TBufferMerger Performance Benchmarks Revisited

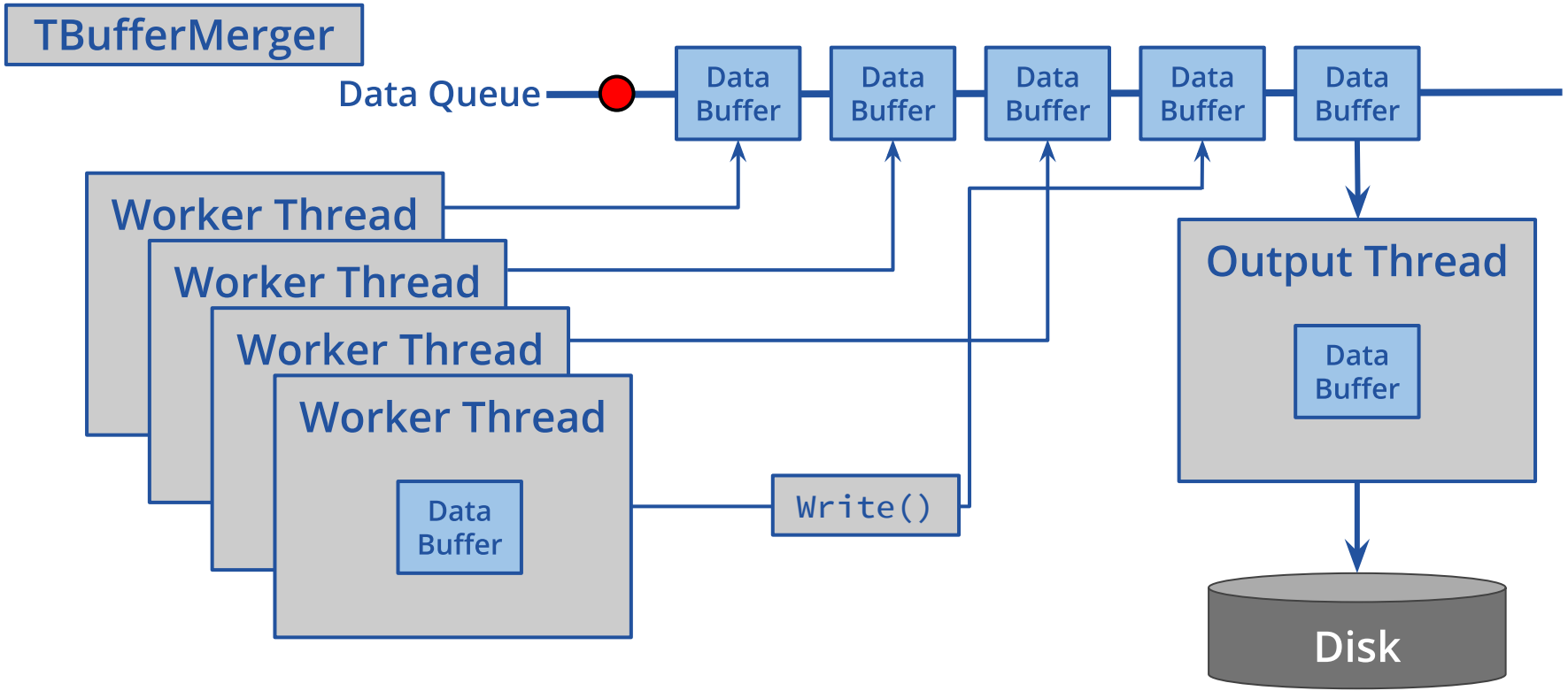
G. Amadio
for the ROOT Team

ROOT
Data Analysis Framework
https://root.cern

Overview

- TBufferMerger
- Performance Benchmarks
- Current Issues
- Possible Solutions
- Summary

TBufferMerger Class

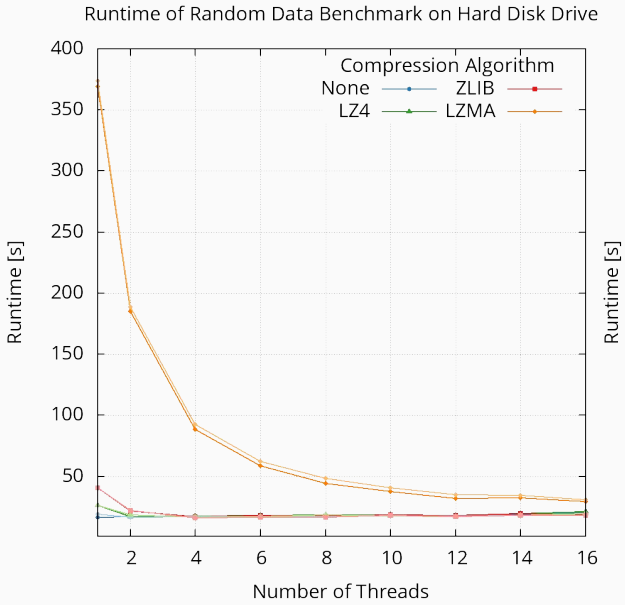


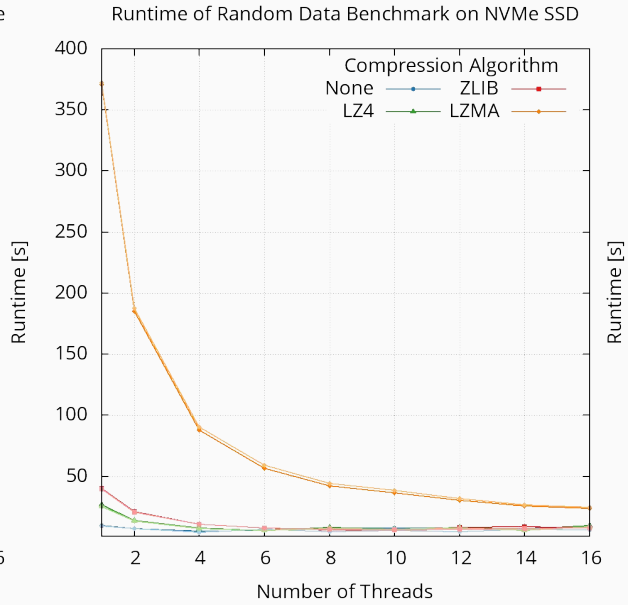
TBufferMerger Performance

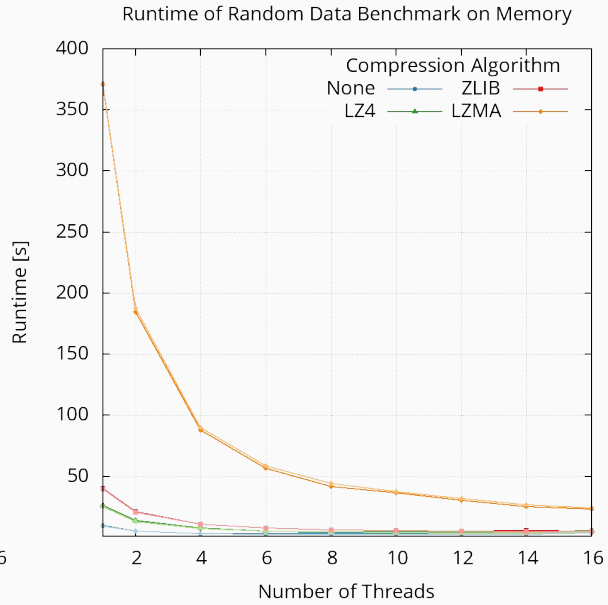
TBufferMerger Performance Benchmark

- Create ~1GB of data and write out to different media using different compression algorithms
- Synthetic benchmark that exacerbates the role of I/O by doing lightweight work
- 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

TBufferMerger Benchmark: Runtime

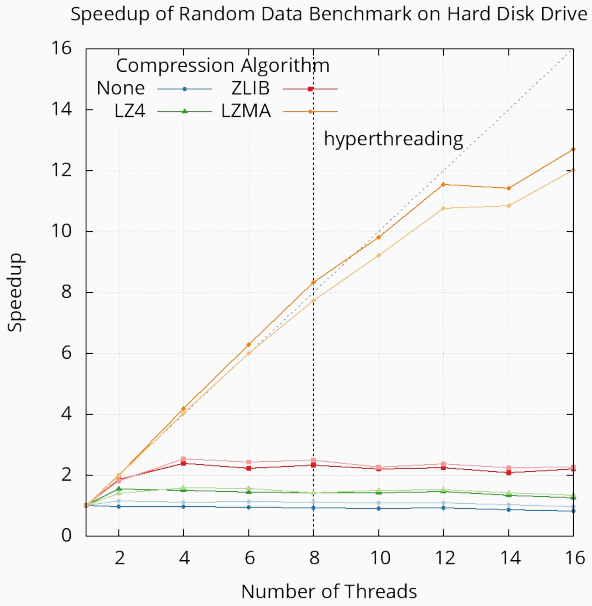


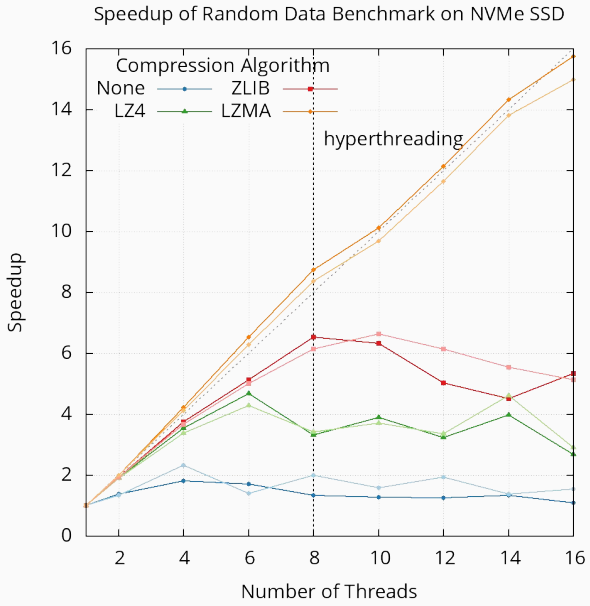


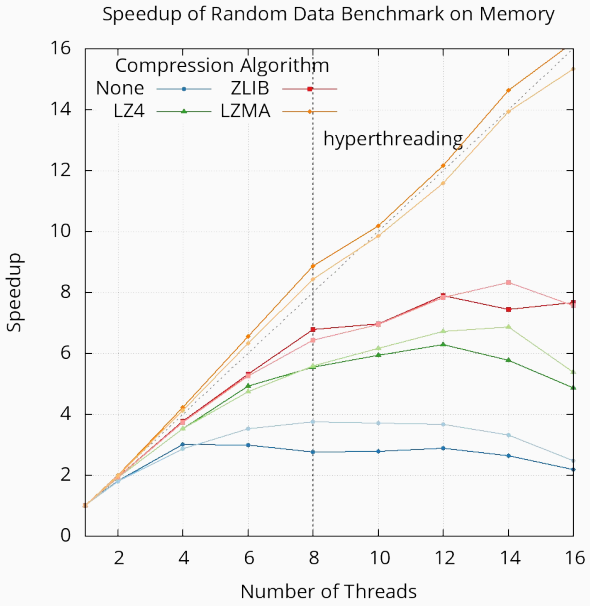


light colors = ROOT 6.12, dark colors = ROOT 6.14

TBufferMerger Benchmark: Runtime



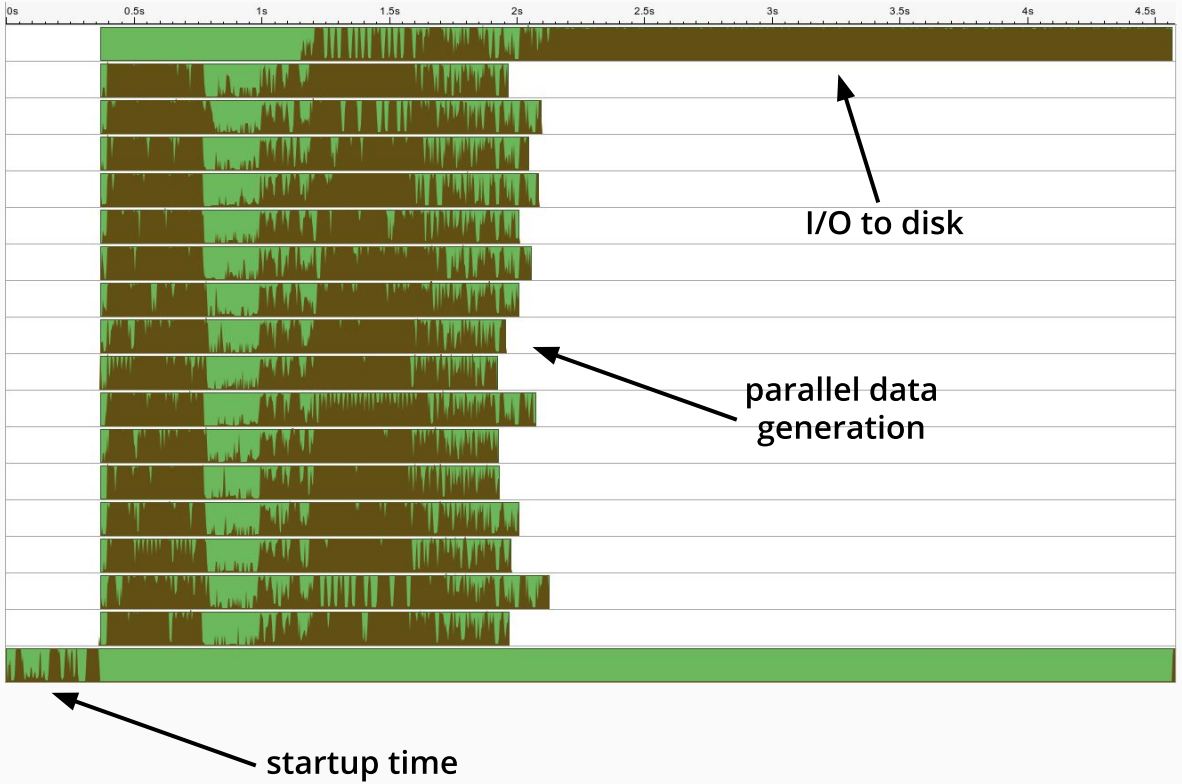




light colors = ROOT 6.12, dark colors = ROOT 6.14

TBufferMerger Performance Analysis

- ROOT can saturate any media type with only a few threads if not CPU bound
- Startup time can be significant depending on media
- 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



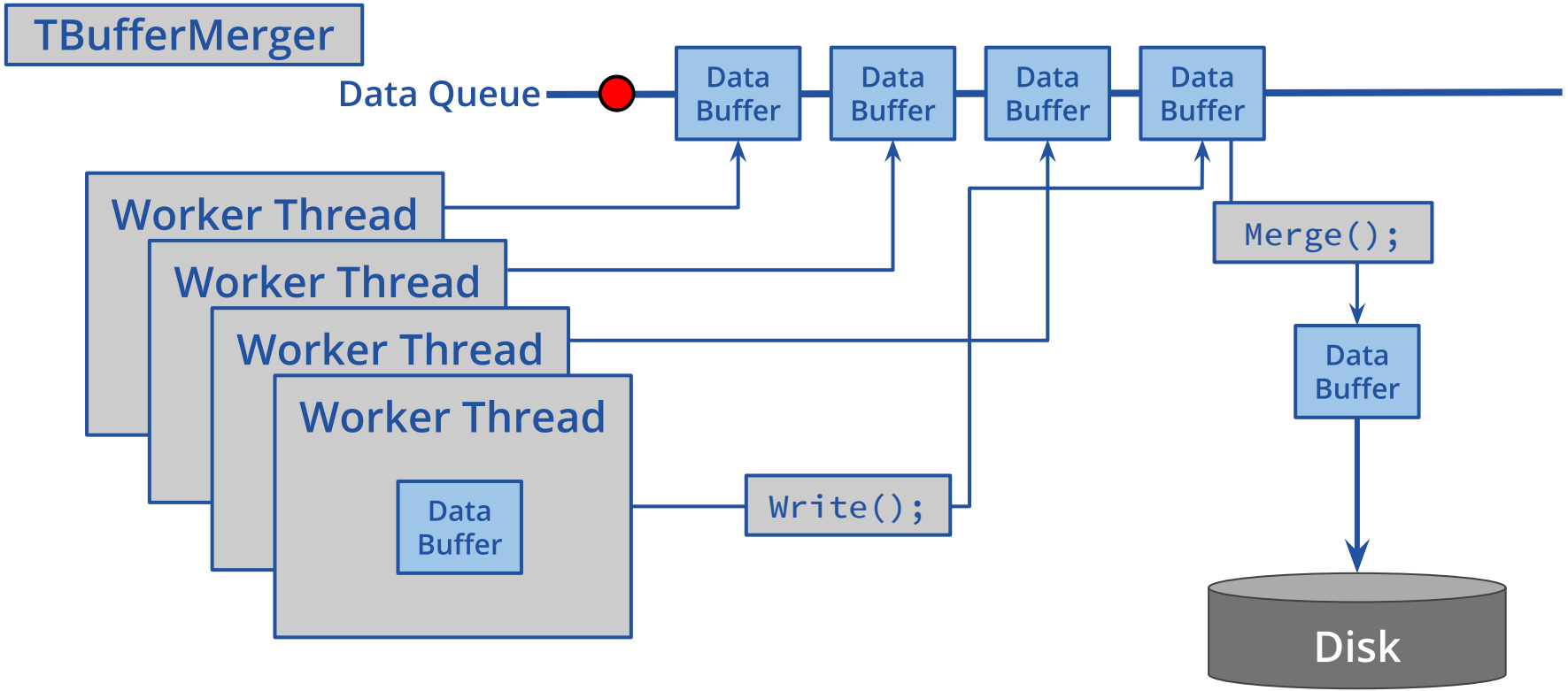
TBufferMerger Issues

- TBufferMerger has no control for limiting data rate
- Can result in large amounts of resident memory during a run if output media cannot keep up
- Difficult to control data rate from TBufferMerger without blocking
- AutoSave doesn’t work when saving every buffer is still too slow
- Merging thread can do too much compression work

TBufferMerger Evolution

- Solution may be to embed merging work into users' threads/tasks
- If TBB is enabled, can launch a task instead of just running a merge
- May interfere with other threads due to necessity of taking ROOT's global interpreter lock, preventing scalability
- Patch exists, but has not been merged because it requires removing current callback functionality to avoid deadlocks

TBufferMerger Without Output Thread



Summary

- Performance is highly dependent on output media
- Benchmarks show good scalability when use case is not I/O bound
- This is the case when LZMA is used, due to large compression work compared to ZLIB, LZ4 compression
- Time using both ZLIB and LZ4 is much lower than LZMA, so CPU cost might erase benefit of higher compression ratio
- TClingClassInfo::GetBaseOffset() degrades performance, need to try to circumvent the need to take ROOT's global interpreter lock there

Backup Slides

Hotspots Analysis in VTune 2018

Process / Module / Class / Function	CPU Time ▼ — Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Instructions Retired	CPI Rate
random-tbm	15.281s	66,866,400,000	0.880
libTree.so	7.990s	44,190,000,000	0.710
TTree	3.921s	15,512,400,000	0.989
TTree::Fill	3.921s	15,508,800,000	0.989
TTree::OptimizeBaskets	0s	3,600,000	0.000
TBranch	3.129s	23,299,200,000	0.521
TLeafD	0.449s	3,207,600,000	0.627
TBasket	0.325s	705,600,000	1.730
TLeaf	0.097s	853,200,000	0.456
[Not part of any known object class]	0.066s	608,400,000	0.396
TTreeCloner	0.003s	3,600,000	2.000
libc-2.27.so	3.091s	867,600,000	13.261
[Not part of any known object class]	3.091s	867,600,000	13.261
__memmove_avx_unaligned_erms	2.701s	864,000,000	11.725
__memset_avx2_erms	0.390s	3,600,000	369.000
random-tbm	2.021s	11,437,200,000	0.704
libRIO.so	1.641s	9,165,600,000	0.672
TBufferFile	0.849s	5,522,400,000	0.564
TBufferIO	0.512s	2,048,400,000	0.951
TDirectoryFile	0.255s	1,562,400,000	0.654
TKey	0.011s	10,800,000	2.000
TFile	0.006s	10,800,000	1.000
TStreamerInfo	0.004s	10,800,000	1.333
TMemFile	0.004s	0	
libCore.so	0.304s	658,800,000	1.563
libCling.so	0.138s	388,800,000	1.000
libpthread-2.27.so	0s	28,800,000	2.625
[Unknown]	0.032s	7,200,000	11.000
ld-2.27.so	0.029s	61,200,000	0.941
libThread.so	0.016s	25,200,000	2.000
libz.so.1.2.11	0.011s	28,800,000	1.250

TBufferMerger 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;
}
```