Optimizing CMS Data Formats for Analysis

Peerut Boonchokchuay

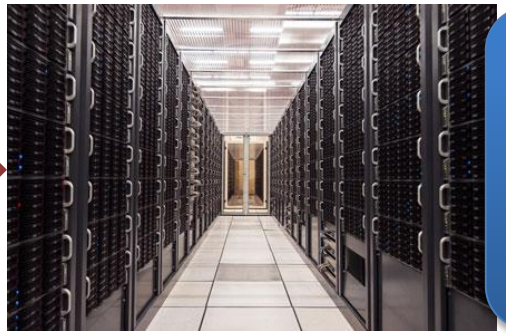
August 11th, 2015

Outline

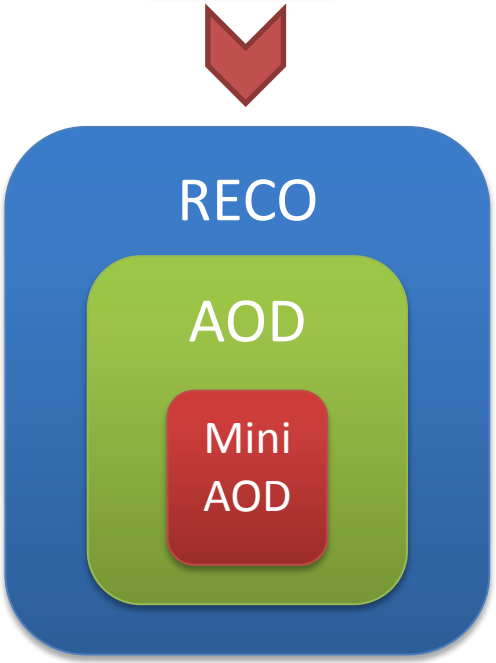
- Introduction to MiniAOD
- Objectives for Project
- Method Developed
- Example Results
- Conclusion and Next Steps

CMS MiniAOD Data Format

RAW



RECO



- First introduced in 2014, MiniAOD is new CMS data format for analysis in LHC Run2
  - Data stored in MiniAOD file comprises of high-level object, i.e., tracks, jets, electrons
  - MiniAOD is a subset of AOD which is, in turn, a subset of RECO – output of Tier-0 event reconstruction
    - MiniAOD is 10x smaller than AOD
  - One advantage is that the miniAOD can be recreated quickly to improve the original physics object definitions using the latest algorithms

Left: https://cms-docdb.cern.ch/cgi-bin/PublicDocDB/RetrieveFile?docid=5936&filename=CMSbookS.pdf
Right: https://public.web.cern.ch/public/features-archive/1204-1206.html

Objectives

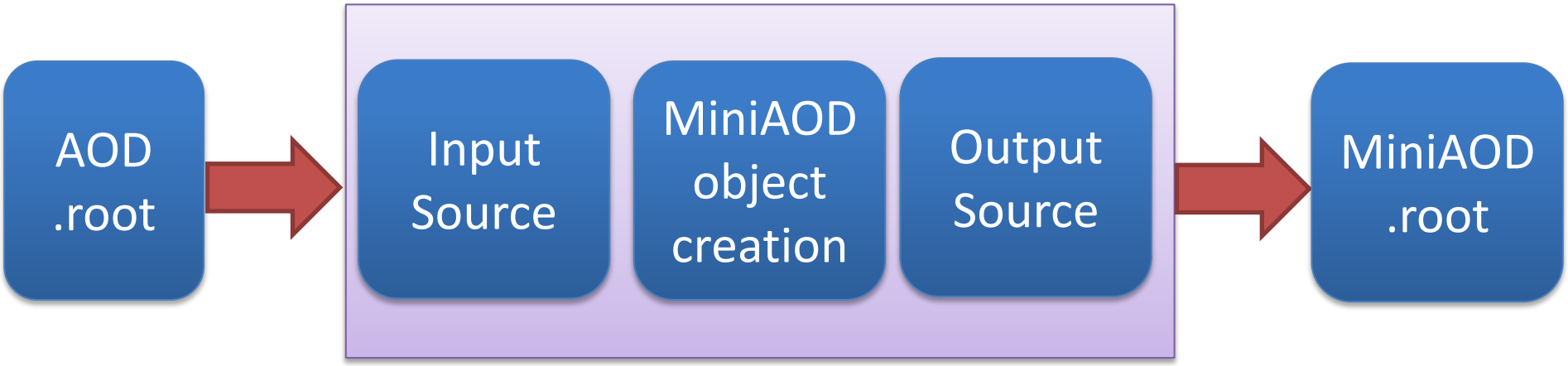
- As with any analysis formats, users can benefit from faster processing times and smaller data sizes (to fit more data on your laptop!)

- To investigate ways to improve the MiniAOD performance in terms of:
  1. Time to read events from file
  2. Size of output file

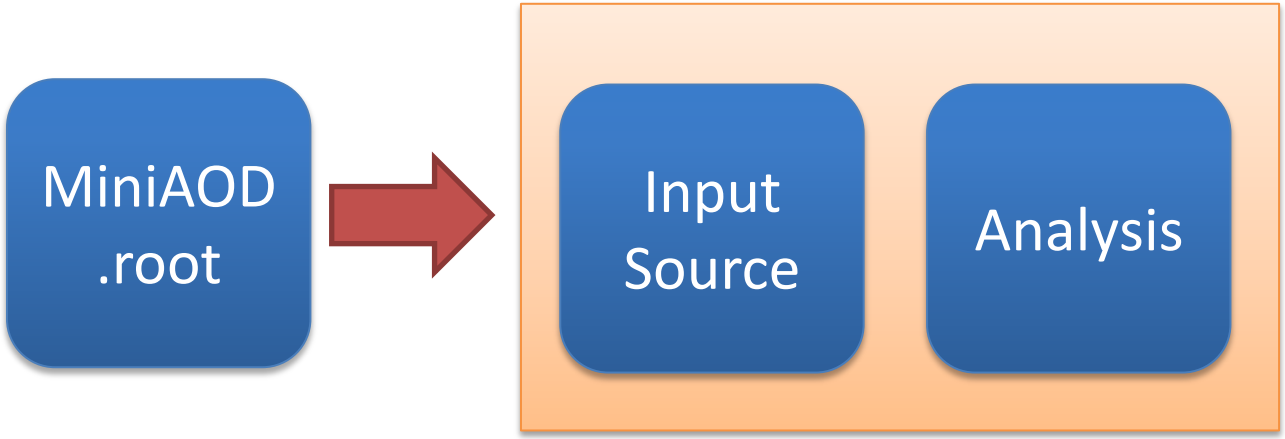
  while withstanding:
  1. Job memory usage (RSS of CMSSW application)
  2. Time to write events to file

CMS MiniAOD data flow

Step 1: Create MiniAOD



Step 2: Analyze MiniAOD



MiniAOD
modules
and
parameters
are based on
ROOT

Method

Study MiniAOD Creation Process

- Execution time
  - Write time
  - Compression time
- Output file size
- Memory usage

Study MiniAOD Reading Process

- Execution time
  - Read time
  - Decompression time

Adjust Parameter Settings

- Number of events
- Basket Size
- Compression Algorithm
- Compression Level
- .....

Measure Performance

- Tools
  - igProf
  - RSS monitoring
- In terms of:
  - CPU time
  - memory usage

Example of igProf Results

Chulalongkorn University
จุฬาลงกรณ์มหาวิทยาลัย
Pillar of the Kingdom

Counter: PERF_TICKS, first 1000 entries

Sorted by cumulative cost

(Sort by self cost)

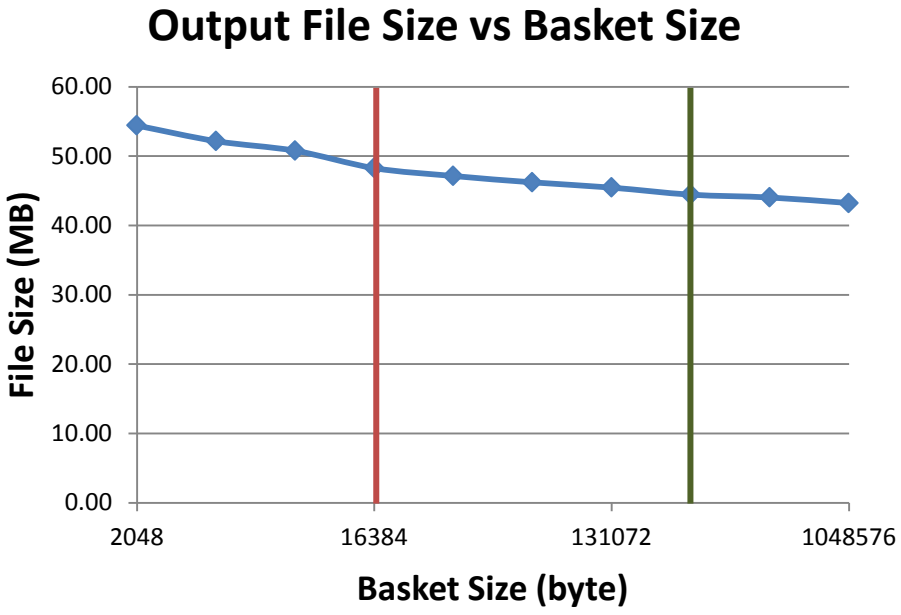
Rank	Total %	Cumulative	Symbol name
1	100.00	212.30	<spontaneous>
3	99.99	212.29	__libc_start_main
2	99.99	212.29	_start
4	99.33	210.89	main
5	99.28	210.78	main::{lambda()#1}::operator()() const
6	77.01	163.50	edm::EventProcessor::runToCompletion()
10	69.24	147.01	edm::EventProcessor::readAndProcessEvent()
9	69.24	147.01	statemachine::HandleEvent::readAndProcessEvent()
8	69.24	147.01	statemachine::HandleEvent::HandleEvent(boost::statechart::state<statemachine::HandleEvent, statemachine::Ha
7	69.24	147.01	boost::statechart::state<statemachine::HandleEvent, statemachine::HandleLumis, boost::mpl::list<mpl_::na, m
13	69.24	147.00	edm::EventProcessor::processEventsForStreamAsync(unsigned int, std::atomic<bool>*)
12	69.24	147.00	edm::StreamProcessingTask::execute()
11	69.24	147.00	tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_for_all(tbb::task&, tbb::t
14	69.02	146.54	edm::EventProcessor::processEvent(unsigned int)
17	68.47	145.37	void edm::StreamSchedule::processOneEvent<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType
16	68.47	145.37	decltype ({parm#1}()) edm::convertException::wrap<void edm::StreamSchedule::processOneEvent<edm::Occurrence
15	68.47	145.37	void edm::StreamSchedule::processOneEvent<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType
19	68.46	145.35	decltype ({parm#1}()) edm::convertException::wrap<bool edm::Worker::doWork<edm::OccurrenceTraits<edm::Event
18	68.46	145.35	bool edm::Worker::doWork<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >(edm::Occurr
20	68.45	145.33	void edm::Path::processOneOccurrence<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >
21	68.45	145.31	decltype ({parm#1}()) edm::convertException::wrap<void edm::Path::processOneOccurrence<edm::OccurrenceTrait

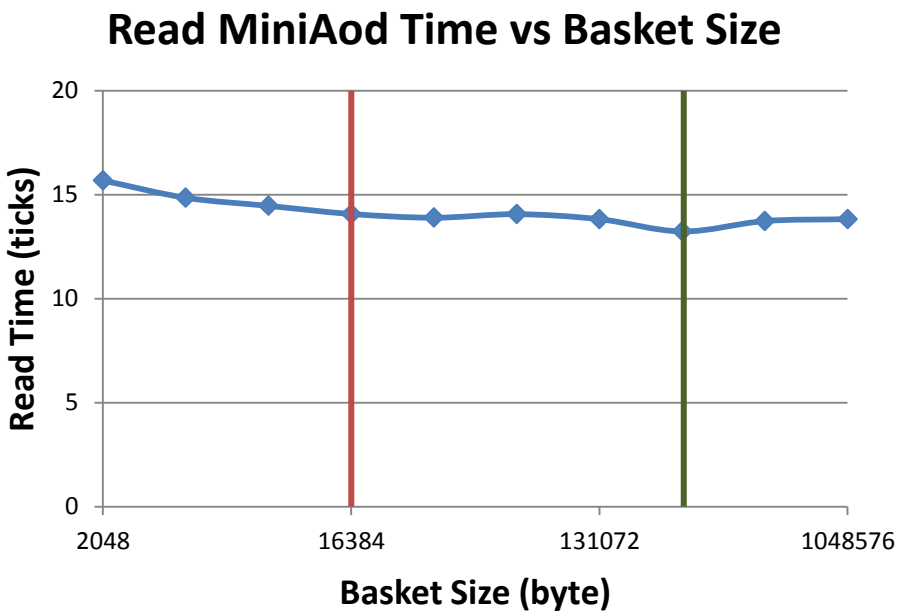
Example of igProf Results

85	8.25	17.91	boost::statechart::simple_state<statemachine::FirstLumi, statemachine::HandleLumis, boost::mpl::list<
86	7.57	16.43	TMVA::Node::ReadXML(void*, unsigned int)'4
87	6.96	15.10	edm::InputProductHolder::resolveProduct (edm::ProductHolderBase::ResolveStatus&, bool, edm::ModuleCal
88	6.95	15.08	edm::reftobase::IndirectVectorHolder<reco::Candidate>::at(unsigned long) const
89	6.92	15.02	edm::EventPrincipal::readFromSource (edm::ProductHolderBase const&, edm::ModuleCallingContext const*
90	6.85	14.87	TMVA::Node::ReadXML(void*, unsigned int)'5
91	6.80	14.75	edm::DelayedReader::getProduct(edm::BranchKey const&, edm::EDProductGetter const*)
92	6.77	14.70	edm::RootDelayedReader::getProduct (edm::BranchKey const&, edm::EDProductGetter const*) const
93	6.77	14.69	@{liblzma.so.5.0.3+8686b}
94	6.46	14.03	void TMVA::Tools::ReadAttr<float>(void*, char const*, float&)
95	6.28		
96	6.24		
97	6.18		
98	5.99		
99	5.50		
100	5.45		
101	4.97		
102	4.88		
108	4.86		
107	4.86		
106	4.86		
105	4.86	10.54	boost::statechart::detail::reaction_result boost::statechart::simple_state<statemachine::HandleFiles,
104	4.86	10.54	boost::statechart::simple_state<statemachine::HandleRuns, statemachine::HandleFiles, statemachine::Ne
103	4.86	10.54	boost::statechart::simple_state<statemachine::HandleLumis, statemachine::HandleRuns, statemachine::Fi

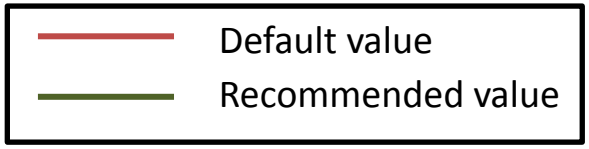
Rank	% total	Counts: to / from this	Counts: Total	Paths: Including child / parent	Paths: Total	Symbol name
	7.09	14.97	15.06	27	30	edm::InputProductHolder::resolveProduct (edm::Pr
[89]	7.09	0.05	14.93	27	27	edm::EventPrincipal::readFromSource_(edm::Produc
	6.98	14.73	14.74	25	26	edm::DelayedReader::getProduct(edm::BranchKey co
	0.04	0.08	0.09	3	4	edm::Principal::checkUniquenessAndType(edm::Wrap
	0.02	0.04	0.04	3	3	edm::BranchKey::BranchKey(edm::BranchDescriptio
	0.02	0.04	0.04	3	3	std:: Sp_counted_deleter<decltype(nullptr), edm
	0.01	0.02	0.02	2	2	edm::InputProductHolder::putProduct (std::unique
	0.01	0.01	5.54	1	494	__init

Example Results: MiniAOD Reading Process

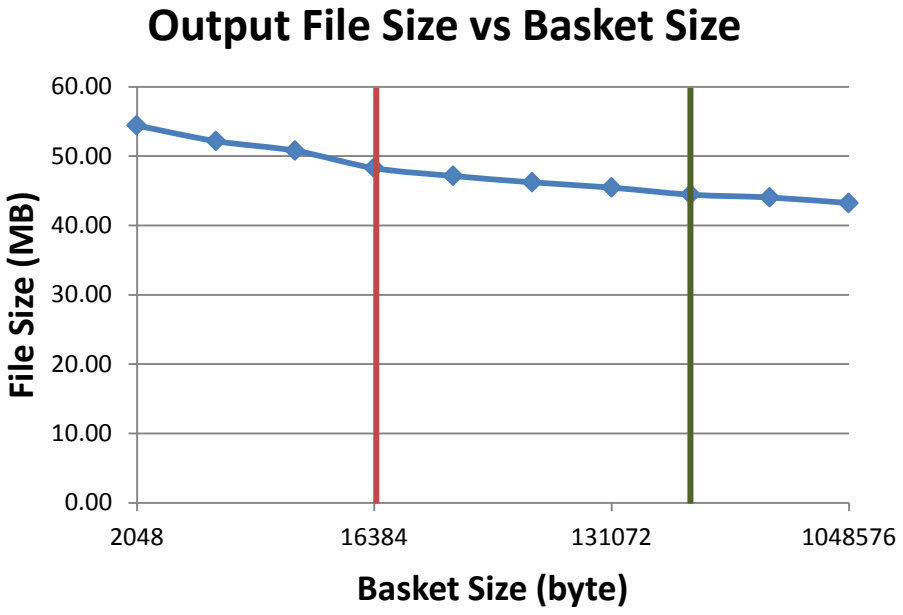


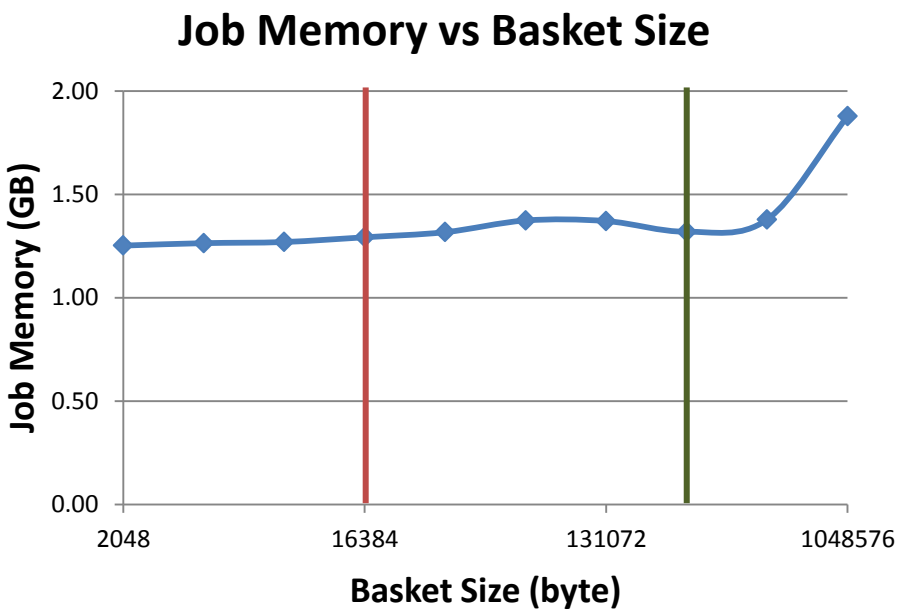


- By adjusting basket size from the default 16384 byte to 262144 byte, we can improve MiniAOD file size by 8% and read back time by 9%

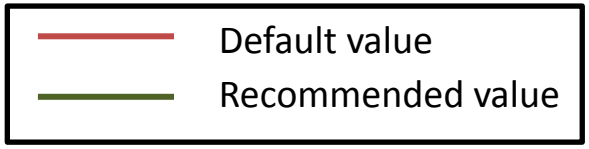


Example Results: MiniAOD Creation Process

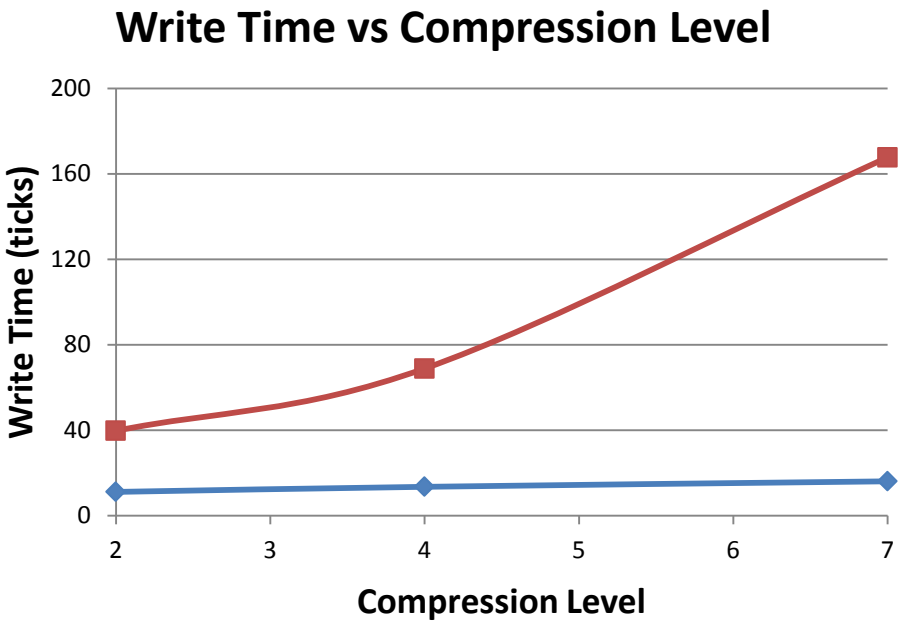


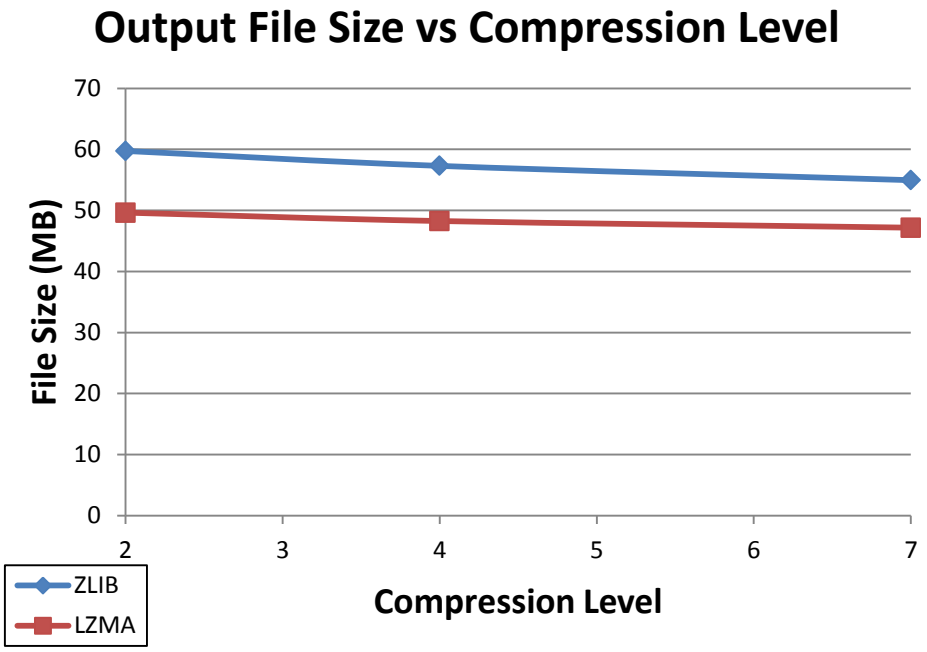


- Increasing the basket size to 262144 bytes also does not significantly increase the total RSS of the application used to create the CMS miniAOD



Example Results: Performance vs File Compression Algorithm





- Of two algorithms, LZMA and ZLIB, the first is more complex, hence taking longer processing time, but yields smaller output size.
- Write time for LZMA rises drastically as compression level increases whereas ZLIB consumes almost the same amount of time
- Compression level has trivial effect on read time for both algorithms, however, LZMA read time could take up for 40% longer than that of ZLIB

Conclusion

- We investigated ways to improve CMS MiniAOD performance.
- We found that optimizing some parameter settings could result in significant performance gain.
- Basket size tuning could improve read back time as well as output file size of MiniAOD while having a relatively small increase in job memory
- Other parameters we investigated, including the compression algorithm used were already set close to their optimal point
- Next step: Investigate ways to improve performance by changing the miniAOD object definitions

Chulalongkorn University
จุฬาลงกรณ์มหาวิทยาลัย
Pillar of the Kingdom

Thank You