analysis.

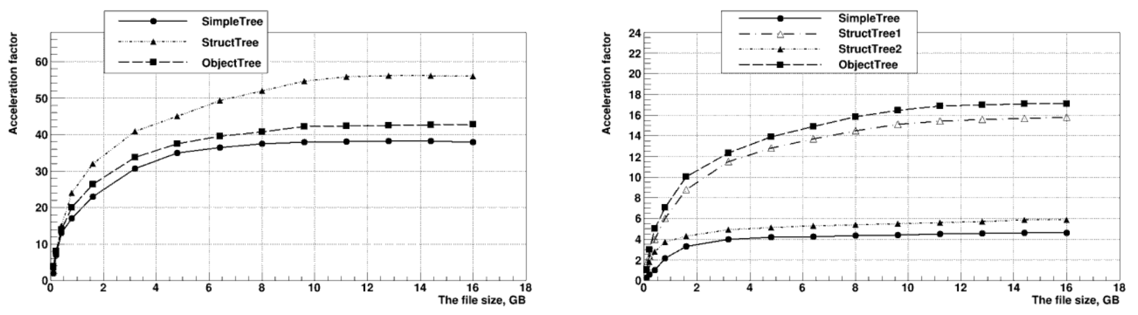

Figure 1. Dependence of the macro acceleration on the type of data structuring

As seen from the graphs, when reading and processing all data using PROOF, the acceleration factor has a maximum value if the data is organized into structures. If only individual columns of root files are read and processed, then the maximum speedup factor is achieved when organizing the data in the form of a tree containing objects.

3. Parallel reading and writing

Different types of analysis have different ratios of time over which different processing steps are performed. When processing large volumes, sometimes the time spent on reading or writing data comes to the fore. New possibilities for parallelizing these processes are actively used in the optimization of HEP software [5-7].

A distinctive feature of ROOT, which made it easy to use, is its columnar data format. The user has the ability to read the data of only those columns that are of interest to him. In the process of reading, the data is unpacked and deserialized. Starting from version 6.08, functions that perform these actions in parallel have been implemented in ROOT. The user only has to specify the number of threads. All implementation details are hidden from the user, which is why this approach is called Implicit Multithreading.

For parallel writing, the TBufferMerger class, implemented in ROOT since version 6.10, is used. This class implements the following scheme of actions. The user specifies how many streams that write data to a single file to create. The generated worker threads divide the data into separate buffers, in each of which the data is serialized and compressed. Therefore, these processes occur independently in each buffer, so they can be performed in parallel. The buffers are merged before closing the output file. In the original version, the merge had a separate output stream. Then a scheme, in which the merge is done by the worker threads themselves on demand, was implemented. This made it possible to reduce the required computing resources.

Our performance tests consisted of generating data, organizing it into a tree with twelve branches, and then writing it to a file. Figure 2 on the left shows the dependence of the running time of a macro on the number of worker threads and the number of generated events. The dependence of the acceleration coefficient on the same factors is illustrated on the right.

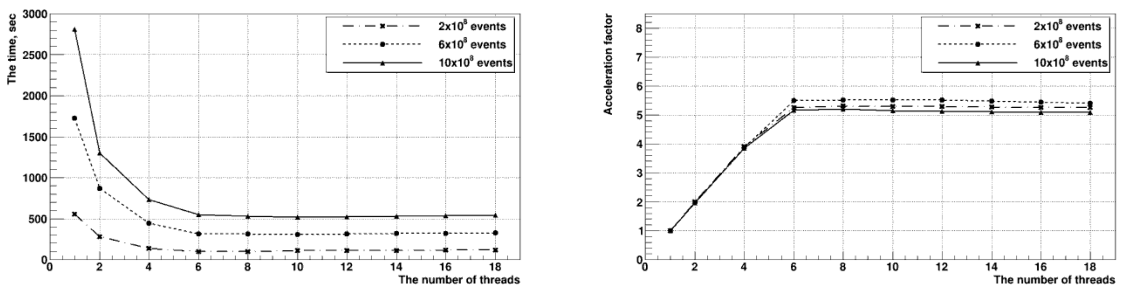

Figure 2. Dependence of the running time of the macro and the acceleration factor on the number of worker threads

From the results presented in the graphs, it can be seen that the processing time decreases with an increase in the number of created threads. When the number of worker threads exceeds the number of cores in use, the reduction in macro runtime stops. The acceleration factor is proportional to the number of available processor cores and does not depend on the number of generated events.

4. Conclusion

Currently, parallel computing is the main reserve for increasing the speed of calculations, therefore, the issues of parallelization of the ROOT package are of great importance. Modern architectures of computing systems make it possible to use various methods of parallelizing the processing of experimental data. The choice of the optimal method in each specific case significantly reduces the task execution time. Our research has shown that when processing data using PROOF, it is desirable to use the highest possible structuring of primary data. When working with the PROOF-Lite version, it is reasonable to set the number of workers equal to the number of reserved cores for that version. Writing to a file in parallel is easy to implement, and the results depend on the number of cores used.

5. Acknowledgement

The author is grateful to his colleagues for a fruitful discussion of the issues, as well as to the HybriLIT group of MLIT JINR for the resources and technical support provided.

References

[1] Brun R., Rademakers F. ROOT – An object oriented data analysis framework //Nuclear Instruments and Methods in Physics Research A. 1997. V. 389. P. 81-86.

[2] Adam G., Korenkov V., Podgainy D., Streltsova O., Strizh T., Zrelov P. HybriLIT – The main component of the MICC for heterogeneous computations at JINR// CEUR Workshop Proceedings (CEUR-WS.org). 2017. V.2023. P.351-356.

[3] Ballintijn M., Roland G., Brun R., Rademakers F. The PROOF distributed parallel analysis framework based on ROOT// Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.

[4] Solovjeva T., Soloviev A. Comparative study of the effectiveness of PROOF with other parallelization methods implemented in the ROOT software package// Computer Physics Communications. 2018. V.233. P.41-43.

[5] Amadio G., Bockelman B., Canal P., Piparo D., Tejedor E., Zhang Z. Increasing Parallelism in the ROOT I/O Subsystem // JoP: Conf. Series. 2018. V.1085. P.032014.

[6] Riley D., Jones C. Multi-threaded Output in CMS using ROOT// EPJ Web of Conferences. 2019. V. 214, P. 02016.

[7] Amadio G., Canal P., Guiraud E., Piparo D. Writing ROOT Data in Parallel with TBufferMerger// EPJ Web of Conferences. 2019. V. 214, P. 05037.