Multithreading experience in CMS reconstruction

HSF Reco and Trigger meeting
April 1, 2020
Slava Krutelyov (UCSD)

Note: due to apparent miscommunication on the topic of this presentation, this talk is not about CMS experience with code optimization

— Status of multithreaded execution in reconstruction
— Threads vs jobs: resource use optimization choice
— Improving efficiency with concurrent "luminosity blocks"
— Things to explore

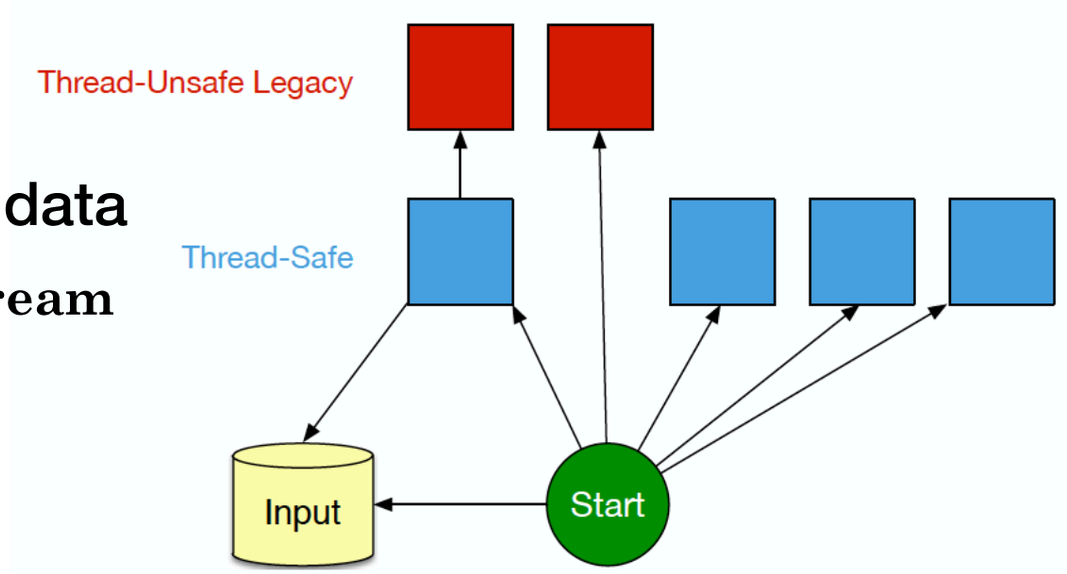
Note: this talk does not cover developments at CMS for heterogeneous computing software, which is actively developing and is planned for deployment in the software trigger (HLT) in Run-3.

Status of multithreading

- **Standard production workflows for both reconstruction and HLT are using multiple threads since the beginning of Run-2. Threads are scheduled with tbb.**
- **Reconstruction jobs on sites are typically running in 8 threads, motivated by the balance of scheduling job payloads and available core-counts on sites**
- **Various elements contribute to thread (in)efficiency, most are outside of the proper "reconstruction" work, like I/O or synchronization driven by changes in conditions.**

➡ Events are processed in "streams", units of managing modules processing single event data

- Production jobs set up for one thread per stream
- Scheduling is dynamic
- Multiple threads per stream can help



➡ Types of modules supported by framework

✓ ::one : only one instance can run, no concurrency based on declared shared resource

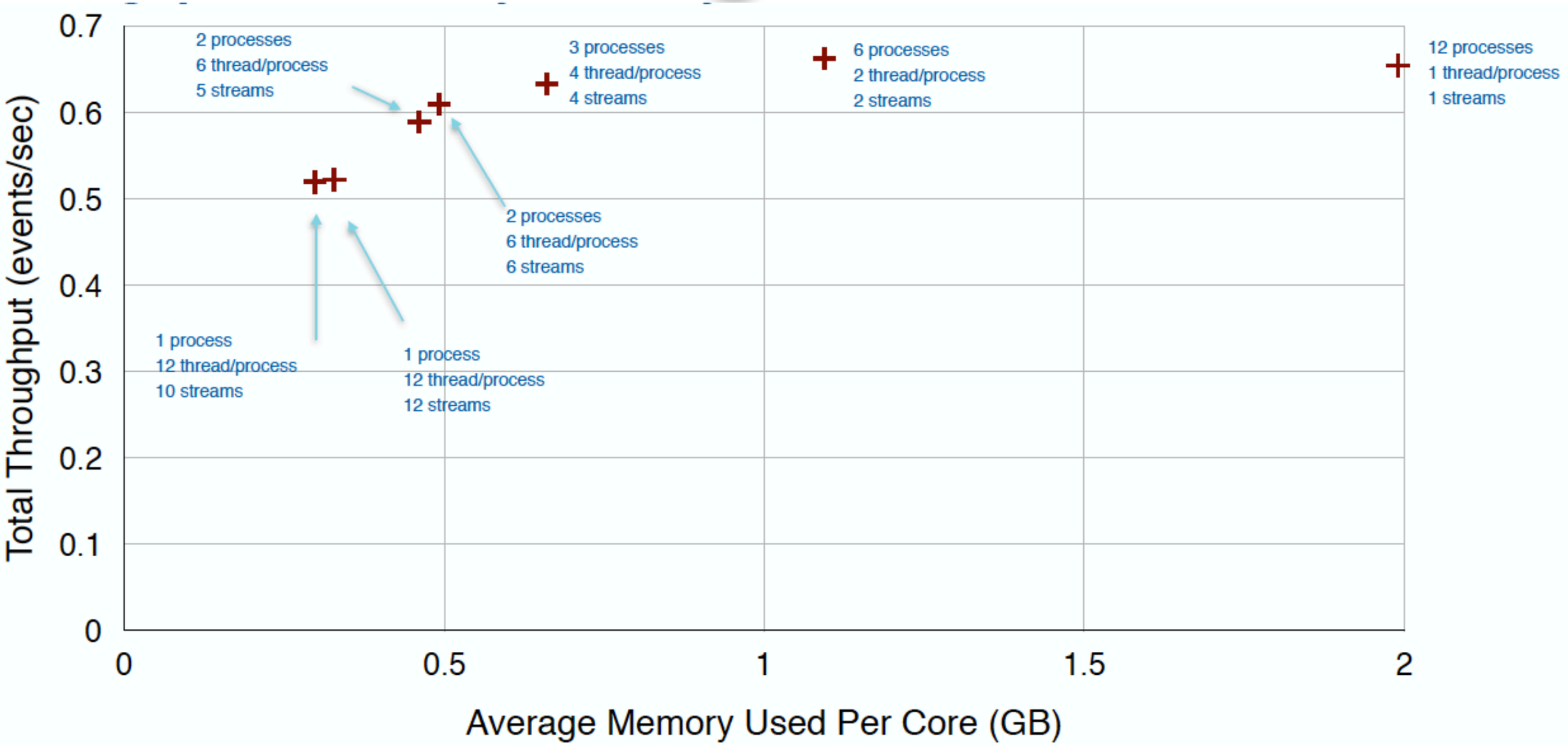
✓ ::stream : one instance is created per stream

✓ ::global : one instance can be called concurrently, ~reentrant

➡ Most reconstruction modules are ::stream modules (no ::one/legacy modules)

✓ Job inefficiency is around 2-3%, somewhat constrained by data I/O

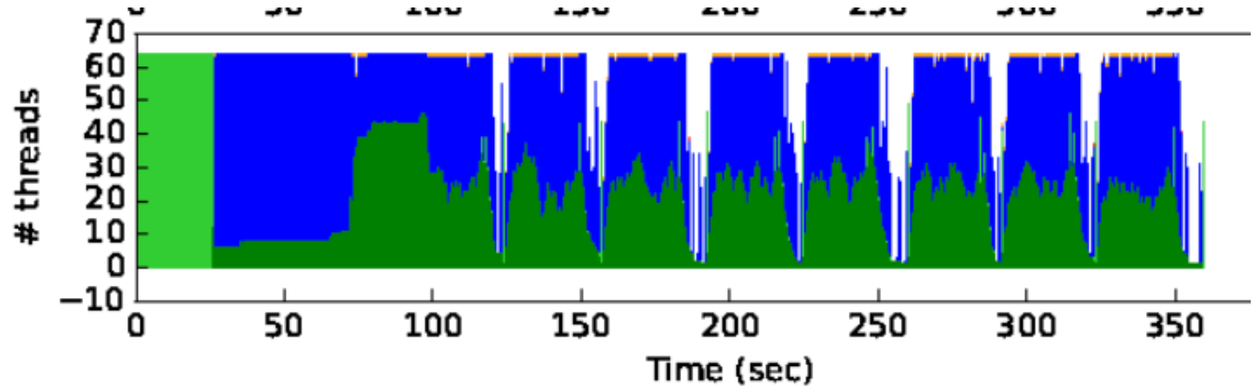
Threading vs resources

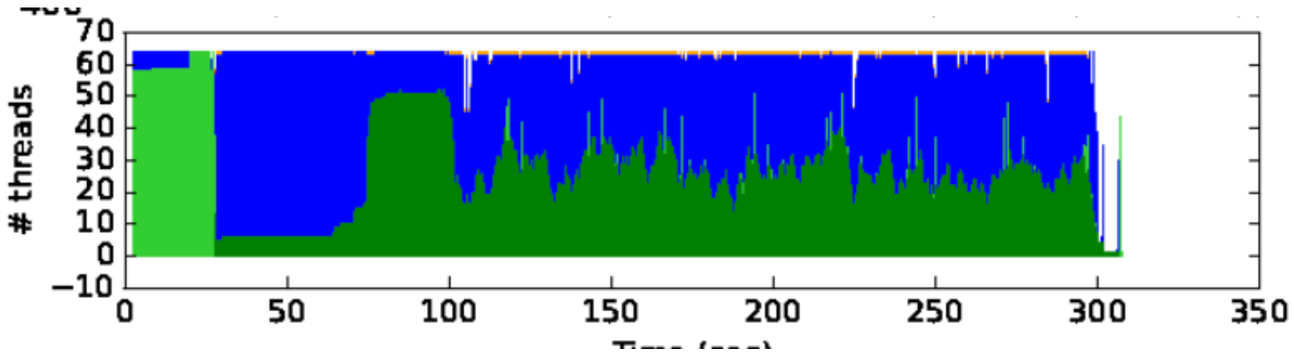


- **[example shown in CHEP 2016, C. Jones]**
- **Memory use per core is the most visible benefit in running multithreaded jobs**
- **Additional effort is made for ::stream modules to still rely on common global condition/configuration data when deploying new algorithms or in optimization of existing algorithms**

Concurrent luminosity blocks

- **Somewhat recent functionality supported by framework: running luminosity blocks concurrently. This is available since 2018 (full deployment still in progress)**
  - ✓ NB: a luminosity block is 23 s, during which conditions data is [implied to be] unchanged; it is also a unit of accounting for data files.
  - ✓ In simulation, lumi blocks are artificial, driven by job time/scheduling needs
  - ✓ In data the number of events in a single lumi block is defined by the trigger rate in a given dataset. Since we [still] do not fragment lumi blocks into separate jobs, max processing time of 48 hours leads to limitations of max rate from trigger in one dataset
- **In many cases neighboring lumis have the same conditions and can be processed concurrently**
  - ✓ lumi boundary inefficiency goes away (unfilled area), as seen in a stall graph





Things to explore

- **Event level parallelism**
  - ✓ Several cases exist already with separable work loops using parallel_for
    - **This helps to better fill gaps in processing mostly in the tails of processing**
  - ✓ Module scheduling by framework is already fairly dynamic to fill most available gaps
  - ✓ Only “embarassingly parallel” work can easily benefit from event level parallelism

- **We could benefit more from regular threading scaling tests with an outlook for possibly running with many more threads**
  - ✓ There was more motivation here when MIC architecture became a possible reality. However current multi-CPU/core hardware status reduced this pressure.
  - ✓ These tests are a good way to stress-test the applications and find bottlenecks or issues
  - ✓ Serialization in I/O is likely to limit this effort for production needs

Summary

- **CMS has successfully utilized multi-threaded processing jobs**
  - ✓ All prompt reconstruction for Run 2 were multi-threaded jobs
- **Allowing multiple threads per event allows**
  - ✓ processing of more memory intensive jobs
  - ✓ utilization of machines with lower memory per core limits