Writing ROOT Data in Parallel with TBufferMerger

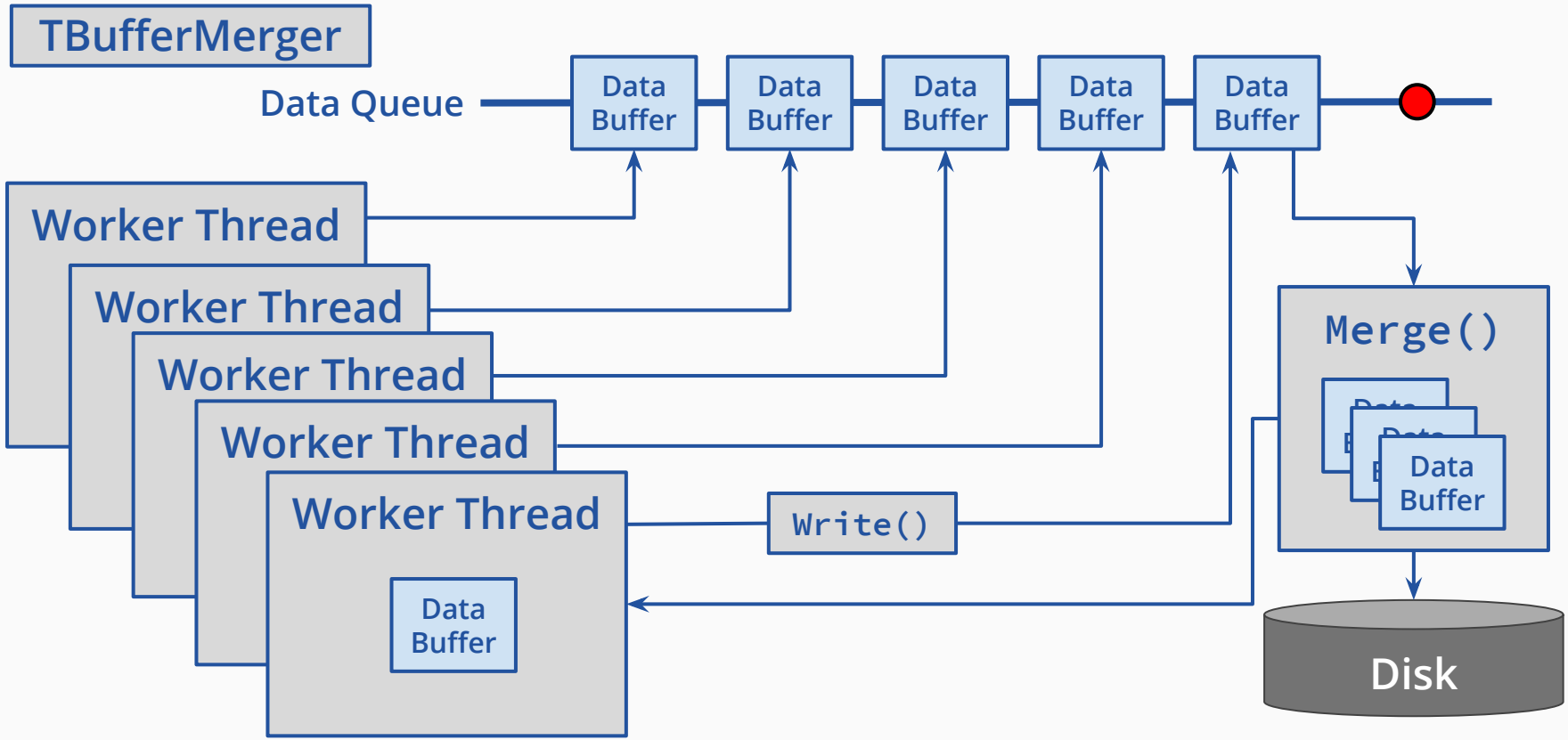
G. Amadio
for the ROOT Team

ROOT
Data Analysis Framework
https://root.cern

TBufferMerger

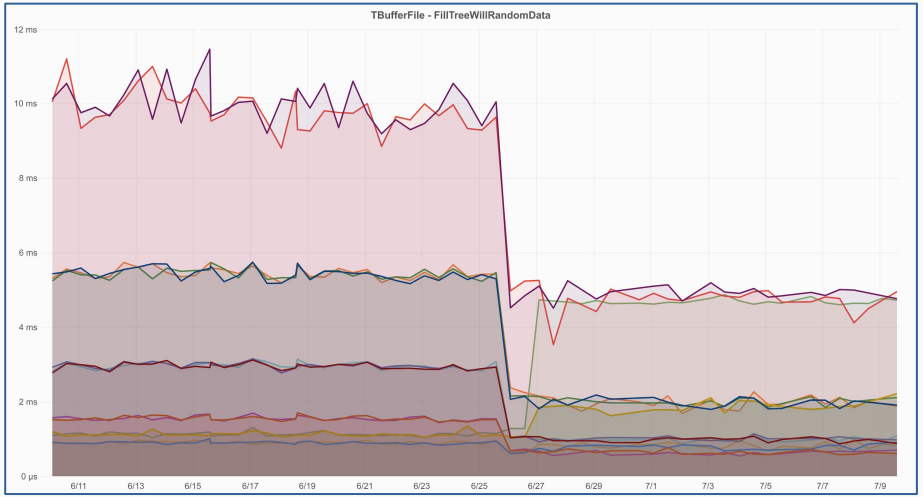
- First available in ROOT 6.10
- Parallel data processing while producing a single output file
- Use cases
  - Facilitate parallel dataset creation (MC generation, RECO, etc)
  - Save result of **`RDataFrame`** analysis in parallel with snapshot action
    (see talks by **E. Guiraud** and **D. Piparo**, and BoF session on Wednesday afternoon)
  - Not meant for merging existing files, for that there is already **hadd**
- Main ideas
  - Leverage the existing classes **`TMemFile`** and **`TFileMerger`**
  - Create an interface similar to single-threaded case to ease conversion

TBufferMerger



Advantages of new implementation

- Agnostic about user's model of parallelism
  - Usable both with threads as well as tasks
  - Integrates better with experiment framework
  - No oversubscription due to compression work in merging thread
- Less lock contention → better performance
- Visible performance improvement in continuous monitoring (see poster about it by **O. Shadura**)



Programming Model

Sequential usage of TFile

```
void Fill(TTree &tree, int init, int count)
{
   int n = 0;

   tree->Branch("n", &n, "n/I");

   for (int i = 0; i < count; ++i) {
      n = init + i;
      tree.Fill();
   }
}

int WriteTree(size_t nEntries)
{

   TFile f("myfile.root");
   TTree t("mytree","mytree");

   Fill(&t, 0, nEntries);

   t.Write();

   return 0;
}
```

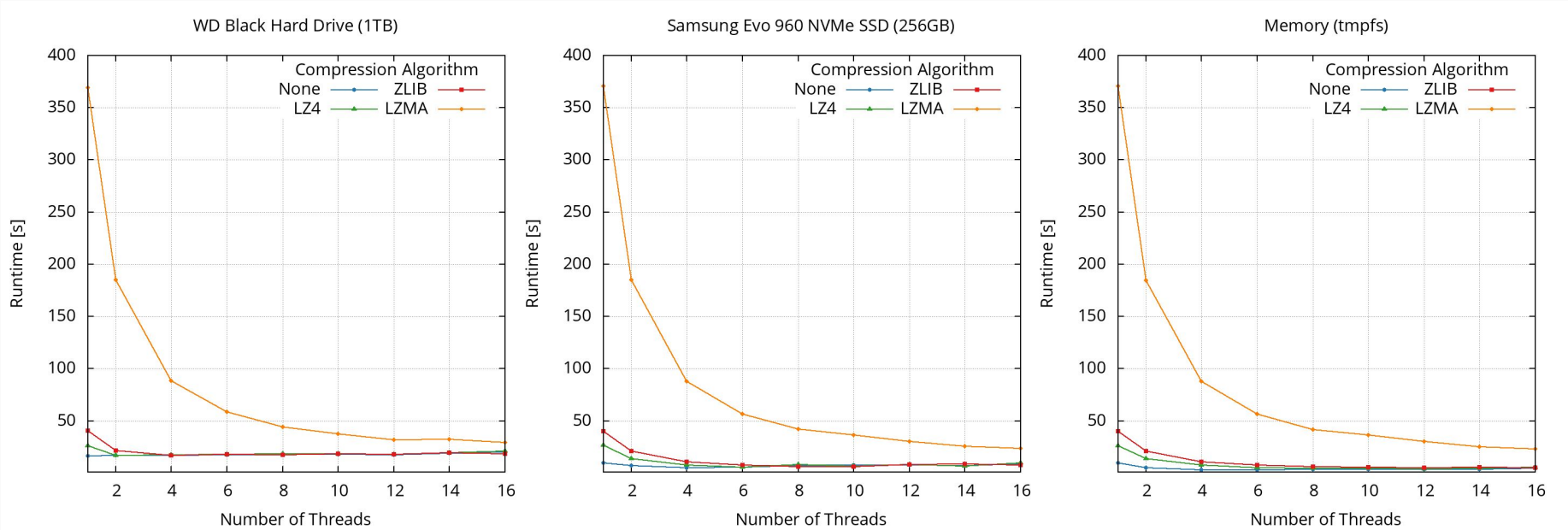
Parallel usage of TFile with TBufferMerger

```
void Fill(TTree *t, int init, int count); // same as on the left

int WriteTree(size_t nEntries, size_t nWorkers)
{
   size_t nEntriesPerWorker = nEntries/nWorkers;

   ROOT::EnableThreadSafety();
   ROOT::Experimental::TBufferMerger  merger("myfile.root");

   std::vector<std::thread> workers;

   auto workItem = [&](int i) {
         auto f = merger.GetFile();
         TTree t("mytree", "mytree");
         Fill(t, i * nEntriesPerWorker, nEntriesPerWorker);
         f->Write(); // Send remaining content over the wire
      };

   for (size_t i = 0; i < nWorkers; ++i)
      workers.emplace_back(workItem, i);

   for (auto&& worker : workers) worker.join();

   return 0;
}
```

TBufferMerger Single Branch Benchmark

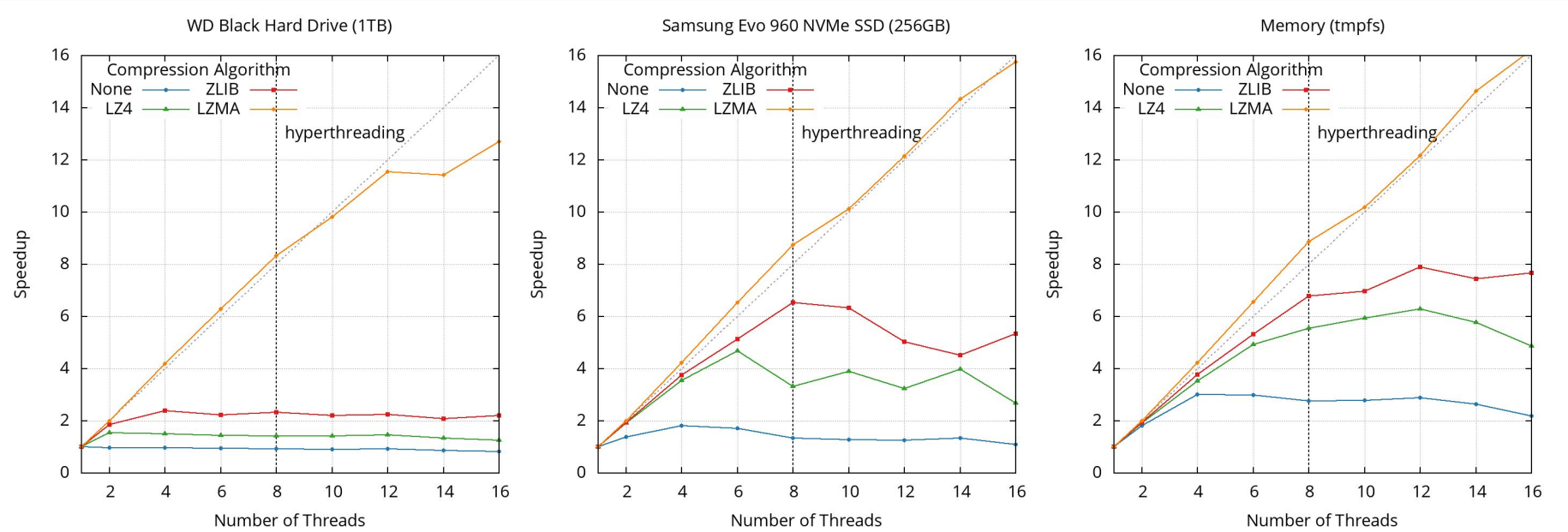
- Create ~1GB of **simple** data and write out to different media using different compression algorithms
- Measured time to flush disk cache is negligible compared to runtime
- Synthetic benchmark that exacerbates the role of I/O by doing light amount of work (generating a random number)
- Test environment
  - Intel® Core™ i7-7820X Processor (8 cores, 11M Cache, up to 4.30 GHz)
  - Write out data to HDD, NVMe SSD, DRAM
  - Compare compression algorithms: LZ4, ZLIB, LZMA, no compression
  - GCC 8.1.0, C++17, -O3 -march=native (skylake-avx512), release build

Single Branch Benchmark: Runtime



All figures using ROOT master branch

Single Branch Benchmark: Speedup



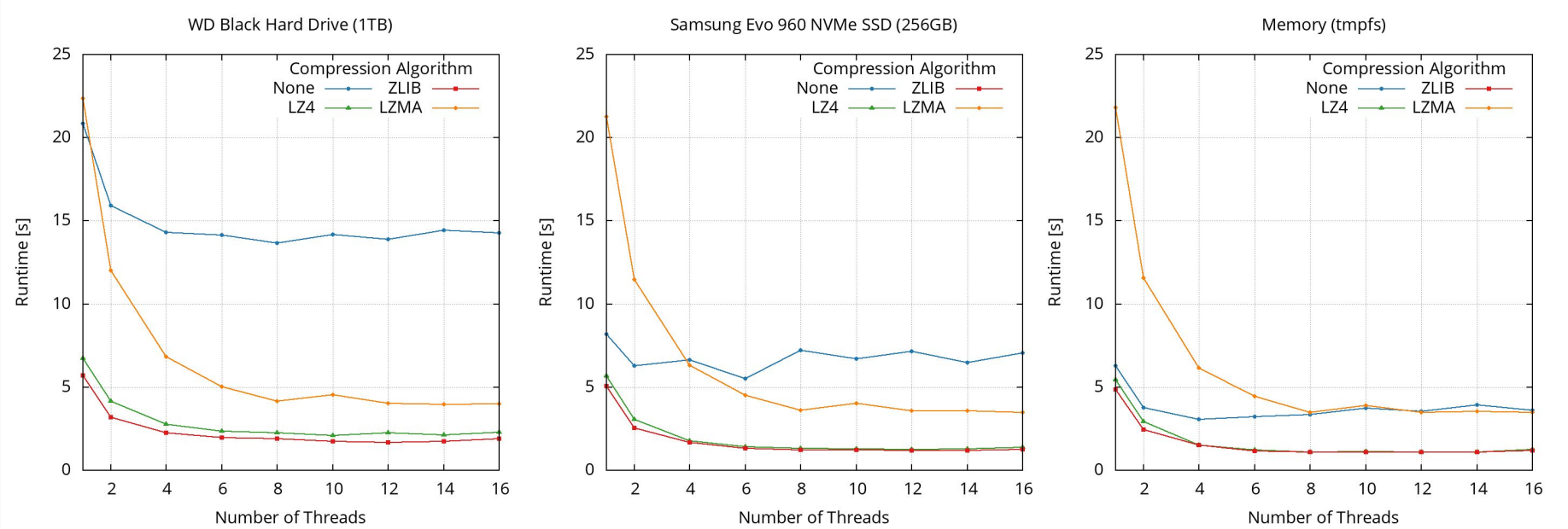
All figures using ROOT master branch

TBufferMerger Multi Branch Benchmark

- Create 1GB of **complex** data and write out to different media using different compression algorithms
- Synthetic benchmark to investigate what changes with added data complexity vs previous benchmark, IMT disabled but speedups are similar
- 1 branch = std::vector<Event> (3x Vector3D, 3x double, 3x int)
- Data compresses better, so uncompressed is writing more output
- Test environment
  - Intel® Core™ i7-7820X Processor (8 cores, 11M Cache, up to 4.30 GHz)
  - Write out data to HDD, NVMe SSD, DRAM
  - Compare compression algorithms: LZ4, ZLIB, LZMA, no compression
  - GCC 8.1.0, C++17, -O3 -march=native (skylake-avx512), release build

Multi Branch Benchmark: Runtime

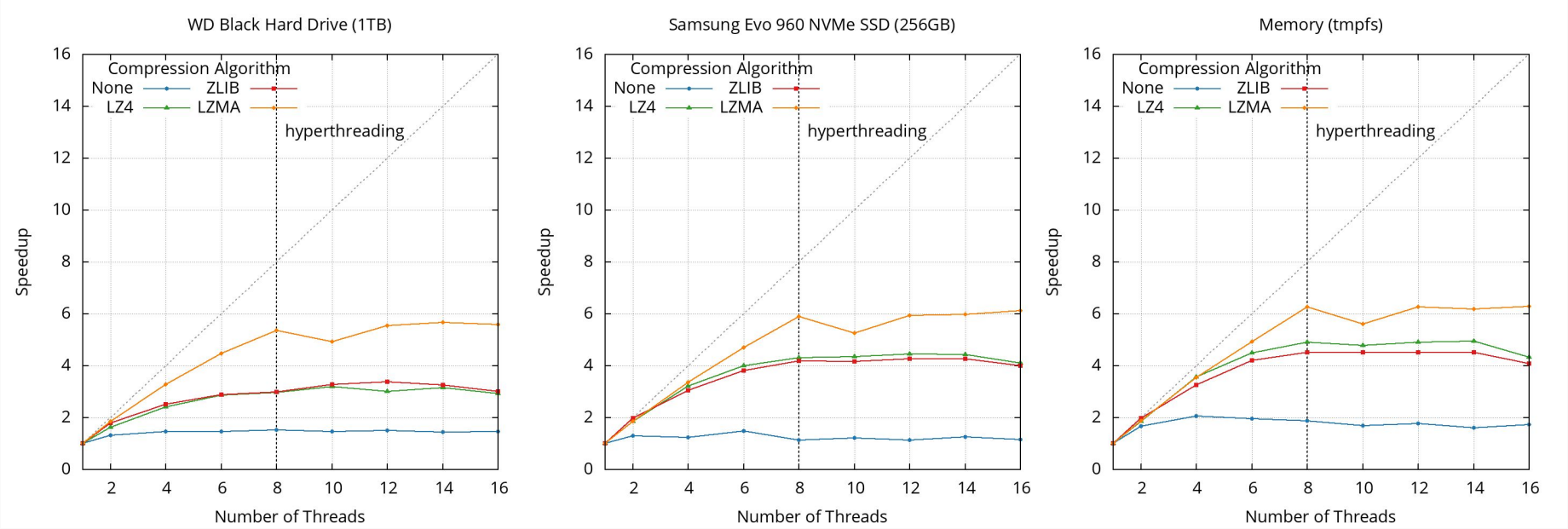
- Test creates 10 branches, each with a vector of 10 Events



All figures using ROOT master branch

Multi Branch Benchmark: Speedup

- Test creates 10 branches, each with a vector of 10 Events



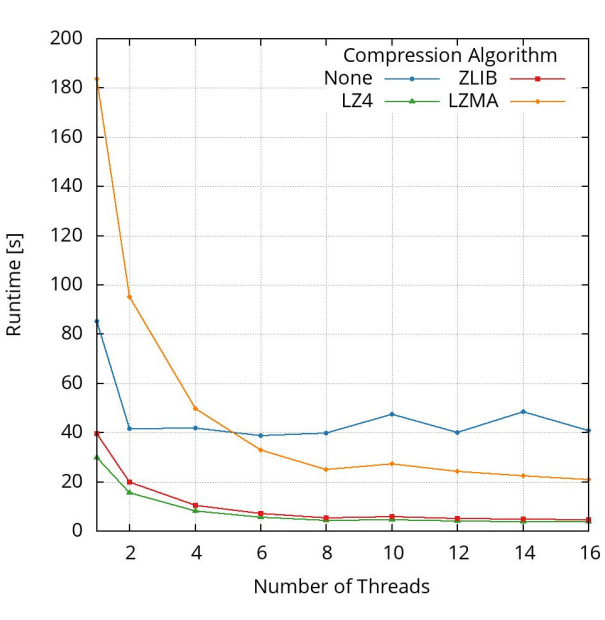
All figures using ROOT master branch

Effect of Number of Branches for Fixed Data Size

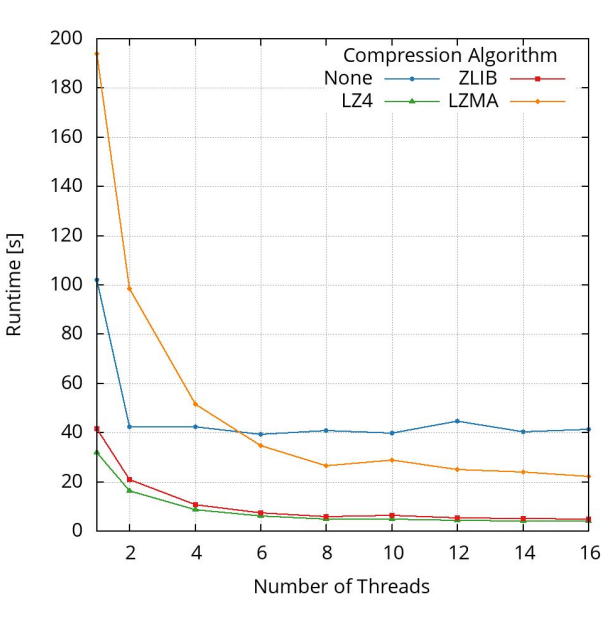
- Complex data test with **10GB** of uncompressed data
- Change flush/save strategy (per thread)
  - Auto save every 10%, Auto-flush every 1%
- Implicit multi-threading **disabled**
- Vary number of branches while keeping data size fixed
- Output to NVMe SSD only

Effect of Number of Branches for Fixed Data Size

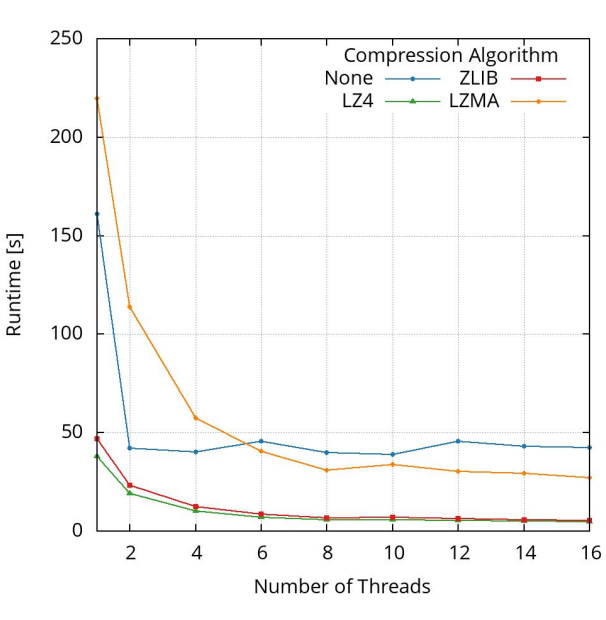
1 branch, 100 events each



2 branches, 50 events each

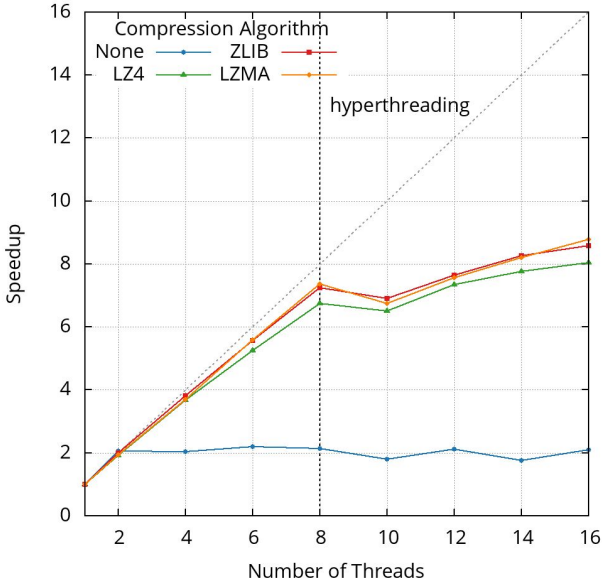


5 branches, 20 events each

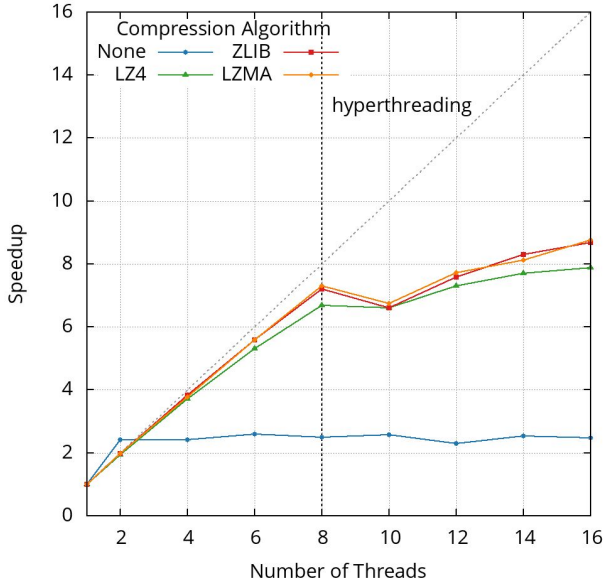


Effect of Number of Branches for Fixed Data Size

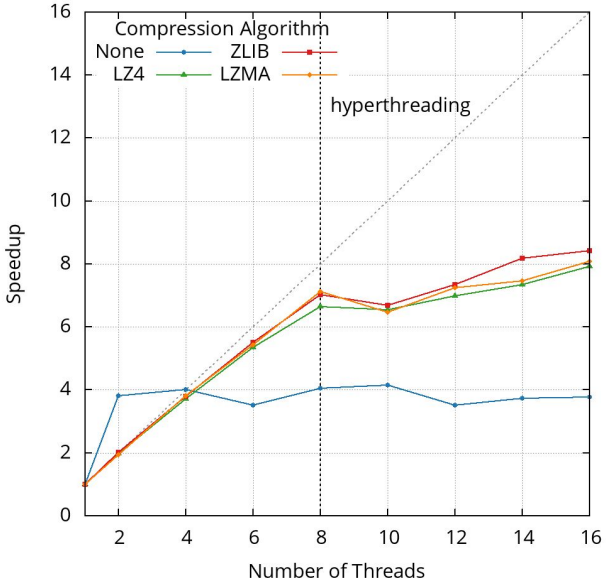
1 branch, 100 events each



2 branches, 50 events each

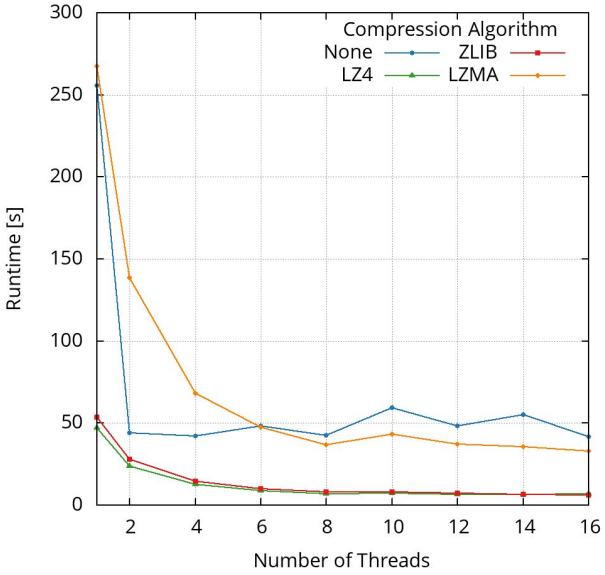


5 branches, 20 events each

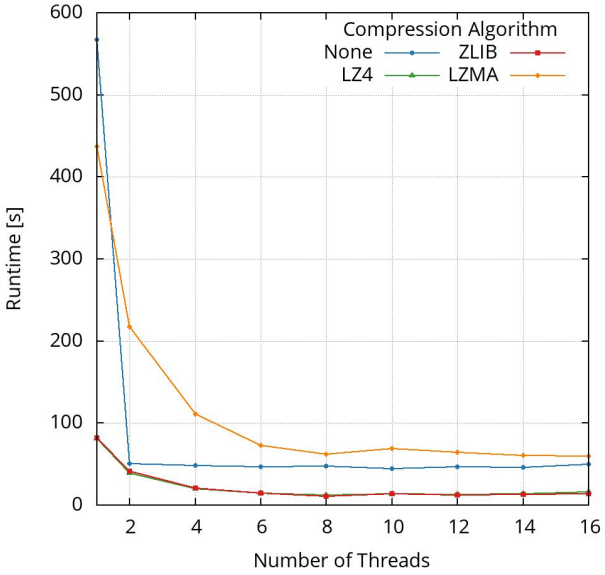


Effect of Number of Branches for Fixed Data Size

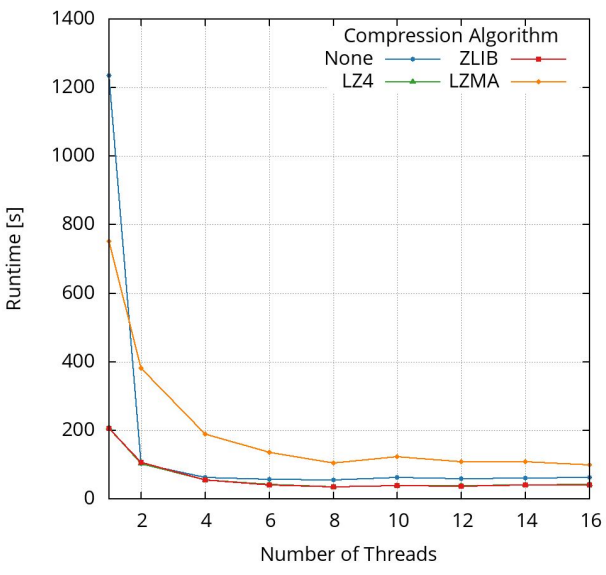
10 branches, 10 events each



25 branches, 4 events each

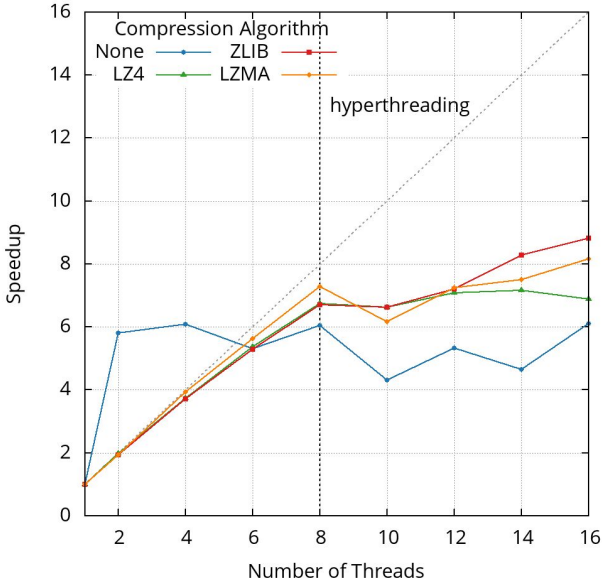


50 branches, 2 events each

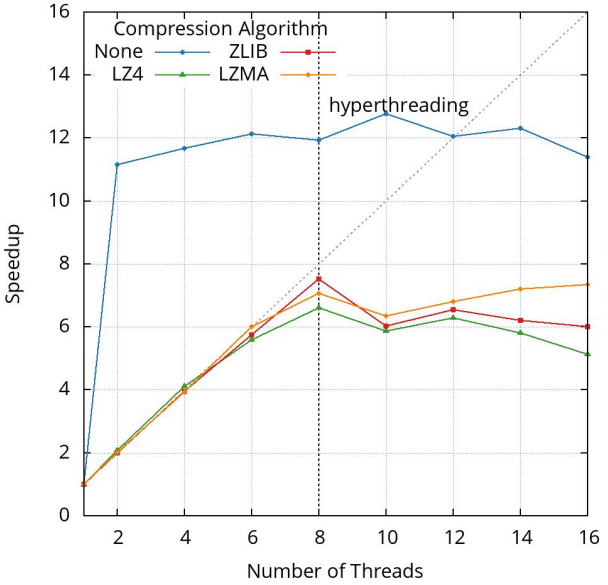


Effect of Number of Branches for Fixed Data Size

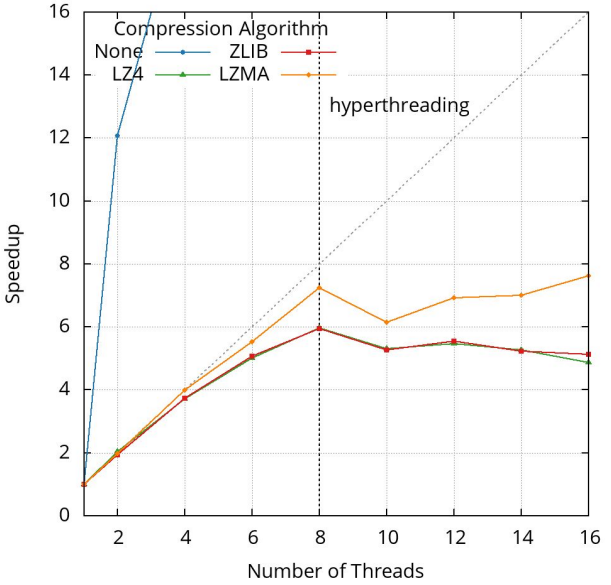
10 branches, 10 events each



25 branches, 4 events each

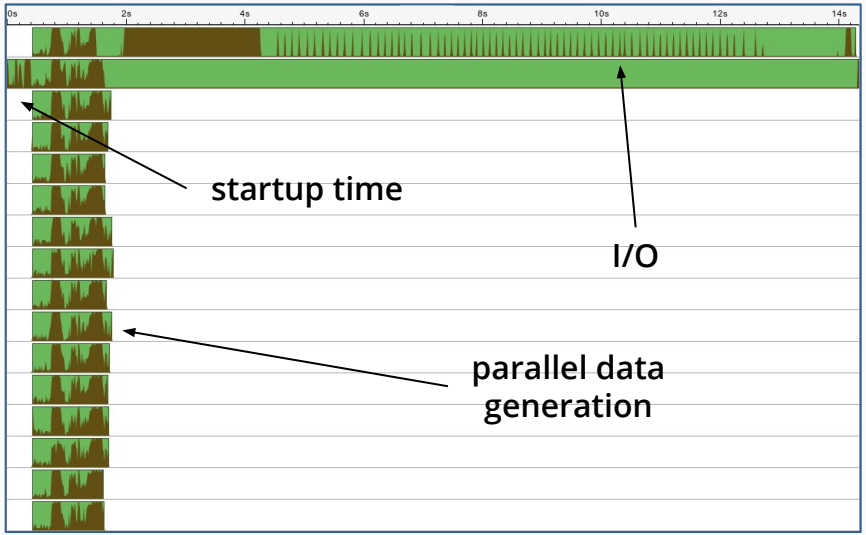


50 branches, 2 events each

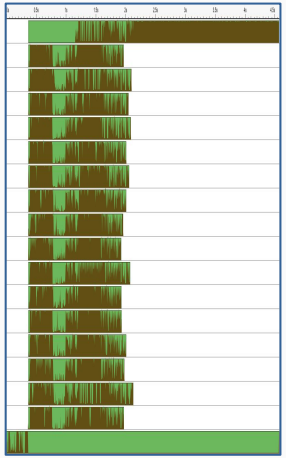


TBufferMerger Performance Analysis

- ROOT can saturate any media type with only a few threads if not CPU bound
- Scalability can still be an issue due to many guards on ROOT's global lock
- Need a solution for backpressure to avoid excessive memory consumption



output to hard disk

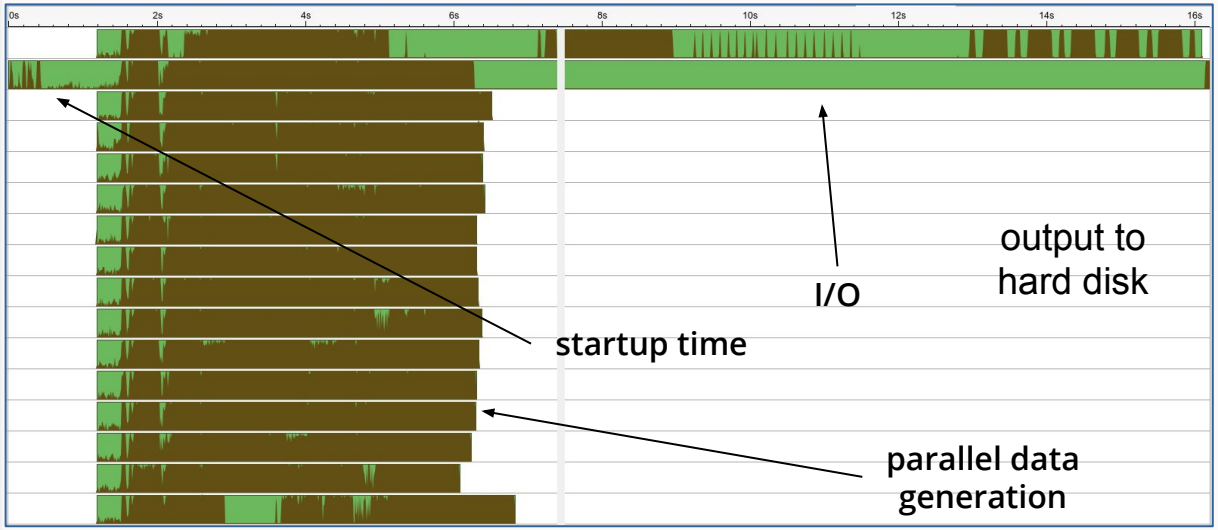


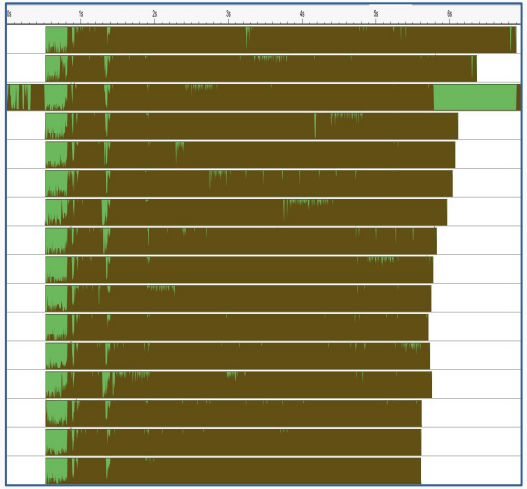
output to memory

	Running (busy)
	Running (idle/waiting)
	Not Running

TBufferMerger Performance Analysis

- ROOT can saturate any media type with only a few threads if not CPU bound
- Scalability can still be an issue due to many guards on ROOT's global lock
- Need a solution for backpressure to avoid excessive memory consumption





output to SSD

	Running (busy)
	Running (idle/waiting)
	Not Running

CMS Reconstruction with TBufferMerger

32 thread RECO-AOD-MINIAOD

Module	Total Loop Time	Total Loop CPU	CPU Utilization	Events/ Second	RSS
Standard w/o IMT	1701	33989	0.62	2.94	9454
Standard w/IMT	1187	32076	0.84	4.21	8981
Parallel 6x6x3	1119	33722	0.92	4.47	13817
Parallel 1x6x3	1088	33396	0.95	4.59	10745
NoWrite	1075	33116	0.96	4.65	12140
NoFill	924	26987	0.91	5.41	7201

Summary

- Benchmarks show good scalability when CPU bound
- Output disk performance has big impact on runtime
- Investigating how to best fix identified issues
  - ROOT caches class streaming information when first needed
  - Requires write-lock during computation, read-lock for access
  - Impacts scalability when using large numbers of threads (10+)