Writing ROOT Data in Parallel with TBufferMerger

G. Amadio
for the ROOT Team

ROOT
Data Analysis Framework
https://root.cern

TBufferMerger

- Parallel data processing without need for a merge step at the end
- First available in ROOT 6.10
- Use cases
  - Facilitate parallel dataset creation (MC generation, RECO, etc)
  - Save result of **`RDataFrame`** analysis in parallel with snapshot action (see talks by Enrico and Danilo)
  - 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

Programming Model

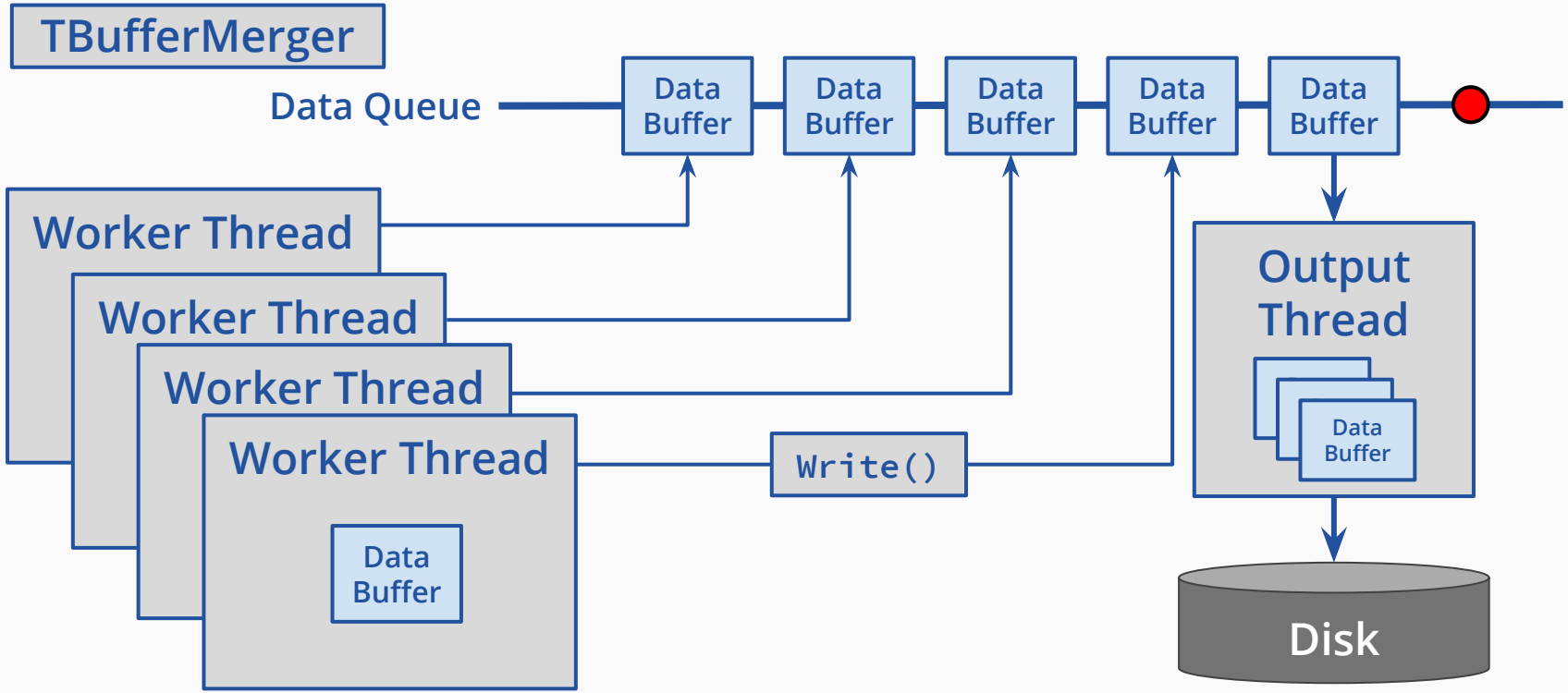
Sequential usage of TFile

```
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;
}
```

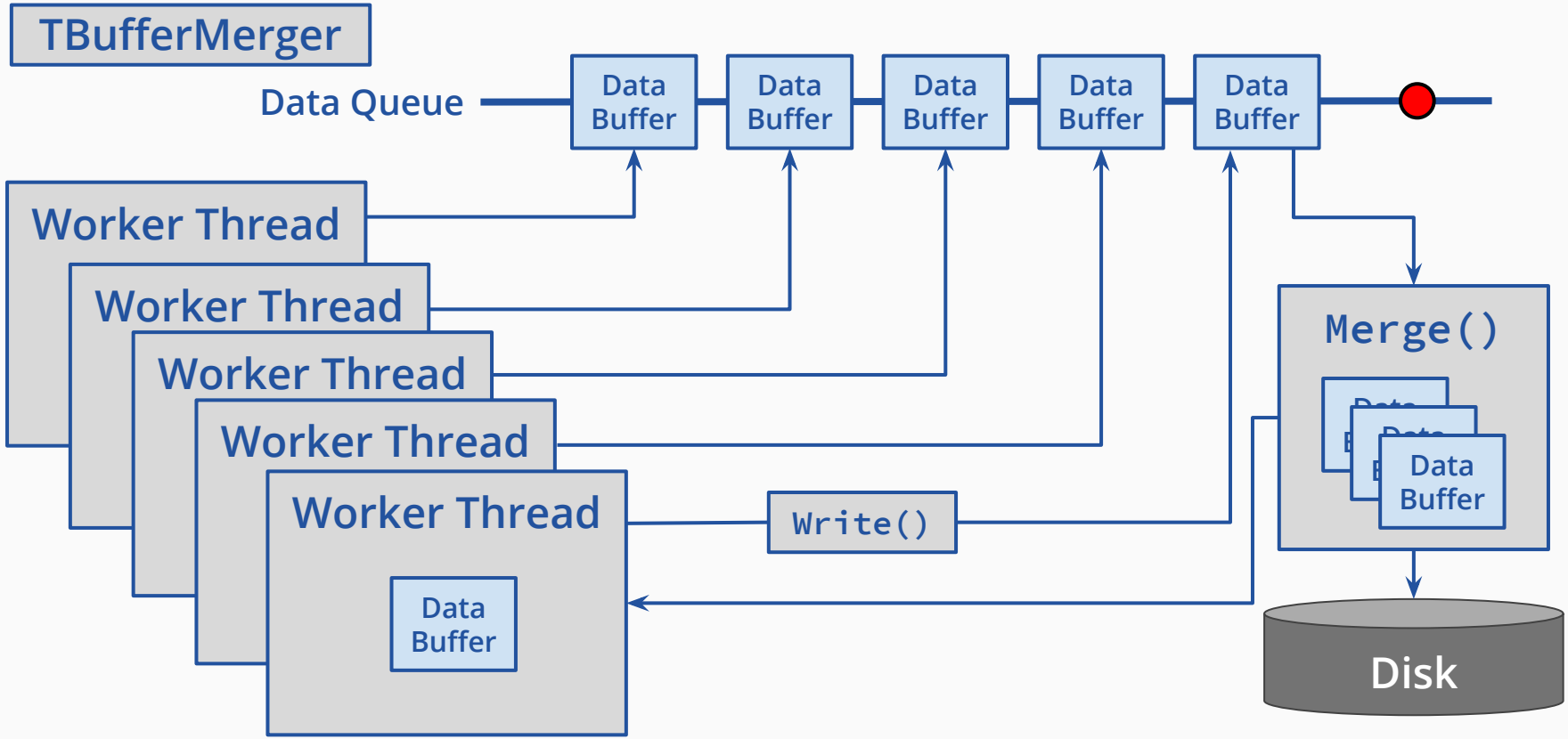
Parallel usage of TFile with TBufferMerger

```
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: First Implementation

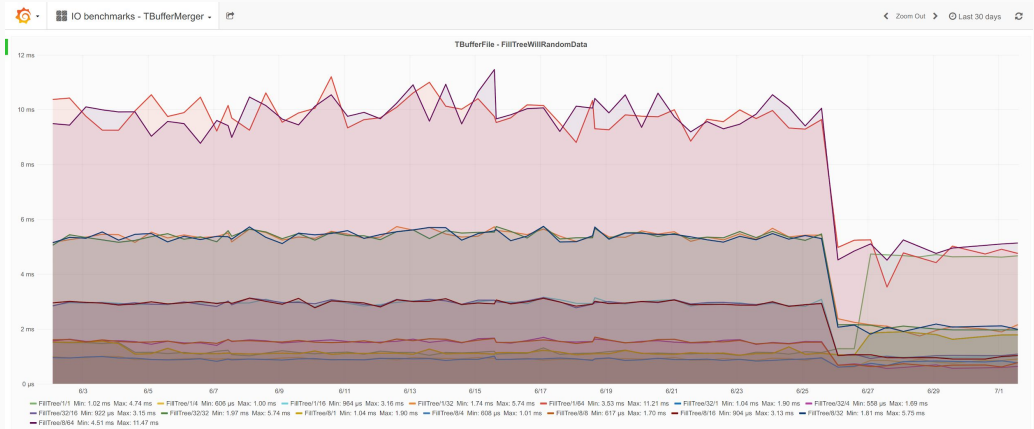


TBufferMerger: Current Implementation



Advantges of new implementation

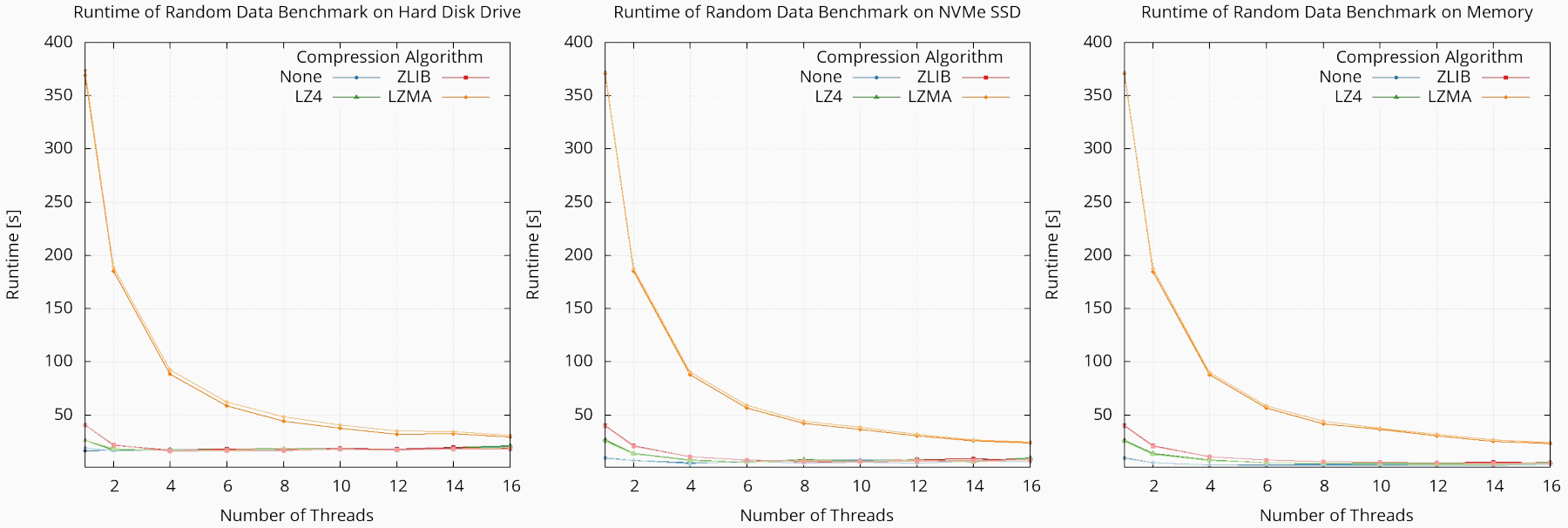
- Agnostic about model of parallelism
  - Usable both with threads as well as tasks
  - Integrates better with experiment framework
  - No oversubscription due to compression work in merging thread
- Less context switches, lock contention → better performance
- Visible gains seen in performance monitoring (see talk by O. Shadura)



TBufferMerger Single Branch Benchmark

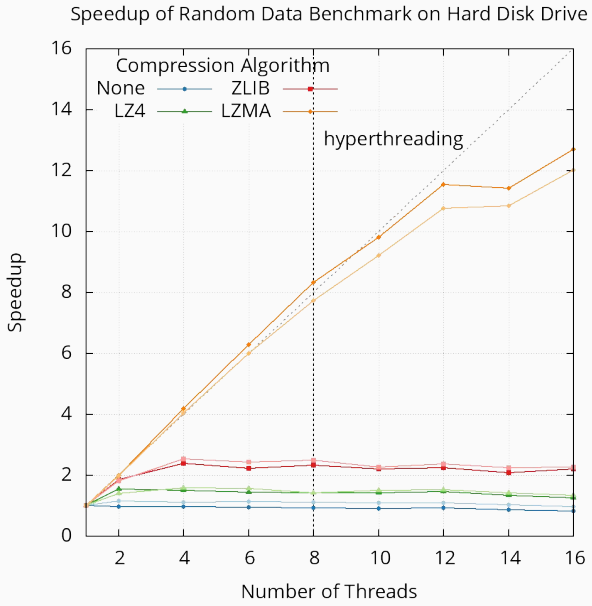
- Create ~1GB of **simple** data and write out to different media using different compression algorithms
- Synthetic benchmark that exacerbates the role of I/O by doing light amount of work (generating a random number)
- Test environment
  - Intel® Core™ i7-7820X Processor (8 cores, 11M Cache, up to 4.30 GHz)
  - Write out data to HDD, NVMe SSD, DRAM
  - Compare compression algorithms: LZ4, ZLIB, LZMA, no compression
  - Compare ROOT 6.12 (light colors) and 6.14 (dark colors)
  - GCC 8.1.0, C++17, -O3 -march=native (skylake-avx512), release build

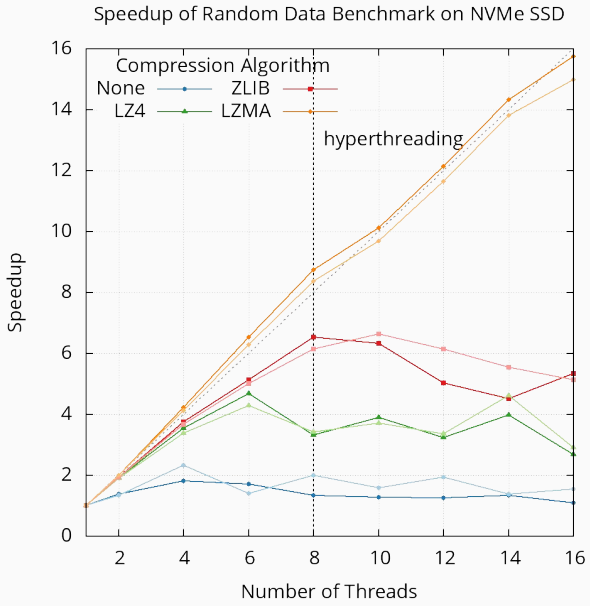
Single Branch Benchmark: Runtime

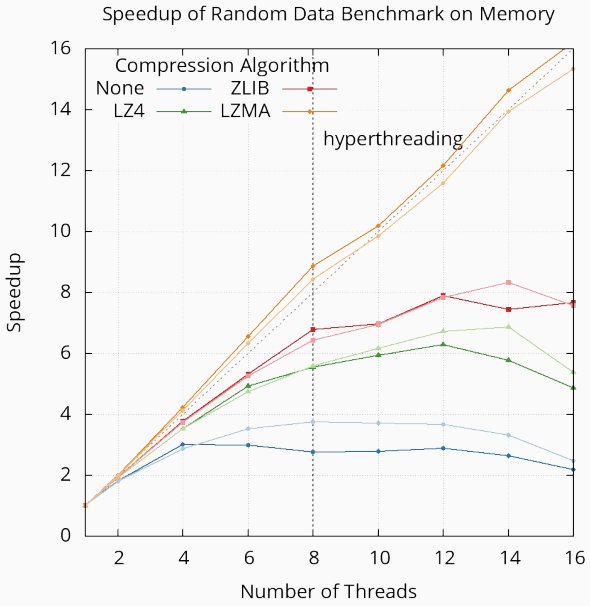


light colors = ROOT 6.12, dark colors = ROOT 6.14

Single Branch Benchmark: Speedup







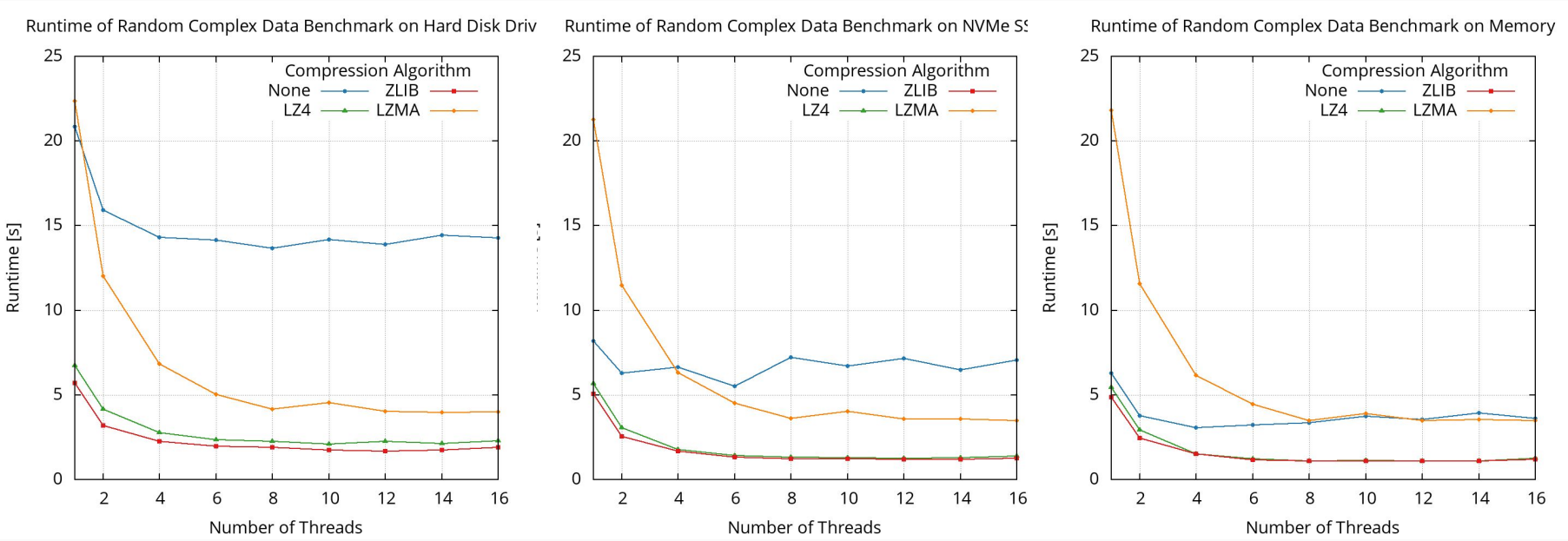
light colors = ROOT 6.12, dark colors = ROOT 6.14

TBufferMerger Multi Branch Benchmark

- Create ~1GB of **complex** data and write out to different media using different compression algorithms
- Synthetic benchmark that enhances the role of the I/O subsystem by creating variable number of branches
- 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 buildl

Multi Branch Benchmark: Runtime

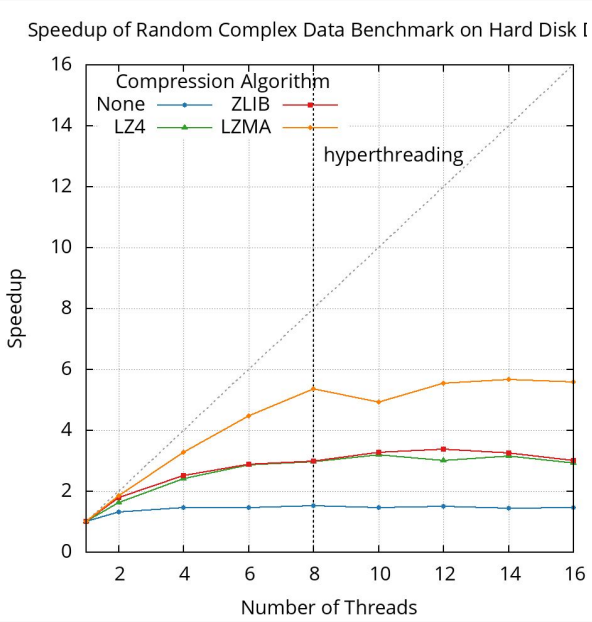
- Branch = std::vector<Event> (3x Vector3D, 3x double, 3x int)
- Test creates 10 branches, each with a vector of 10 Events

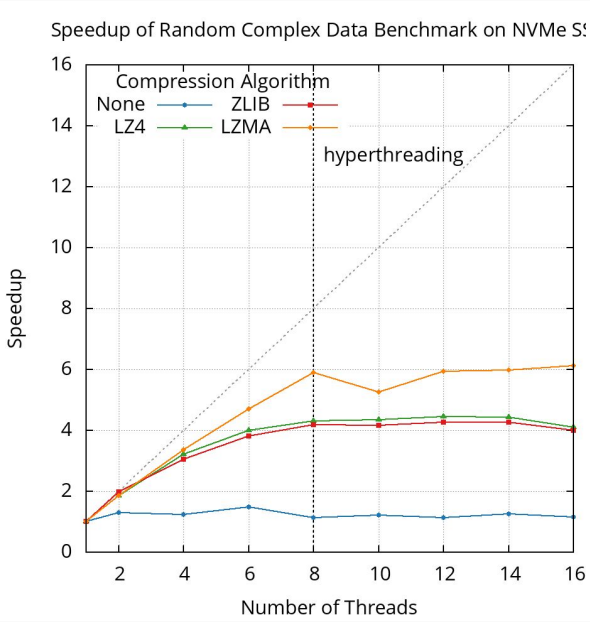


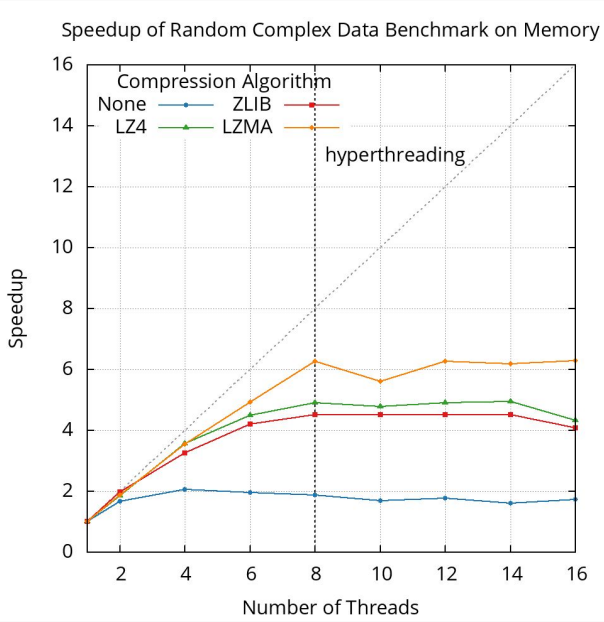
All figures using ROOT master branch

Multi Branch Benchmark: Speedup

- Branch = std::vector<Event> (3x Vector3D, 3x double, 3x int)
- Test creates 10 branches, each with a vector of 10 Events



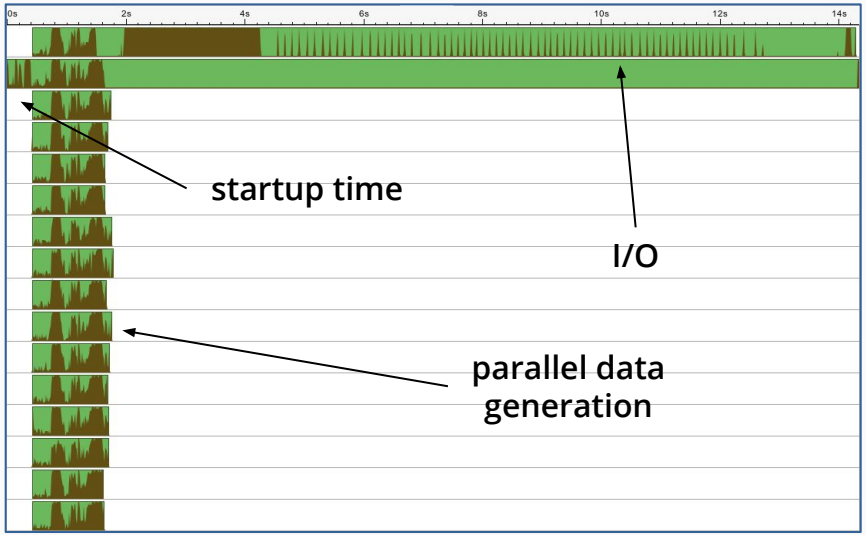




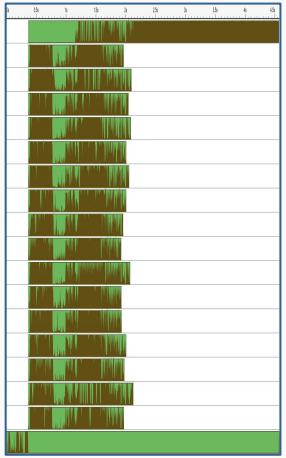
All figures using ROOT master branch

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

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
- Ivestigating how to best fix identified performance 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+)