Cornell University

Fermilab

Multi-threaded Output in CMS using ROOT

CHEP2018

Dan Riley (Cornell) & Chris Jones* (FNAL)

Output in CMS using ROOT

About ROOT I/O

- **ROOT streams C++ objects into files using a columnar data format with compression**
  - Involves both serialization and compression of the data
  - A column can be an entire C++ object or a subfield
- **ROOT accumulates data in per-column buffers that it compresses and flushes to disk at regular intervals**
  - The buffer sizes and flush frequency are automatically tuned using a target for the buffer size
- **Originally single-threaded, recently acquiring more multi-threaded capability**
  - In the process of migrating from a “big lock” architecture to more fine-grained locking

CMS multi-threading and ROOT output

- **CMS has been aggressively working on scaling jobs to efficiently use multiple cores**
  - CMS production jobs routinely use 4 or 8 cores
- **ROOT output is currently our main obstacle to scaling beyond 8 cores**
  - Only one thread at a time can write to a ROOT file
  - Bottlenecks are mainly in serialization and compression (CPU), not disk I/O

CMS data tiers

CMS has several different data tiers with varying event content and IO characteristics:

- **Analysis formats: AOD/MINIAOD**
  - Relatively small data volumes, infrequent flushes (AOD: 100 events, MINIAOD: 1000 events), expensive compression (LZMA), many data columns
  - Compression is the main bottleneck
- **Full reconstruction: RECO**
  - Large data volume, frequent flushes (every ~10 events), fast compression (zlib), many data columns
  - Both serialization and compression limit scalability
- **Simulated data: GENSIM**
  - Moderate data volume, moderate flush frequency (every ~100 events), expensive compression (LZMA), relatively few data columns, columns have very different sizes
  - Compression is the main bottleneck

ROOT I/O multithreading

ROOT has recently added two tools for addressing I/O bottlenecks*:

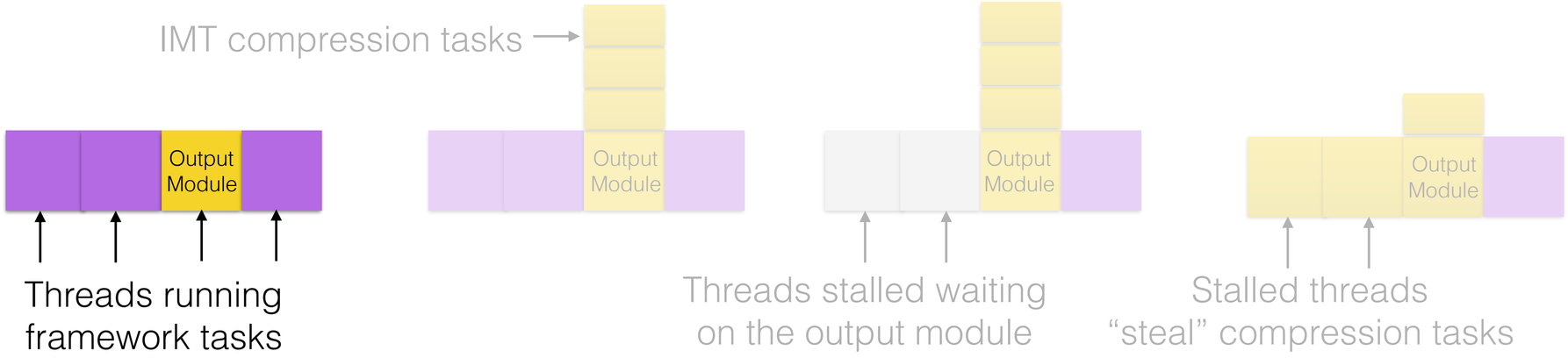
- **Implicit Multi-Threading (IMT, in ROOT 6.08+)**
  - IMT parallelizes data serialization and compression of the per-column buffers
  - Helps most when there are many data columns with expensive compression
  - Nearly a “free lunch”, as no development work is needed to enable it, but there can be unexpected interactions from sharing the same task pool
- **TBufferMerger (in ROOT 6.10+)**
  - The output file has multiple memory buffers that threads write to
  - When the memory buffer is flushed to disk, it is compressed on the worker thread and then merged to the output file
  - Current production version of TBufferMerger uses an auxiliary thread for the merge operation
  - CMS is using a developmental version that performs the merge on the worker thread so that the merge operation is within the CMS framework’s CPU scheduling

* Increasing Parallelism in the ROOT I/O Subsystem (Amadio, Bockelman, Canal, Piparo, Tejedor, Zhang)

IMT in Schematic Form

IMT takes advantage of threads that would otherwise stall

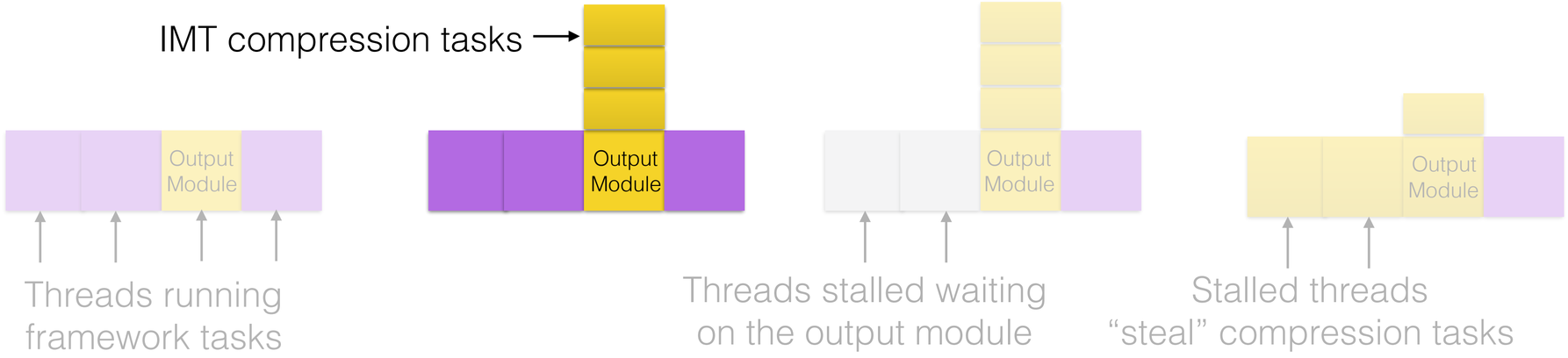
- **IMT creates Threading Building Blocks (TBB) tasks to compress data buffers**
  - Uses the same "task arena" as the CMS framework
- **Tasks are queued on the output module thread's task queue**
- **If another thread has no work on its task queue, it will "steal" work from the output module's queue**



IMT in Schematic Form

IMT takes advantage of threads that would otherwise stall

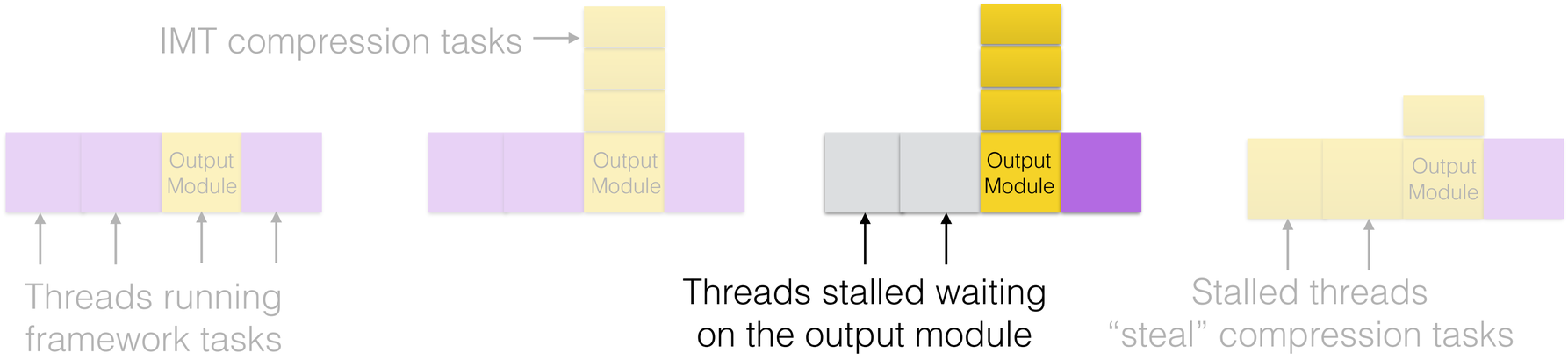
- **IMT creates Threading Building Blocks (TBB) tasks to compress data buffers**
  - Uses the same "task arena" as the CMS framework
- **Tasks are queued on the output module thread's task queue**
- **If another thread has no work on its task queue, it will "steal" work from the output module's queue**



IMT in Schematic Form

IMT takes advantage of threads that would otherwise stall

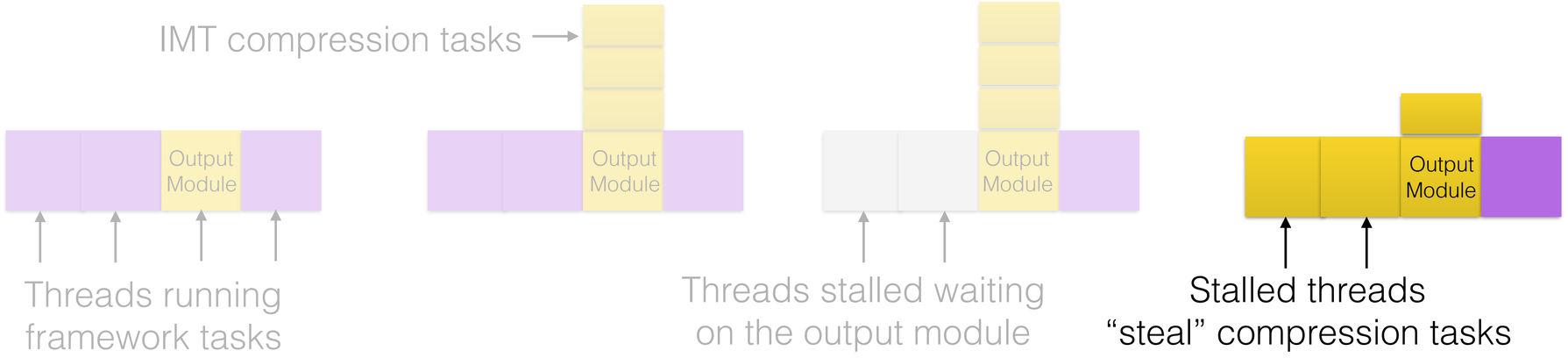
- **IMT creates Threading Building Blocks (TBB) tasks to compress data buffers**
  - Uses the same "task arena" as the CMS framework
- **Tasks are queued on the output module thread's task queue**
- **If another thread has no work on its task queue, it will "steal" work from the output module's queue**



IMT in Schematic Form

IMT takes advantage of threads that would otherwise stall

- **IMT creates Threading Building Blocks (TBB) tasks to compress data buffers**
  - Uses the same "task arena" as the CMS framework
- **Tasks are queued on the output module thread's task queue**
- **If another thread has no work on its task queue, it will "steal" work from the output module's queue**



CMS parallel output module

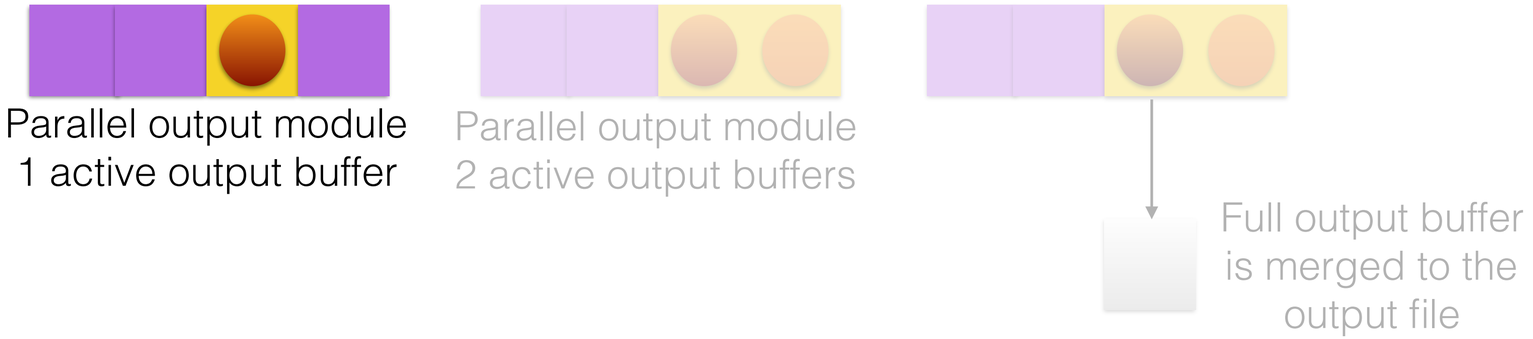
CMS parallel output module use the ROOT TBufferMerger to further parallelize I/O

- **Allocating a TBufferMerger buffer for every thread would be wasteful and inefficient**
  - ROOT data buffers would require too much memory or be too small for good compression
  - Contention from synchronization effects where multiple threads flush buffers to disk at the same time
  - Increases edge effects and partially filled buffers that compress poorly
- **CMS framework added a new module type with limited concurrency**
  - Concurrency is limited to avoid excessive resource allocation

Parallel output module schematic

The parallel output module keeps a pool of TBufferMerger buffers

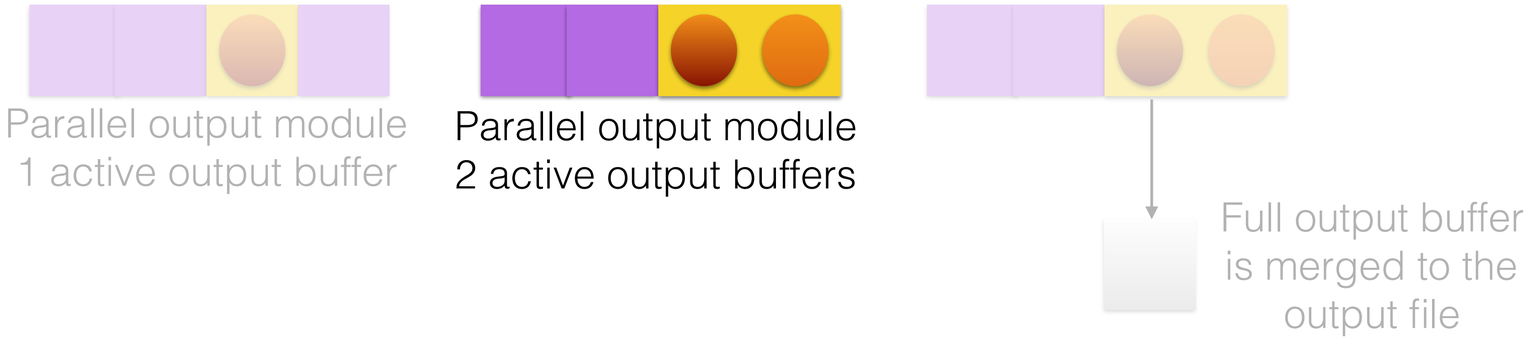
- **Output module has limited concurrency to limit the # of buffers created**
  - CMS framework needs to know about the limit so it can schedule accordingly
- **Always fill the available buffer with the most entries**
  - Avoids synchronization effects, minimizes tail effects, approximates serial ordering
- **Data buffer compression is initiated on the output module's thread**
  - Possibly parallelized by IMT—can lead to non-trivial interactions



Parallel output module schematic

The parallel output module keeps a pool of TBufferMerger buffers

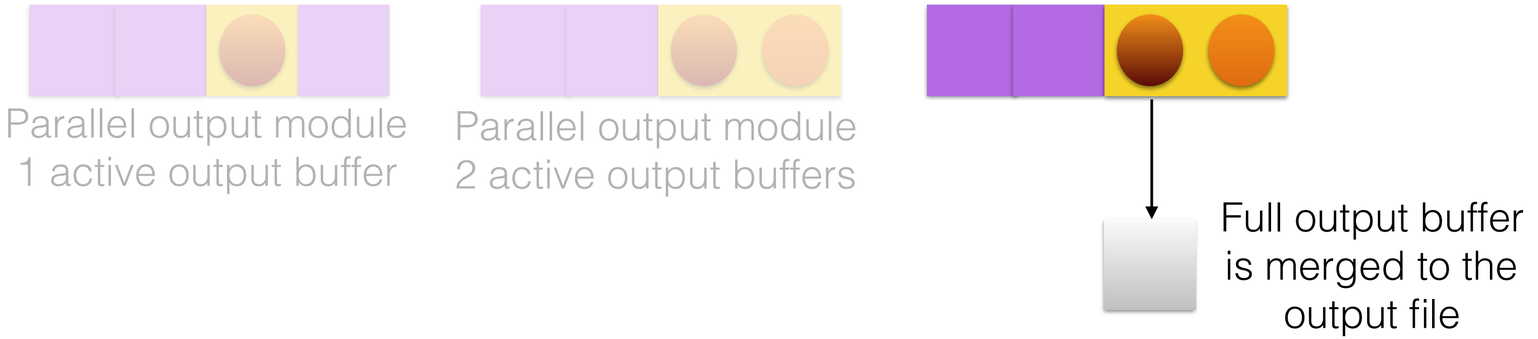
- **Output module has limited concurrency to limit the # of buffers created**
  - CMS framework needs to know about the limit so it can schedule accordingly
- **Always fill the available buffer with the most entries**
  - Avoids synchronization effects, minimizes tail effects, approximates serial ordering
- **Data buffer compression is initiated on the output module's thread**
  - Possibly parallelized by IMT—can lead to non-trivial interactions



Parallel output module schematic

The parallel output module keeps a pool of TBufferMerger buffers

- **Output module has limited concurrency to limit the # of buffers created**
  - CMS framework needs to know about the limit so it can schedule accordingly
- **Always fill the available buffer with the most entries**
  - Avoids synchronization effects, minimizes tail effects, approximates serial ordering
- **Data buffer compression is initiated on the output module's thread**
  - Possibly parallelized by IMT—can lead to non-trivial interactions



Performance Tests

Test configurations:

- **13 GeV TTBar simulated data, LHC run2 conditions with semi-realistic pileup**
- **Full reconstruction step, with two output scenarios**
  - Writing full RECO, AOD and MINIAOD
  - Writing full AOD and MINIAOD (no RECO)
- **Platform:**
  - 32 core Skylake-SP Gold 6130 CPU @ 2.10 GHz
  - 64 core Xeon Phi KNL 7210 @ 1.30GHz

Tests:

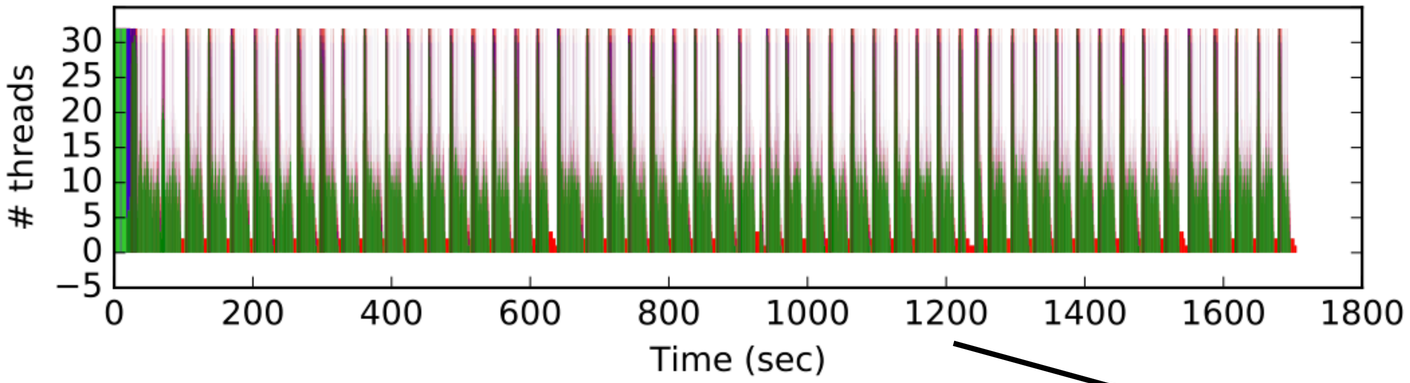
- **Standard output module with and without IMT**
- **Parallel output module with IMT**
  - RECO/AOD/MINIAOD: RECO output concurrency 6, AOD concurrency 6, MINIAOD 3 (6x6x3)
  - AOD/MINIAOD: AOD concurrency 4, MINIAOD 2 (4x2)
- **Dummy output module that produces no output, to test scaling limits**
  - Implemented by an option for the parallel output module to skip filling and writing the ROOT trees

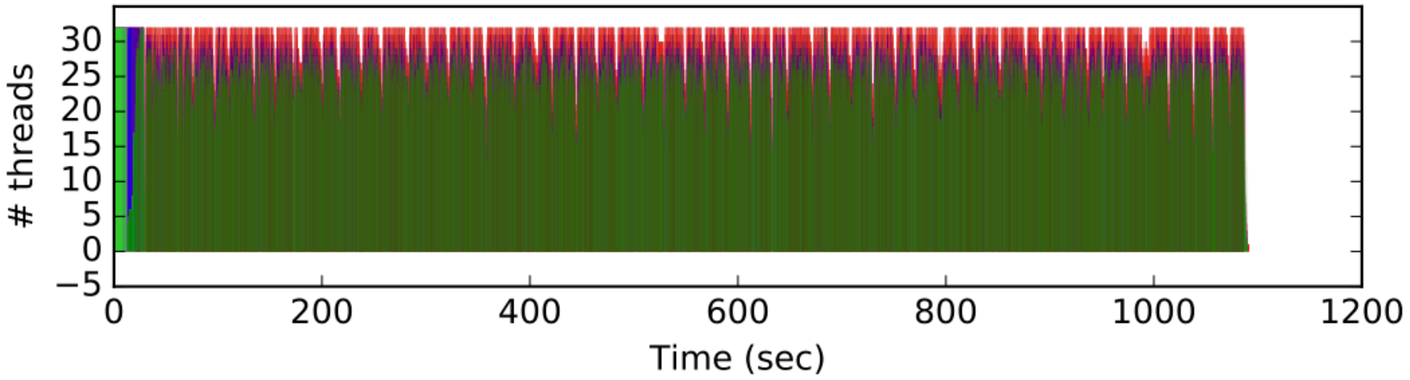
Standard output vs. parallel concurrency

Concurrency plot shows the total number of concurrent modules

- Perfect efficiency when # of modules == # of threads
- Dark green shows concurrent events
- Gaps are inefficiencies

Serial output, No IMT

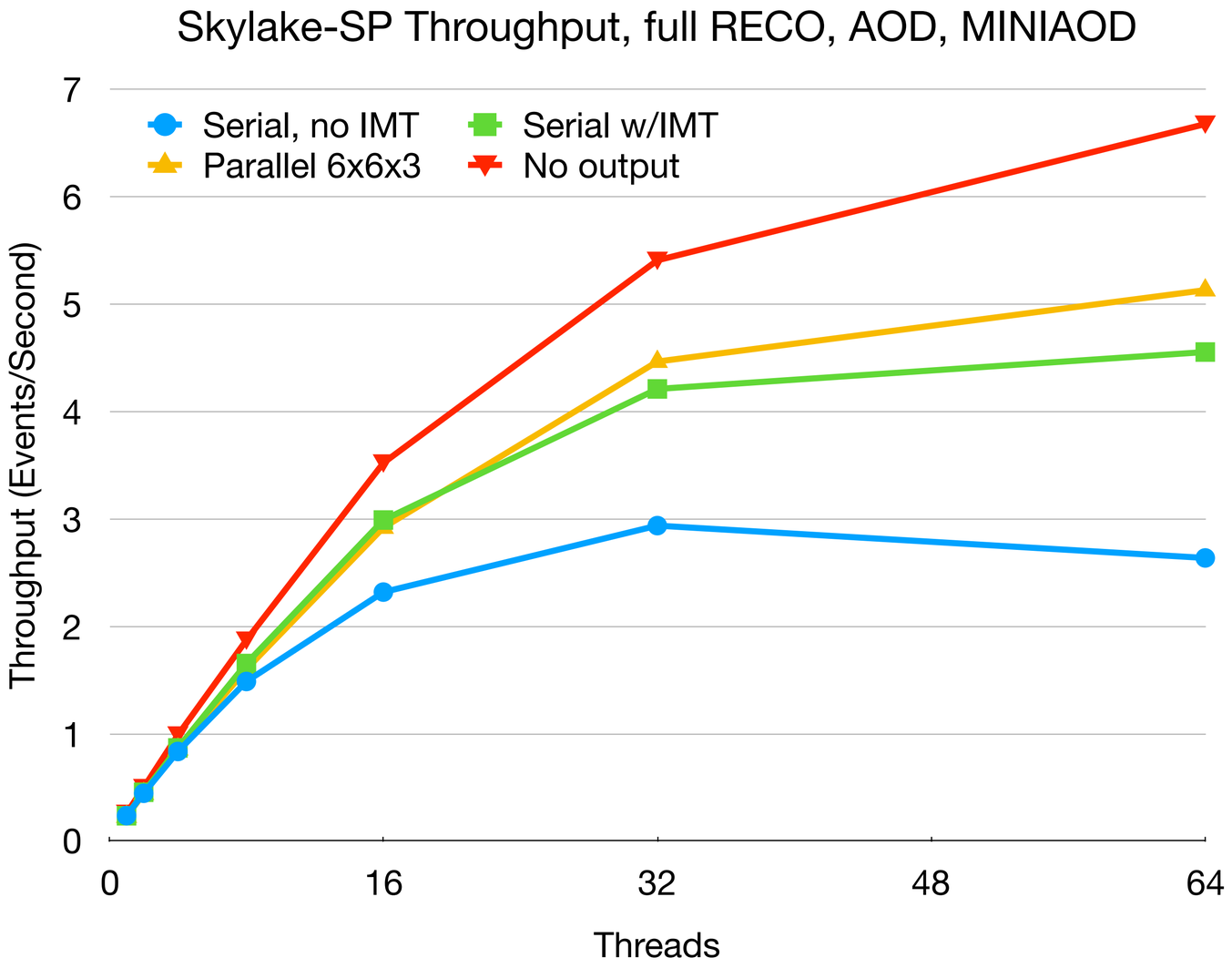




Parallel output with IMT

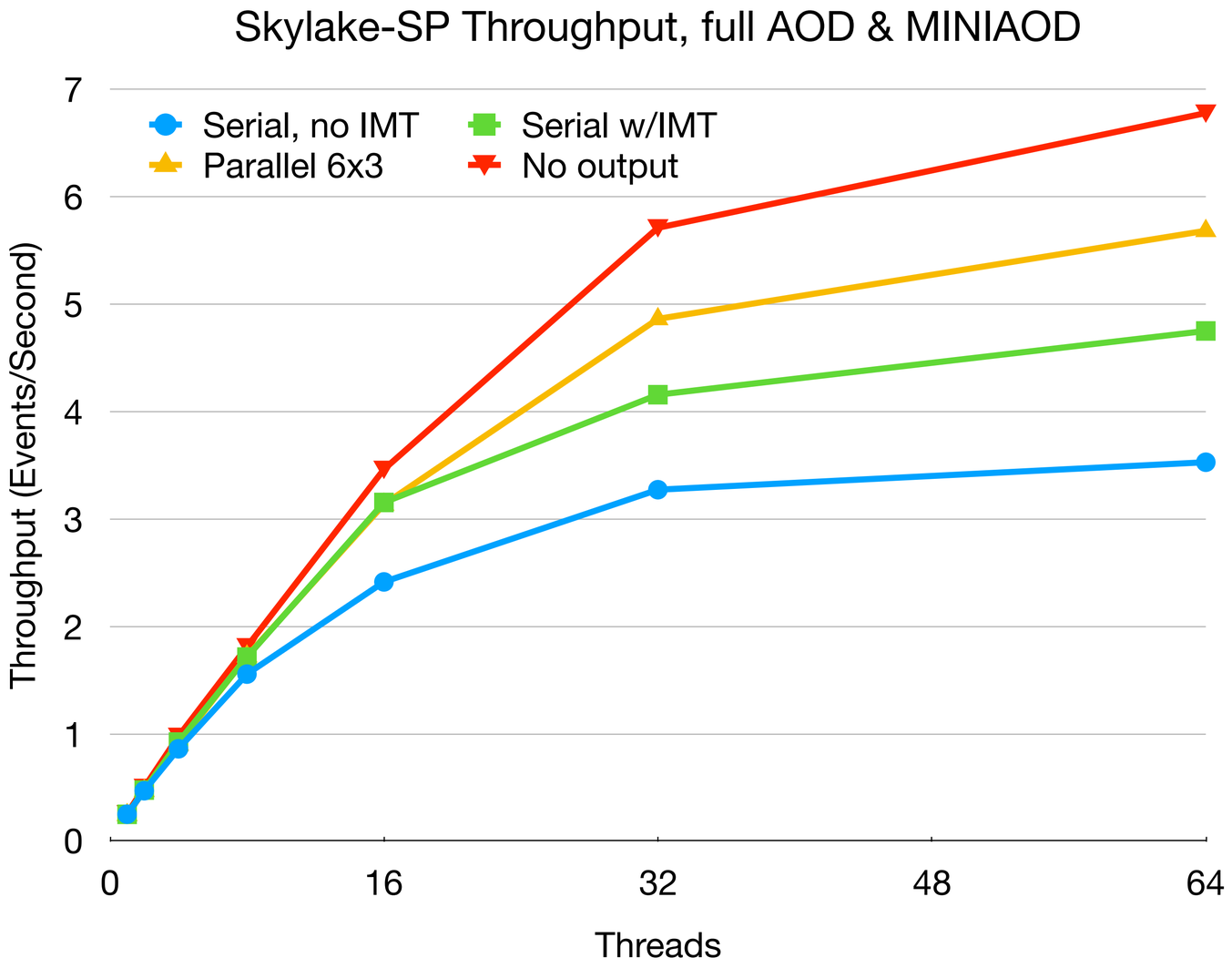
RECO/AOD/MINIAOD Scaling

- IMT alone does well up to 32 threads
- RECO has frequent flushes and more buffers to compress in parallel, so IMT alone does well on RECO
- IMT + parallel output ultimately scales better at high thread count, but not dramatically so



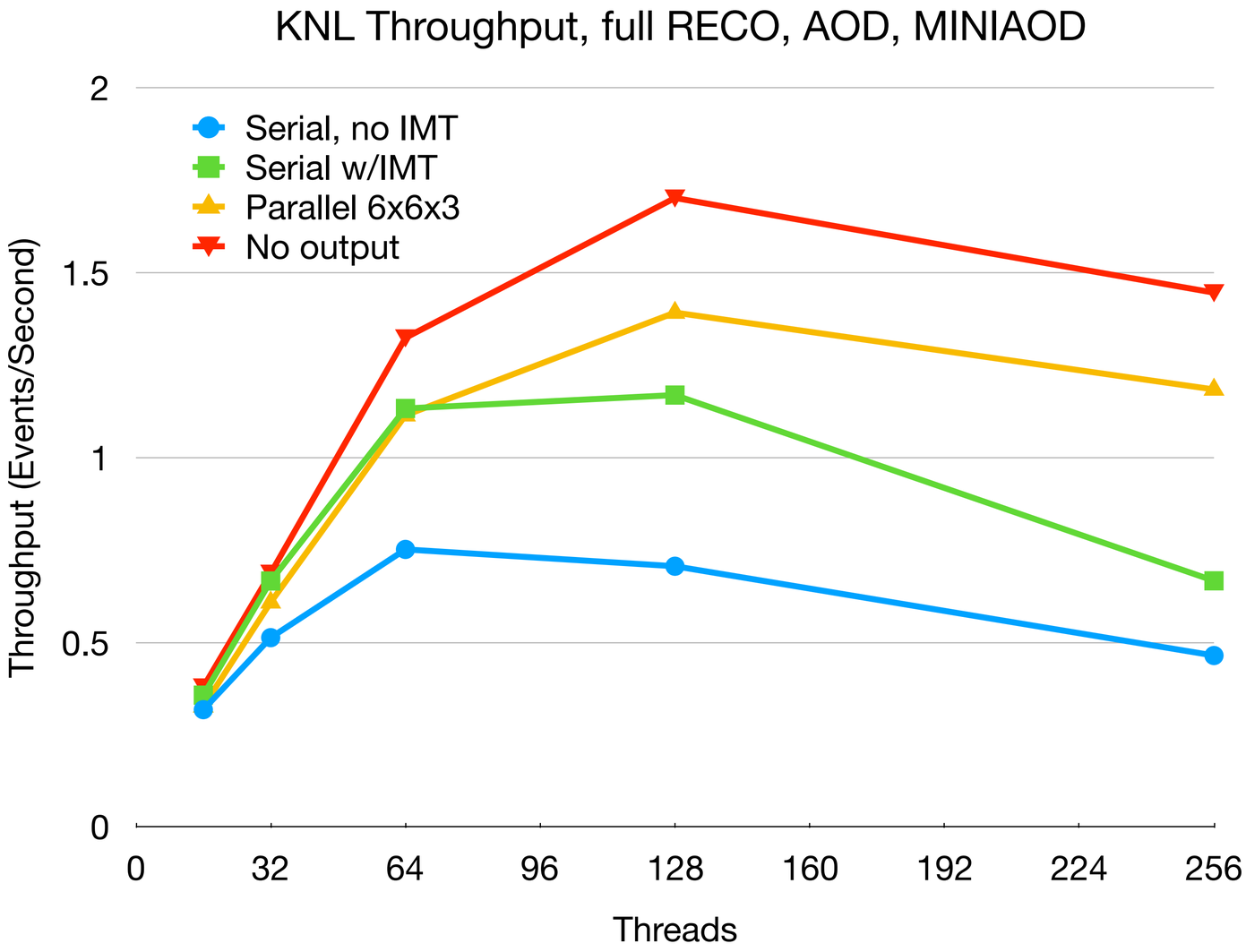
AOD/MINIAOD Scaling

- Parallel output module does substantially better at high thread count
- Some evidence that failure to reach the scaling limits are due to ROOT "big lock" contention
  - ROOT team is working to remove the mutex CMS has identified as a scaling limit



RECO/AOD/MINIAOD on KNL

- IMT alone does well up to 64 threads
- IMT + parallel output continues to improve somewhat up to 128 threads
  - Follows the “no output” scaling, which drops off past the number of physical cores



Conclusions

Recent ROOT output concurrency developments can significantly improve CMS multi-thread scaling

- **IMT is a clear win for CMS**
  - Does better on some data tiers, especially RECO
- **The combination of IMT and the parallel output module using TBufferMerger does better than either alone in most cases**
  - Combined these can dramatically improve output scaling for most (all?) CMS data tiers
  - Scaling should improve further as ROOT's internal concurrency improves
  - But finding the right combination of concurrency levels currently requires some tuning
  - Would like to have either automatic tuning, or a standard set of configurations for concurrency levels

Backup Slides

TBB interactions

Using TBB tasks for IMT can lead to unexpected interactions

- **Example: GEN-SIM production**
  - GEN-SIM has time consuming GEANT simulation tasks
  - Output file has few branches
- **Scenario:**
  - Output module does a TTree::Fill() that results in a flush operation
  - IMT parallelizes the compression of the (small number of) buffers
  - Output module thread gets a relatively small buffer to compress, finishes early, and has to wait for other tasks to finish branch buffer compression
  - Starved for work, output module thread “steals” a GEANT simulation task
  - Output module task is blocked until the GEANT simulation task finishes

Solution/workaround

- `tbb::this_task_arena::isolate( [&]{ tree_->Fill(); } );`
  - Keeps the output module thread “honest” (no task stealing)