Thread safety in SIOVDetPedestalService and SIOVChannelStatusService

Giuseppe Cerati (FNAL)

LArSoft Coordination Meeting

Nov. 1, 2022

Introduction

- art and larsoft provide multithreading capabilities through TBB library
  - art multithreading can process concurrently data across events or within the same event
  - see talk1 and talk2

- Grid allocations have total available memory split by CPU cores
- Grid jobs often need slots with large memory, thus getting multiple cores
- Production jobs are however running single-threaded, thus use only one core

- We can achieve significant processing speedups if we are able to exploit multithreading and increase our core utilization efficiency
  - multithreading within the event doesn't need to load more event data, can exploit unused cores given the same memory allocation

Services and Multithreading

- Art does not allow to run multithreaded if services are not thread safe and consequently marked as “SHARED”
  - see this talk by Kyle for details

- Currently in ICARUS stage0_run2_icarus_mc.fcl the following services are loaded, and only the first two are LEGACY (not SHARED)
  - SIOVChannelStatusService, SIOVDetPedestalService, DetectorClocksServiceStandard, DetectorPropertiesServiceStandard, SignalShapingICARUSService, IcarusGeometryHelper, ICARUSChannelMap, LArPropertiesServiceStandard

- Scisoft team has been working on larsoft services with the goal of making them thread safe. Work is however taking significant time as changes are non-trivial and require to be propagated to downstream experiment code

However…

- Scisoft team is targeting thread safety both across and within events
- Since we only care about the latter, the situation is significantly simpler:
  - SIOVChannelStatusService and SIOVDetPedestalService access information from a DB
  - Thread safety within events only requires that the DB access is done at event boundaries
  - Once this is enforced we can make them SHARED, and prevent them to be used when parallelism across events is attempted
    - using the “EnsureOnlyOneSchedule” functionality (link to class)

- Test branch of larevt doing what described above
  - https://github.com/LArSoft/larevt/compare/develop...cerati:larevt:feature/cerati_EnsureOnlyOneSchedule

Implementation (V1)

- Draft PR: https://github.com/LArSoft/larevt/pull/18
  - CI builds are successful (failure are due to broken workflows)

- Main idea:
  - services inherit from EnsureOnlyOneSchedule to throw exception if >1 schedules used
  - enforce DB update takes place at event boundaries SIOVXService::PreProcessEvent
    - previously only updating the time stamp here
  - prevent code other than the service to update the provider
    - make Update and UpdateTimeStamp private, and add the service as friend class
  - change SIOVXService to SHARED

- Possible limitation:
  - current develop version is “fully lazy”, with DB updates only when needed (e.g. GetChannelStatus)
  - if services are loaded in a job but not used then unnecessary pressure is put on the DB

Other possible implementations

- “V3”
  - revert changes to services: still LEGACY and only time stamp update in PreProcessEvent
  - add new services “XEnforceEventUpdate” implementing features as V1, to be used only where needed to avoid unnecessary calls to DB
  - https://github.com/cerati/larevt/compare/feature/cerati_EnsureOnlyOneSchedule...feature/cerati_EnsureOnlyOneScheduleV3

- “V4”
  - same as V1 except only time stamp update in PreProcessEvent
  - add mutex to DBUpdate function for thread safety while keeping service lazy
  - https://github.com/cerati/larevt/compare/feature/cerati_EnsureOnlyOneSchedule...feature/cerati_EnsureOnlyOneScheduleV4

Multithreading implementation in modules

- Significant work on multiple fronts in the past years, time to harvest it
  - hit finder [paper], icarus signal processing, Wire Cell

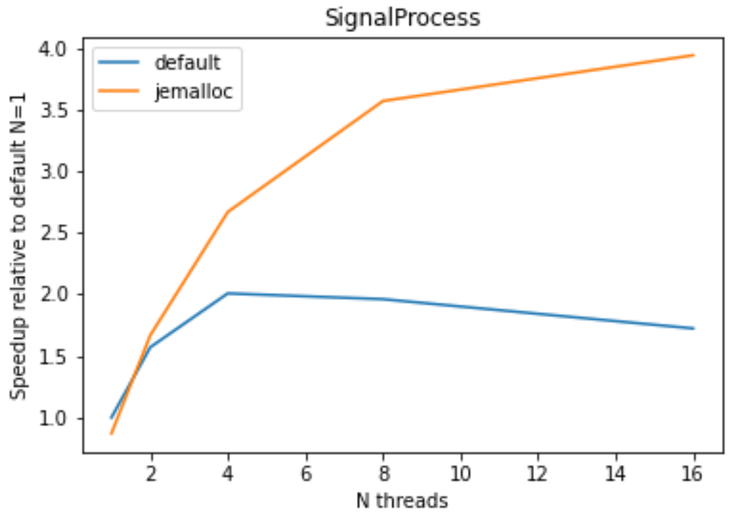
- Icarus stage0_run2_icarus_mc.fcl has basically the following steps multithreaded with TBB already:
  - MCDecoderICARUSTPCwROI, Decon1DROI, ROIFinder, GausHitFinder
  - Although with some updates to Decoder and ROIFinder

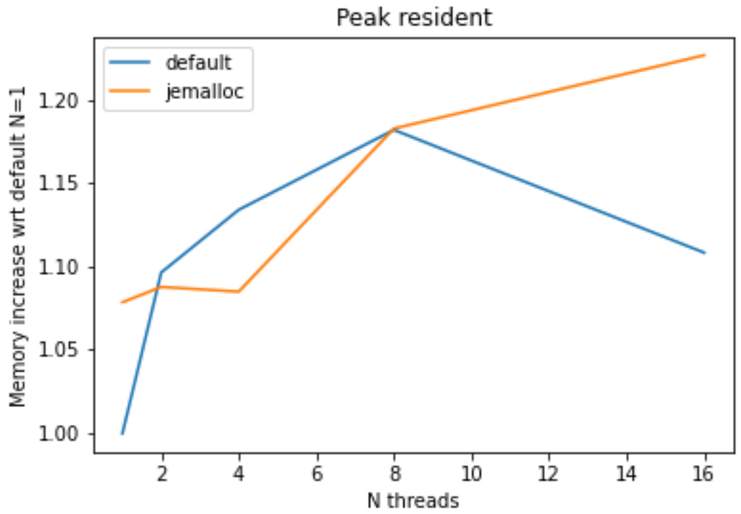
- Next I show results from this setup:
  - icaruscode v09_60_00+branches, running stage0_run2_icarus_mc.fcl
    - tested without (default) and with jemalloc library for memory allocations
  - not meant to be a “final” version, goal is to motivate work to get there
    - both for 1D and 2D deconvolution processing

Scaling results

Out of the box, not necessarily optimized/tuned.

- Tested on icarusbuild02, without other ongoing jobs
  - not a production environment
- Can achieve up to 4x speedup for the 4 modules that are multithreaded
- FullEvent stage0 processing speedup limited by other time consuming modules
- Memory increase is overall small, as expected

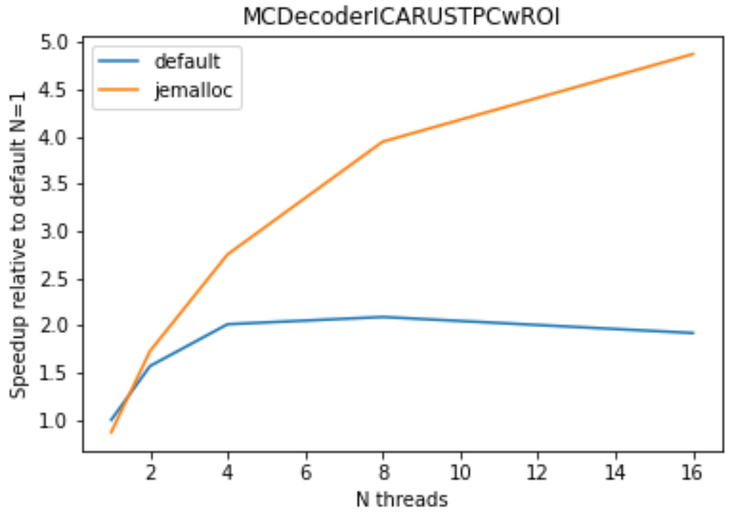


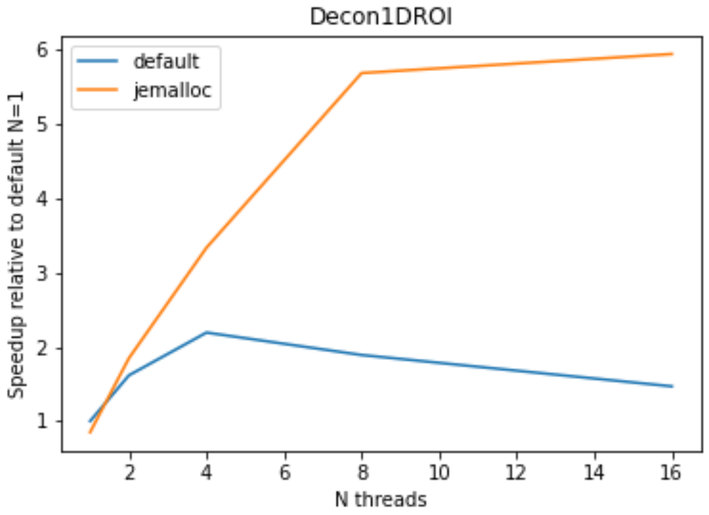


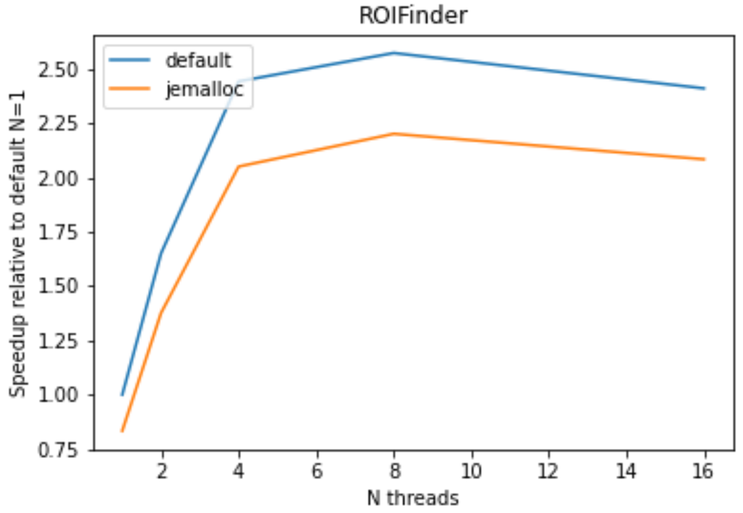
Fermilab

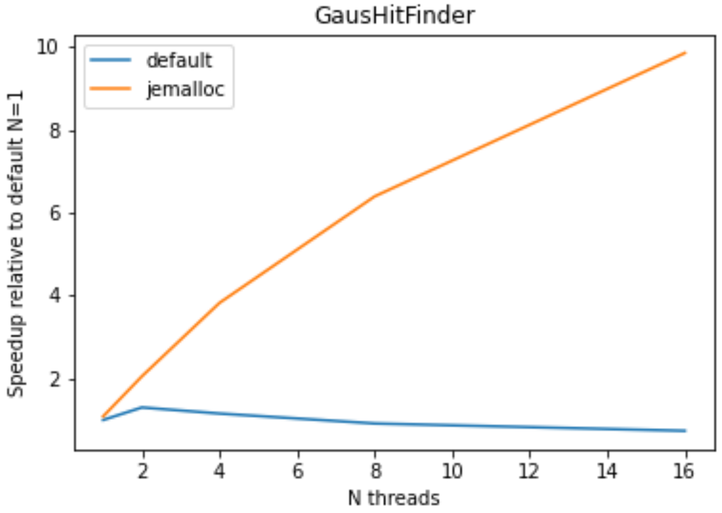
Scaling of individual modules

Out of the box, not necessarily optimized/tuned.









Conclusions

- Short-term path for SHARED services allowing for multithreading within event
  - work for SciSoft team still needed for full thread safety across schedules
- Tests on 1D deconv. signal processing (up to hit finder) give speedups ≤4x
  - actual speedups need to be measured in production environment
  - use of jemalloc is typically beneficial for multithreading
- Defining services as SHARED is critical for moving forward, input welcome!
  - hope to proceed as early as this week if any of the versions presented here is acceptable
- Thanks to Tracy Usher, Erica Snider, Kyle Knoepfel for useful discussions!

Backup