In process TTree Parallel Merge with new TBufferMerger class

G. Amadio and D. Piparo for the ROOT Team

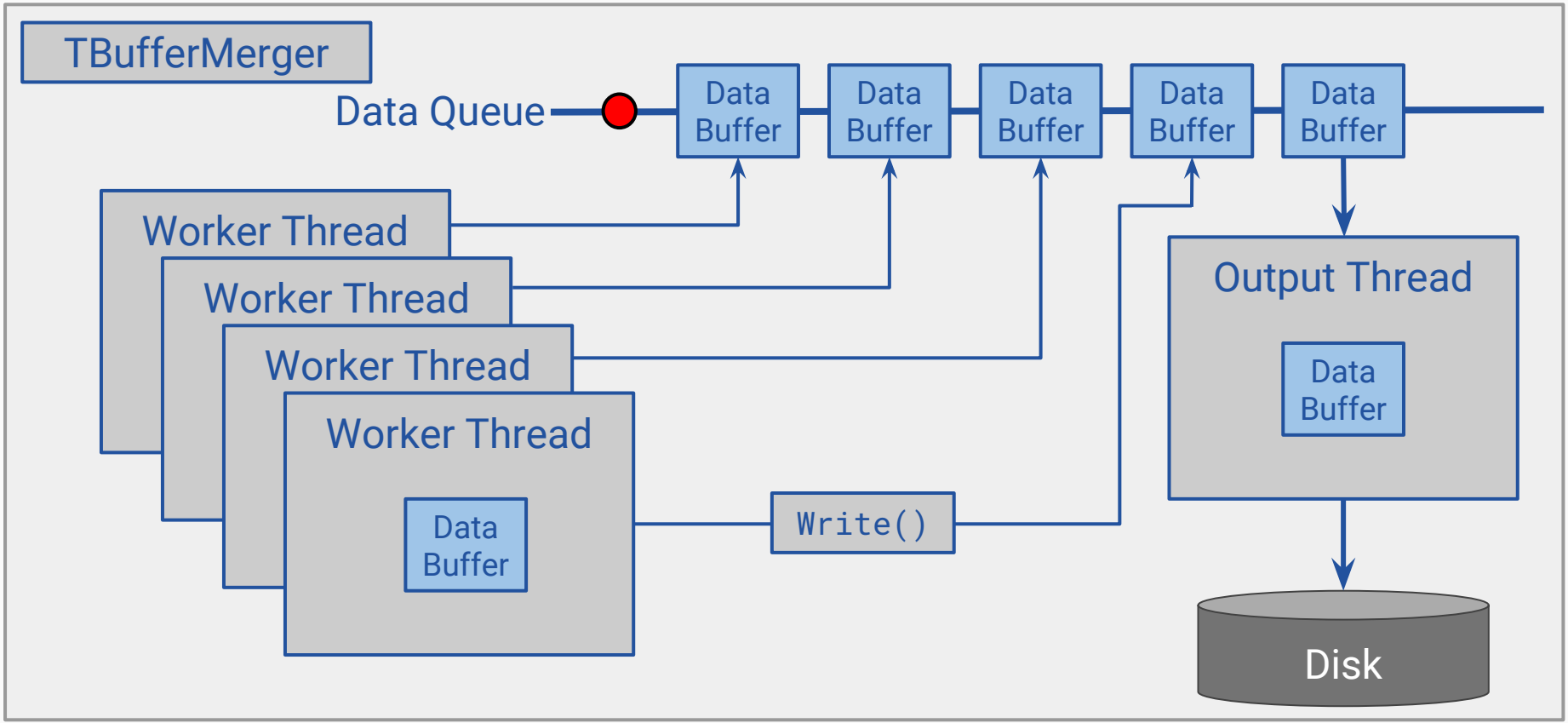
New feature in ROOT: Writing TTree in parallel

- New TBufferMerger class, using TFileMerger, can now be used to write to a TTree in parallel from multiple threads
- Based on similar concept class from GeantV project
- TBufferMerger acts as a factory of slightly modified TMemFiles that write buffers into a merging queue
- Output from all threads is automatically merged in the background by a separate thread launched by TBufferMerger

Motivation

- Prior parallel solutions involved creating many files and merging them at the end, cumbersome with many threads
- With the new TBufferMerger, the user can get a single file with everything automatically merged with little overhead
- Example use case: generate Pythia events in parallel and save them into a single TTree in a ROOT file
- TBufferMerger was created to allow the snapshot action in TDataFrame to write out user-defined branches based on the input data back into a ROOT file in parallel

Data Flow with TBufferMerger



Programming Model: TFile vs TBufferMerger

TFile

```
#include "TFile.h"
#include "TTree.h"

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 main(int argc, char **argv)
{
   size_t nEntries = 16*1024*1024;

   {
      auto f = TFile::Open("myfile.root");
      auto t = new TTree("mytree","mytree");

      Fill(&t, 0, nEntries);

      f.Write();
   }

   return 0;
}
```

TBufferMerger

```
#include <thread>
#include "ROOT/TBufferMerger.hxx"
#include "TTree.h"

void Fill(TTree *t, int init, int count);

int main(int argc, char **argv)
{
   size_t nWorkers = 8;
   size_t nEntries = 16*1024*1024;
   size_t nEntriesPerWorker = nEntries/nWorkers;

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

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

   for (size_t i = 0; i < nWorkers; ++i)
      workers.emplace_back([=, &merger]()
      {
         auto f = merger.GetFile();
         auto t = new TTree("random", "random");

         t->ResetBit(kMustCleanup); // avoid unresolved bug in ROOT

         Fill(t, i * nEntriesPerWorker, nEntriesPerWorker);

         f->Write();
      });

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

   return 0;
}
```

Programming Model: Pythia event generation

```
#include <thread>
#include <vector>

#include "Pythia8/Pythia.h"
#include "ROOT/TBufferMerger.hxx"
#include "TROOT.h"
#include "TTree.h"

using namespace ROOT::Experimental;

int main() {

   size_t nWorkers = 4;
   size_t nEvents  = 1000;
   size_t nEventsPerWorker = nEvents/nWorkers;

   std::string filename("pythia-tbm.root");

   gROOT->SetBatch();
   ROOT::EnableThreadSafety();

   TBufferMerger merger(filename.c_str(), "recreate");
```

```
   auto pythia_gen = [=, &merger]() {
      Pythia8::Pythia pythia;
      pythia.readString("HardQCD:all = on");
      pythia.readString("PhaseSpace:pTHatMin = 20.");
      pythia.readString("Beams:eCM = 14000.");
      pythia.init();

      auto f = merger.GetFile();
      Pythia8::Event *e = &pythia.event;
      auto t = new TTree("pythia", "pythia");
      t->ResetBit(kMustCleanup);
      t->Branch("event", &e);

      for (size_t n = 0; n < nEventsPerWorker; ++n) {
         while (!pythia.next());
         t->Fill();
      }
      f->Write();
   };

   std::vector<std::thread> workers;
   for (size_t i = 0; i < nWorkers; ++i)
      workers.emplace_back(pythia_gen);

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

   return 0;
}
```

Programming Model: TBufferMerger vs TDF

TBufferMerger

```
#include <random>
#include <thread>
#include "ROOT/TBufferMerger.hxx"
#include "TTree.h"

int main(int argc, char **argv)
{
   size_t nWorkers = 8, nEntries = 16*1024*1024;

   ROOT::EnableImplicitMT(nWorkers);
   ROOT::Experimental::TBufferMerger merger;

   auto fill_random = [&]() {
      thread_local std::default_random_engine g;
      std::uniform_real_distribution<double> dist(0.0, 1.0);
      auto f = merger.GetFile();
      auto t = new TTree("random", "random");
      t->ResetBit(kMustCleanup);
      double rng;
      t->Branch("random", &rng);

      for (size_t i = 0; i < nEntriesPerWorker; ++i) {
         rng = dist(g);
         t->Fill();
      }
      f->Write();
   };

   std::vector<std::thread> workers;
   for (size_t i = 0; i < nWorkers; ++i)
      workers.emplace_back(fill_random);

   for (auto&& worker : workers) worker.join();
   return 0;
}
```

TDataFrame

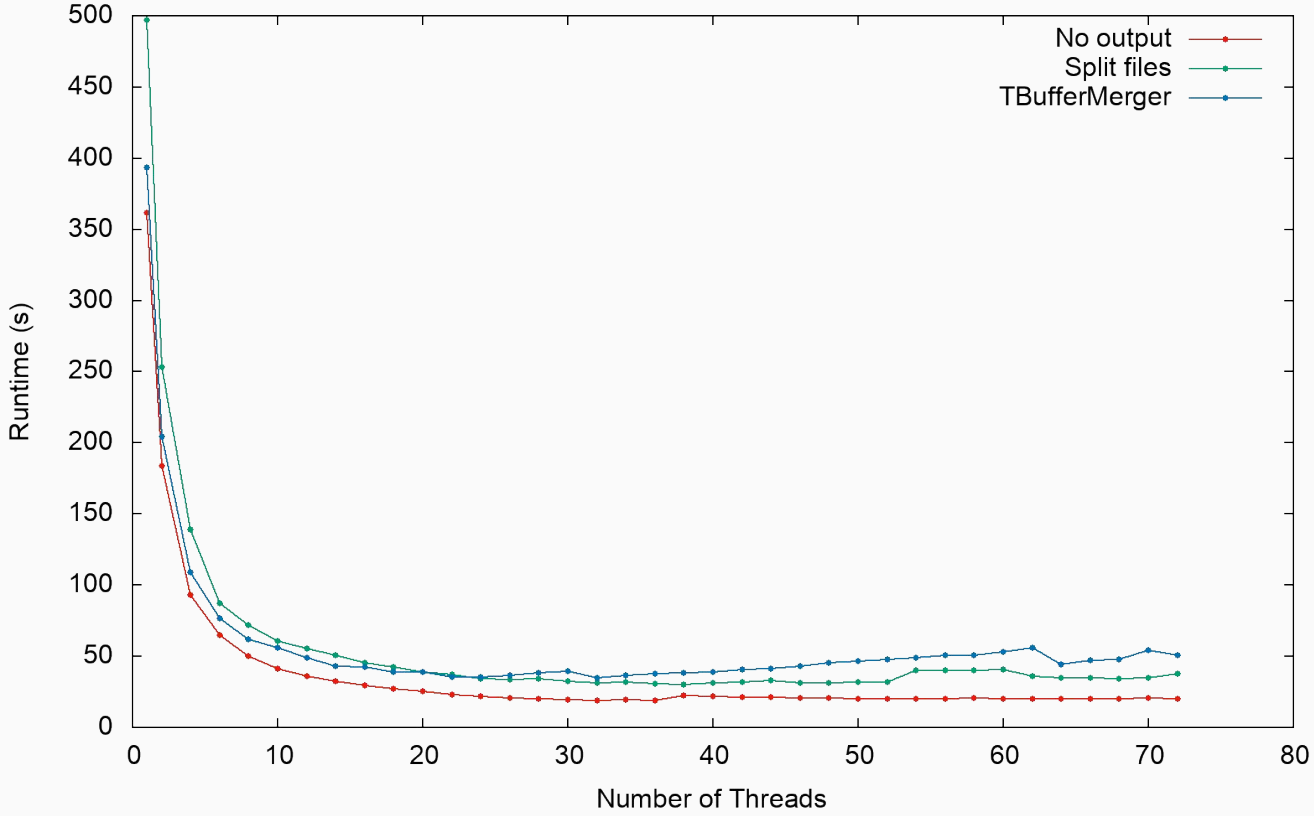
```
#include <random>
#include "ROOT/TDataFrame.hxx"
#include "TTree.h"

int main(int argc, char **argv)
{
   size_t nWorkers = 8, nEntries = 16*1024*1024;

   ROOT::EnableImplicitMT(nWorkers);
   ROOT::Experimental::TDataFrame tdf(nEntries);

   auto fill_random = [=]() {
      thread_local std::default_random_engine g;
      std::uniform_real_distribution<double> dist(0.0, 1.0);
      return dist(g);
   };

   tdf.Define("random", random)
      .Snapshot<double>("tree", "random.root", { "random" });

   return 0;
}
```

TBufferMerger Benchmarks

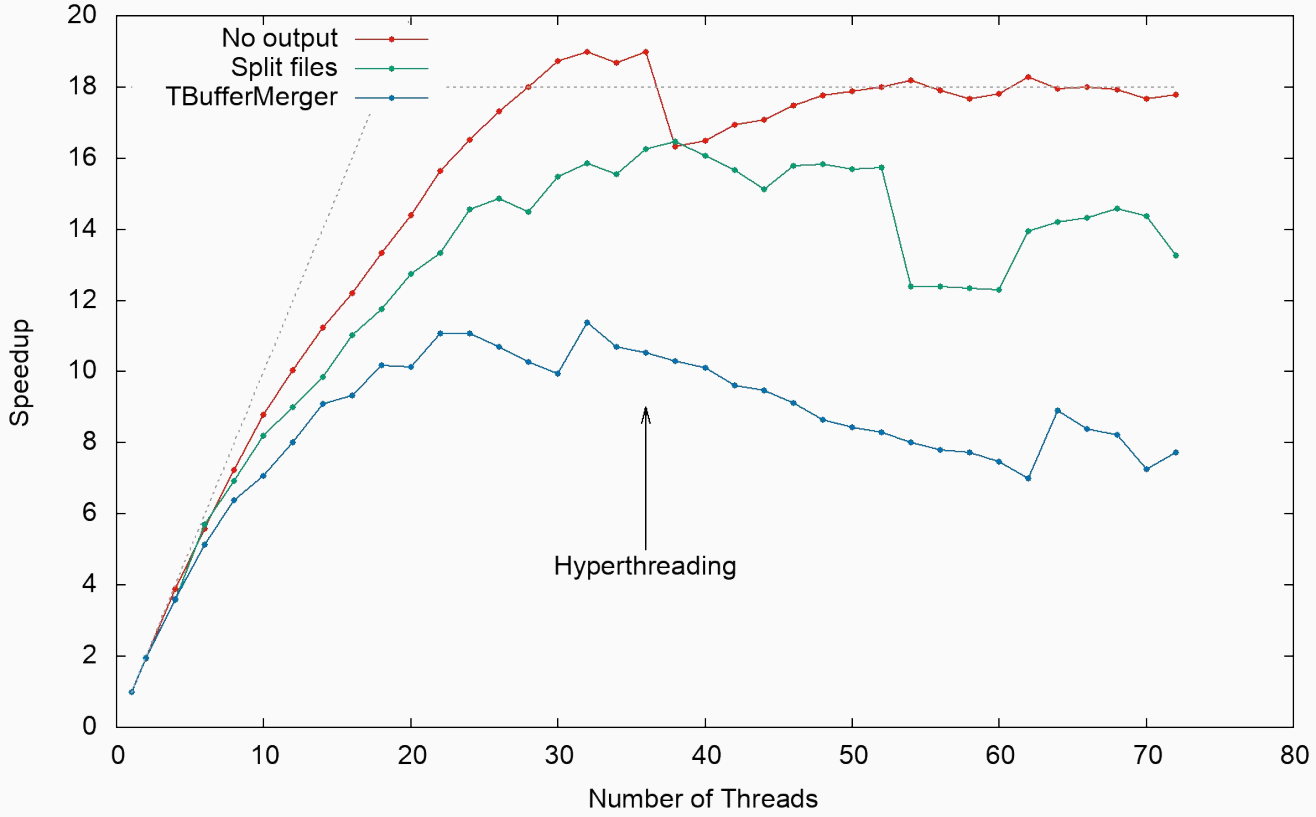
Hardware configuration:

- Partnership with São Paulo State University (UNESP)
- Server: 2x Intel Xeon E5-2699 v3 (2.30GHz, 128GB RAM)
- Intel Xeon Phi KNL (68 cores, 1.4GHz, 210GB RAM)

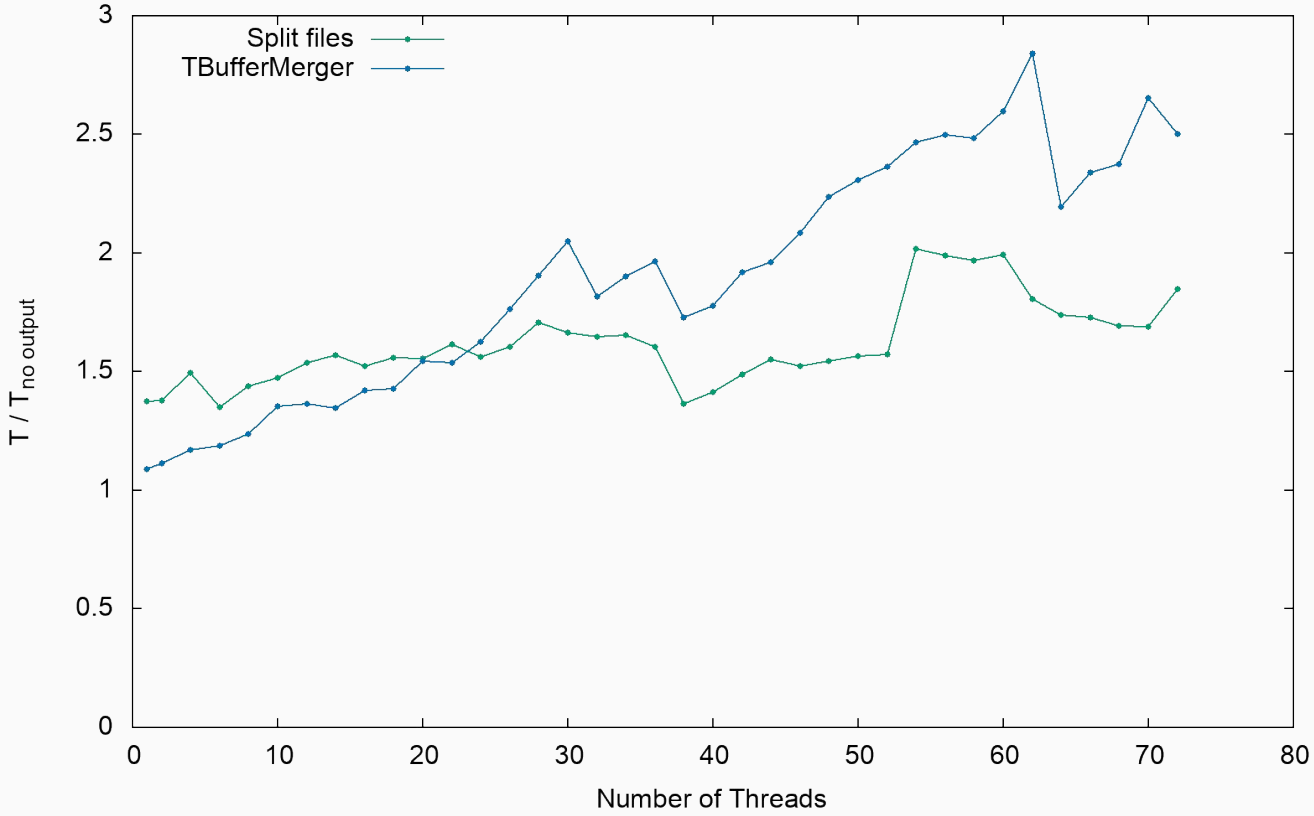
Pythia Event Generation: Runtime



Pythia Event Generation: Speedup



Pythia Event Generation: Relative Speedup



Summary and Conclusion

- New TBufferMerger class allows to write TTree in parallel
- Benchmarks performed on a dual-socket 18 core Xeon server at UNESP, more benchmarks to be run on Knights Landing
- Good performance compared with writing to multiple files
  No significant relative overhead up to ~30 threads
- Parallel snapshot action now available without changes in user code other than calling `ROOT::EnableImplicitMT()`
- However, some scaling issues remain for large numbers of worker threads, currently under investigation