

Optimizing CMS Data Formats for Analysis

Summer Student Report

Peerut Boonchokchuay

Department of Computer Engineering

Chulalongkorn University, Thailand

Supervisor

David Lange

CMS, Lawrence Livermore National Laboratory, USA

I. Introduction

The introduction of the new CMS analysis format, MiniAOD, dated back to early 2014 during LHC long shutdown after Run-I. However, the adoption of this new data format, designed to be more compact yet preserving sufficient amount of information and precision, first data with miniAOD has reconstructed in June 2015. Intended to succeed AOD as the collaboration basis of analysis format during LHC Run-II, improvements on MiniAOD in various aspects has still been active as request for features persist.

This project aims at exploring methods to improve the MiniAOD performance in terms of time to read events from file and size on disk resulting from its creation process. Possible increases in time to write events to file, and application memory usage (resident memory size, or RSS) are taken into consideration.

In order to achieve the goal, the author attempted to improve the performance by investigating two methods: job configuration layer and implementation layer. Both approaches are benchmarked using the same metrics. The first approach involves investigating on parameter settings while the other approach entails altering class definitions in CMS framework.

II. Background Studies

CMSSW

CMSSW is the software framework by which CMS reconstruction, simulation and offline analyses are performed. Like other large-scale software, CMSSW is divided into interrelated modules each devoted to specific functionality. While framework is written in C++, it provides python language interface where users could perform physics analysis by submit a job configuration written in python.

PAT

Physics Analysis Toolkit (PAT) is a part of CMSSW that provides users with to access high-level data formats objects as well as algorithms by following PAT workflow. All PAT objects inherit from `pat::Candidate` which, in turn, inherits from `reco::Candidate`. The PAT workflow is the process of creating PAT objects from the CMS reconstruction, where they can be further used in analyses. The PAT workflow starts with extracting event data into `pat::Candidate`, selecting candidates based on user-defined condition and cleaning up possible duplication of selected candidates. It is compulsory to have the last step because one event data could be used to create multiple candidates.

ROOT

ROOT [1] is a software toolkit that provides a wide spectrum of features including data storage, data analysis and data visualization. Developed primarily at CERN, ROOT comes with C++ interpreter by which users are able to run not only compiled programs but also C++ scripts, called macros. Like other experiments, CMS builds analysis applications and persistently stores its data formats using ROOT.

MiniAOD

MiniAOD is one of CMS data formats used for offline analysis. Data stored in MiniAOD file comprises high-level objects that are stored in ROOT format using one branch per data member – some of which, including muon, electron, tau and jet, are PAT objects. MiniAOD is subset of AOD which is, in turn, a subset of RECO – the full event reconstruction output. With a smaller size than AOD, by the factor of ten, MiniAOD can be recreated quickly to improve the original physics object definitions. Figure 1 displays example objects stored in MiniAOD which can be retrieved by ‘edmDumpEventContent’ command.

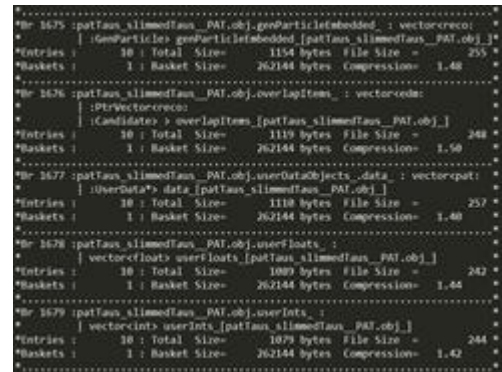


Figure 1 Example of MiniAOD Events Tree

MiniAOD Creation Process

Creating MiniAOD is a process usually carried out by expert group using a production system. First, AOD event dataset are taken as input into the InputSource module. Then the event data is used by a series of modules to create lighter objects by trimming unnecessary attributes and reducing its precision to acceptable level. The OutputSource module plays important role in persistency by writing data created in the last step, now kept in buffer, to MiniAOD file (Figure 2).



Figure 2 MiniAOD Creation Process

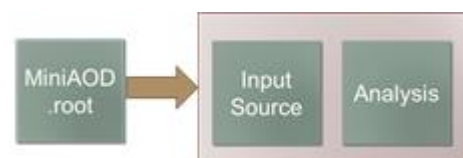


Figure 3 MiniAOD Reading Process

III. Tools

IgProf Profiler

IgProf [2] is a performance and memory profiler that provides feature to generate profiling report in the various formats from text file to structured relational database tables. The program also features report visualization through a simple web interface. The use of the profiler throughout the project was mainly in performance mode – to measure time to create MiniAOD and read back the containing data.

EDAnalyzer Framework

EDAnalyzer is one of the three framework modules that provide building blocks for CMS offline analyses. Unlike others framework modules, EDProducer and EDSelector, that both read and produce – or filter in EDSelector – event data as output, EDAnalyzer only reads the data and perform analysis without yielding output. This makes EDAnalyzer a better candidate for readback time measurement as the figure could be more significant instead of being shrouded by the longer writeback time. We can easily control the set of PAT objects that are read and perform a basic analysis with them to ensure that we have read them correctly at each step in the process.

IV. Methods

MiniAOD Creation Process

MiniAOD creation process in this project owed input datasets to the ttbar production at $\sqrt{s} = 13$ Tev which is located in the CMS Release Validation samples (CMS.ReVal). In the first step, we ran 'makeMiniAOD' jobs with the number of events ranging from 100 to 10,000 in order to confirm the linearity of execution time and size of output file. It is worth to note that performance measurements throughout the project were reported by in Tick unit, the IgProf's native unit to measure the frequency at which its sampling found each call stack. Having been aware the execution times scale linearly, we then selected 1,000 and 10,000 event samples as basis to which all following jobs in this project would be configured.

In an attempt to improve creation time of MiniAODs and their size on disk, parameters were selected from OutputSource module to study each impact on both optimization goals. The candidates included: basketSize, compressionLevel, compressionAlgorithm, eventAutoFlushCompressedSize, fastCloning and splitLevel. In the next step, we ran different jobs configurations where one parameter's value was adjusted at a time and, consequently, decided which settings contributed to better performance – less time to write event to file, more compact size, or both. The resulting MiniAOD files, different in size and structure depending on configuration, would be useful for the readback time analysis to be mentioned in the next section.

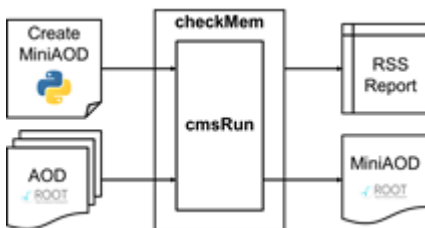


Figure 4 Application Memory-Measurement Workflow

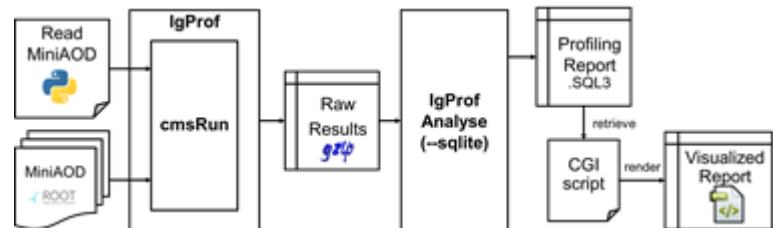


Figure 5 Time to Read Measurement Workflow

On the other hand, there were possibilities that changes in parameter settings caused side effects, namely, the growth of application memory usage as well as time to read event. We explored the first drawback using a python script that acted as a wrapper to 'makeMiniAOD' job while keeping track of memory occupied by the application every second. The highest figure printed out refers to RSS of a job which could then be treated as a metric (Figure 4).

MiniAOD Reading Process

To measure MiniAOD readback time entails a cycle of workflow similar to the creation process. Between the two processes, only the input data format and modules prescribed in job configuration – now MiniAOD and EDAnalyzer for reading PAT objects – were components that differed (Figure 5).

After that, we went further into figuring out which PAT objects in the MiniAOD account for the longest read time and the largest size to achieve the two improvement criteria in finer level. We investigated

time to read for each object: Muon, Electron, Photon, Tau, Jet, fat-Jet and MET with the aim of selecting objects of with the largest read time, inspecting each the class definition and its hierarchy and modifying some data member definitions suspected for being redundant or unnecessary.

In particular, data members of type double, occupying 32 bits memory, are of higher than sufficient precision for storing the data and can be substituted for 16-bit or even 8-bit float. Moreover, data members containing string might be replaced with enum, or possibly removed altogether, due to reducing lookup time and eliminating large amount redundant data.

Custom data structures whose purposes are already supported by standard library ought to be changed into standard way since they are more likely to cause problem than they could solve, i.e., abstaining from using two vectors to implement key-value map. Such changes, however, involves not only class definition but also the implementation which is far from scope of this report, but it is by all means worth to mention for future development.

Although each physics object was inspected in full hierarchy, only part of classes having 'pat' namespace did the project concentrate on. We, therefore, set data members (attributes) that contain strings to be transient, meaning those attributes would be absent from the created object, thus not being stored in persistent storage. A list of attributes is displayed in Figure 6. To sum up, eight attributes became transient from Muon, nine for Electron and Tau, and ten for Jet.

Class	Attribute	Class	Attribute
pat::Lepton	isoDeposits_	pat::PATObject	efficiencyNames_
pat::Muon	-		overlapLabels_
pat::Electron	electronIDs_		userDataLabels_
pat::Tau	tauIDs_		userFloatLabels_
pat::Jet	subjetLabels_		userIntLabels_
	pairDiscriVector_		userCandLabels_
	tagInfoLabels_		kinResolutionLabels_

Figure 6 Lists of Attributes to Become Transient for Our Test

Altering the class definition required part of CMSSW to be recompiled before both creation and reading process were repeated once more in order to produce MiniAOD files with structure modified in accordance with previous changes in class definition. In the last step, the readback time and size of the modified MiniAOD was benchmarked against the original one.

V. Results and Discussion

Parameter Settings

Among the parameters selected from OutputSource module, basketSize – size of the buffer – had a noticeable effect on both MiniAOD size and readback time. Plotted in base-2 logarithmic scale, Figure 7 indicates that size of the output MiniAOD generated by 'makeMiniAOD' job declined with the increase in basketSize. The readback time followed the same trend as in Figure 8. Despite the stable trend after 262,144 bytes, running the creation process with basketSize larger than such value was reported to have a drastic rise in application memory (Figure 9). There is a possibility that by adjusting basketSize from the default 16384 bytes to 262144 byte as illustrated in red and green lines, respectively, we can improve MiniAOD file size by 8% and read back time by 9% while not significantly increasing the RSS.

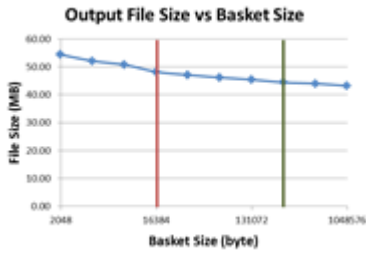


Figure 7 Output File Size vs Basket Size

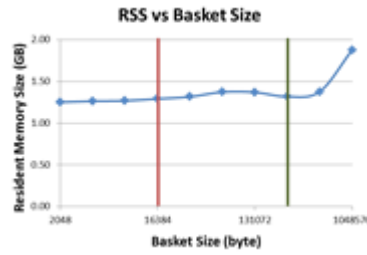


Figure 8 RSS vs Basket Size

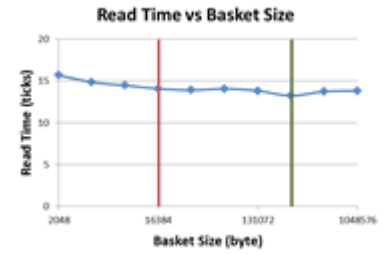


Figure 9 Read Time vs Basket Size

Compression algorithm and level could also affect the read and write performance. There are two compression algorithms provided by the framework, LZMA and ZLIB. Each offers a range of compression level from 0, where no compression is performed, to 9, the most intense level. In comparison, LZMA has more complexity, hence taking longer processing time yet producing output of smaller size. Figure 10 compares writeback time for LZMA application growing dramatically as compression level increases with that of ZLIB application which consumes almost the same amount of time. In addition to that, compression level has trivial effect on readback time for both algorithms; however, LZMA read time could take up to 40% longer than that of ZLIB (Figure 11).

Investigation on others parameters shows that eventAutoFlushCompressedSize and splitLevel were by default set close to the optimal value while whether fastCloning was enabled or not had insignificant effect on both performance and size.

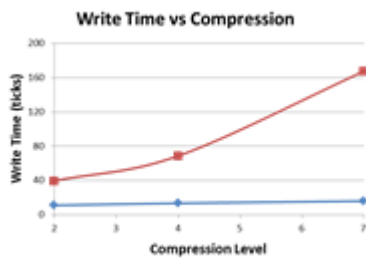


Figure 10 Write Time vs Compression

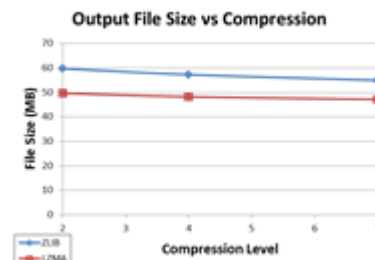


Figure 11 Output File Size vs Compression

Class Definitions

Figure 12 exhibits that Jet accounts for 29.17% of the total size followed by Muon, Electron and Tau with the percentage of 28.74, 19.15 and 10.21, respectively. We then chose these four objects for further inspection into their attributes. It is important to note that sizes of the seven objects combined made up for 19.2% of the total MiniAOD file, spanning 963 out of 2055 branches.

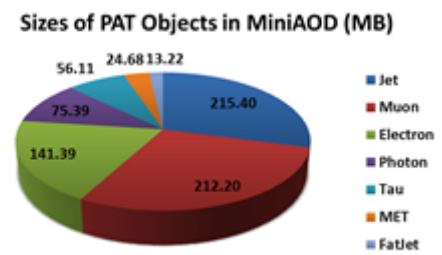


Figure 12 Sizes of PAT Objects in MiniAOD

After marking as transient the data members containing strings of the selected classes: Muon, Electron, Tau, Jet as well as part of their hierarchy having pat namespace, Events tree of the new MiniAOD file contains 1975 branches, having 80 less branches than the original tree and the size reduced by 1.7%.

MiniAOD Input File	Time to read event (ticks)									
	Seven objects	Selected objects	Muon	Electron	Photon	Tau	Jet	Fat Jet	MET	No object
Original	27.03	23.19	9.17	4.22	2.95	3.94	9.32	1.82	2.21	1.14
Modified	22.67	18.78	8.97	3.83	2.84	2.86	6.39	1.69	2.12	1.06

Figure 13 Comparison of Time to Read Event by Object between Different Structures of MiniAOD

The read time broken down by object of both version of MiniAOD is shown in Figure 13. Comparing with the original structure, reading event data took 31% less time for Jet, 27% for Tau, 9% for Electron, and 2% for Muon while readback time for all seven objects and only selected object were improved by 16% and 19% respectively. The reason for Muon to have the least improvement could be that none of its data members contains string type. Therefore, no attribute became transient except for seven of those obtained by inheritance.

VI. Concluding Remarks

The strategies to improve CMS MiniAOD performance involved setting OutputSource module parameters and modifying class definition of PAT objects. We found that some parameter settings could result in a significant performance gain. BasketSize tuning could improve readback time as well as MiniAOD size while having a relatively small increase in RSS. Other parameters were explored, but some were already set close to their optimal point. Improvement in class definition level was approached as well. Time to read event was measured for each high-level physics object and attributes of Muon, Electron, Tau and Jet were removed from the MiniAOD. The improvement in read performance and reduced data size were observed to be most significant in Jet and Tau. With the result from both approaches, users could benefit as much from faster processing time and smaller data size as experts could do from faster creation time and more robust product. Having investigated the worst case scenario for string type in class definition level optimization, at the same time we have opened a certain aspects of implementation level optimization.

VII. Acknowledgement

David Lange provides reading materials, script for measuring application memory usage and generous supports throughout the project. Norraphat Srimanobhas provides in-depth advices on PAT and working environment which largely alleviate productivity.

VIII. References

- [1] Rene Brun and Fons Rademakers,
ROOT - An Object Oriented Data Analysis Framework,
Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389
(1997) 81-86. See also <http://root.cern.ch/>
- [2] Eulisse, Giulio and Tuura, Lassi. 2013. *IgProf, The Ignominous Profiler*.
<http://igprof.org/>
(Accessed: 2015-09-09)