Performance Measurements for RNTuple

Dr Christopher Jones
CCE-SOP

17 April 2024

Goal

- Compare throughput and memory usage between RNTuple and TTree

- Need to store the exact same data in both cases

- CMS’s MiniAOD format uses C++ classes RNTuple can not serialize
  - Created a *hacked* MiniAOD format file
    - Copies much of the data to C++ classes RNTuple and TTree can both serialize
    - Data that can not be copied to new classes is dropped (~10% of the data by storage)

Throughput Measurement Methodology

- Compare writing using the following methods
  - RNTupleWriter
  - TTree with un-split top level TBranches
  - TTree with fully split top level TBranches
  - TTree with mixed split of TBranches
- Use LZMA compression level 4
- Use the RepeatingRootSource as input
  - cache in memory 500 Events of data and replay over and over
  - both RNTuple and TTree get data from the *hacked* MiniAOD file
- Run for sufficient amount of time to see scaling
  - Usually 10,000 events * # threads in job which was ~5 minutes

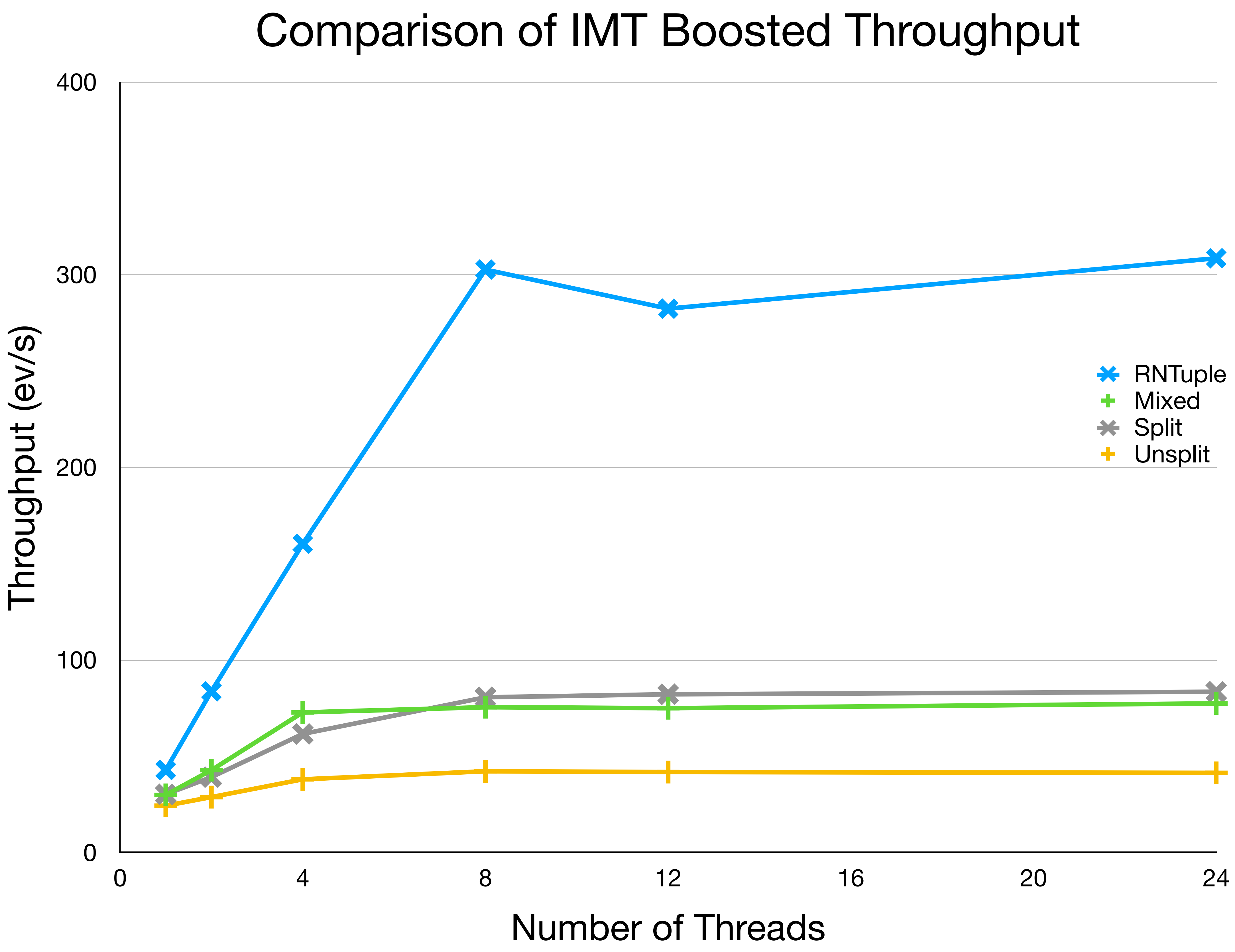
Throughput Measurement Methodology (2)

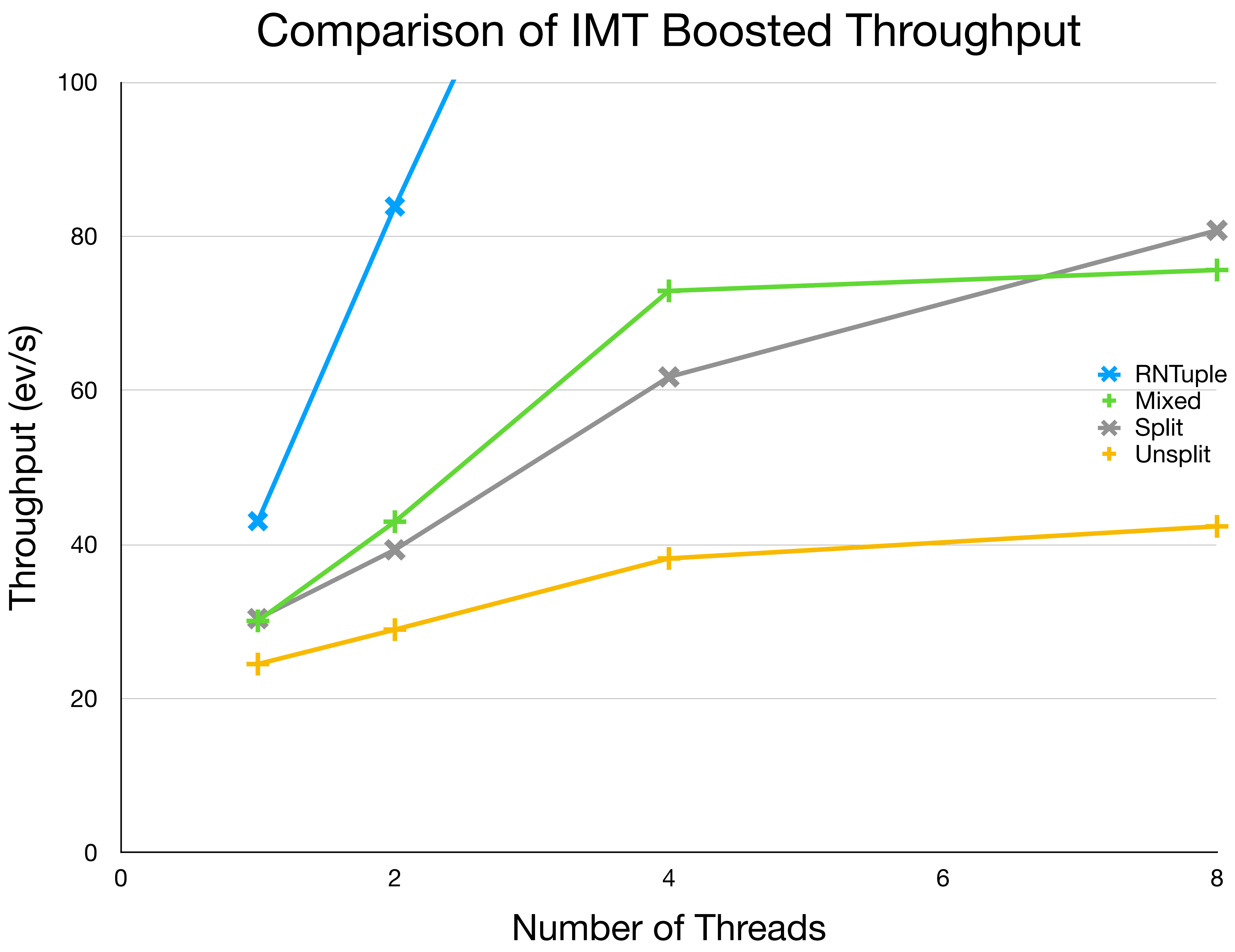
- Fill as much of the machine resources as possible
  - Run as many processes*threads as there are cores on the machine
    - 24 core machine
    - exception, only ran 12 , concurrent single threaded jobs
      - for 1 thread jobs, RNTuple filled the machine memory at just 12 jobs

- Use 1 concurrent Event in the job
  - Scaling comes from ROOT's Implicit Multi-threading

- For TTree, used CMS standard configuration
  - Auto flush buffers every 900 events

Throughput Results





RNTuple scales perfectly up to 8 threads

TTree has much weaker scaling

ParallelRNTupleWriter

- RNTuple interface to allow concurrent writing of entries to an RNTuple
  - Just synchronizes when need to write buffer out to the file
  - Does not appear to use ROOT's implicit multi-threading

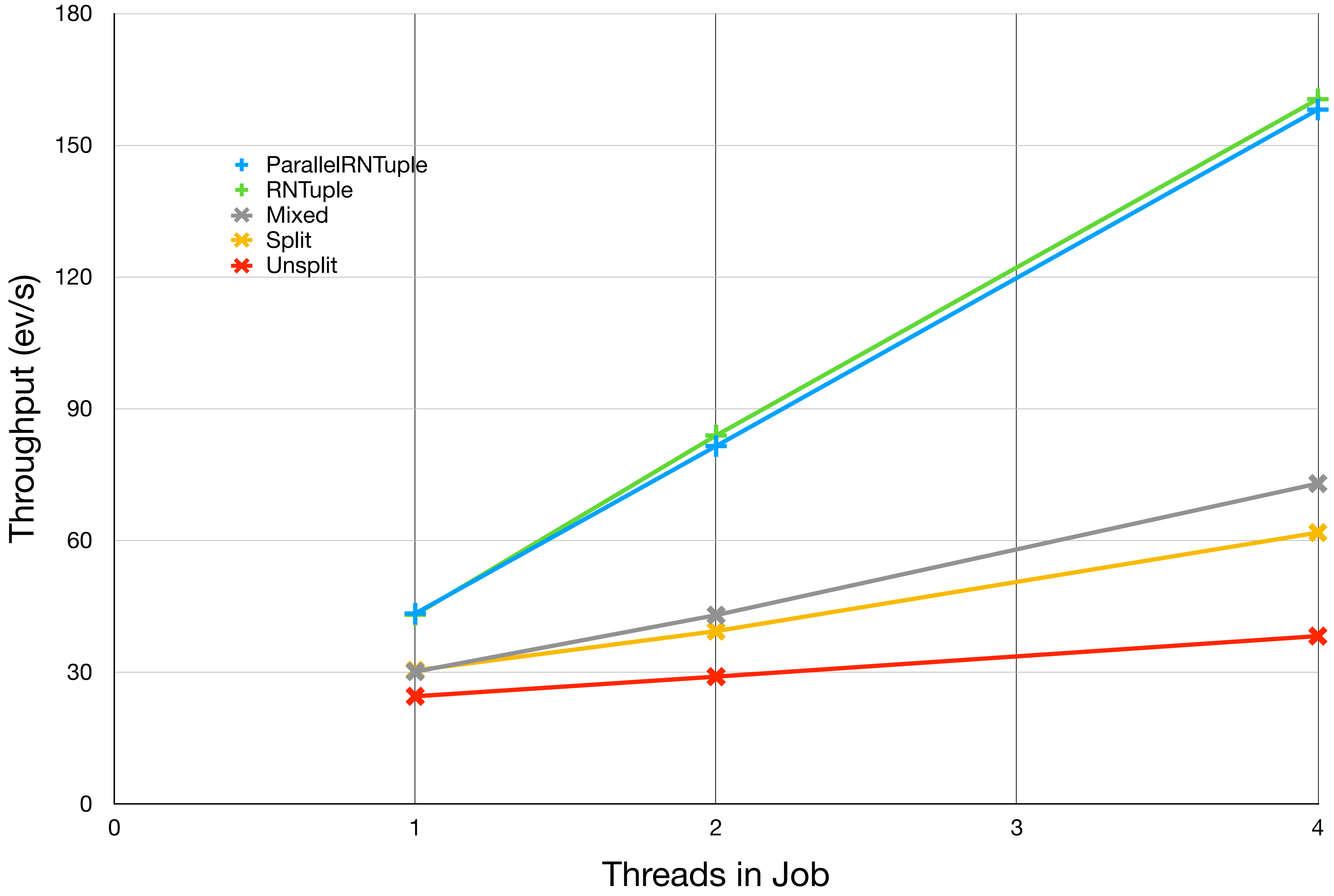
- ParallelRNTupleOutputer
  - testing framework outputted that uses ParallelRNTupleOutputer
  - uses a ParallelRNTupleWriter per Lane (i.e. event loop)

Throughput Measurement Methodology (3)

- ParallelRNTupleOutputer needs lots of memory
  - Could not run 8 or more threaded jobs on the machine I was testing as ran over VSize limitations (16GB)

- Had to restrict number of concurrently running jobs to avoid swapping
  - 10 concurrent jobs for 1 or 2 threaded jobs
  - 5 concurrent jobs for 4 threaded job

Throughput Comparison with ParallelRNTupleOutputer



ParallelRNTupleWriter nearly as scalable as RNTupleWriter using IMT

Memory Measurement Methodology

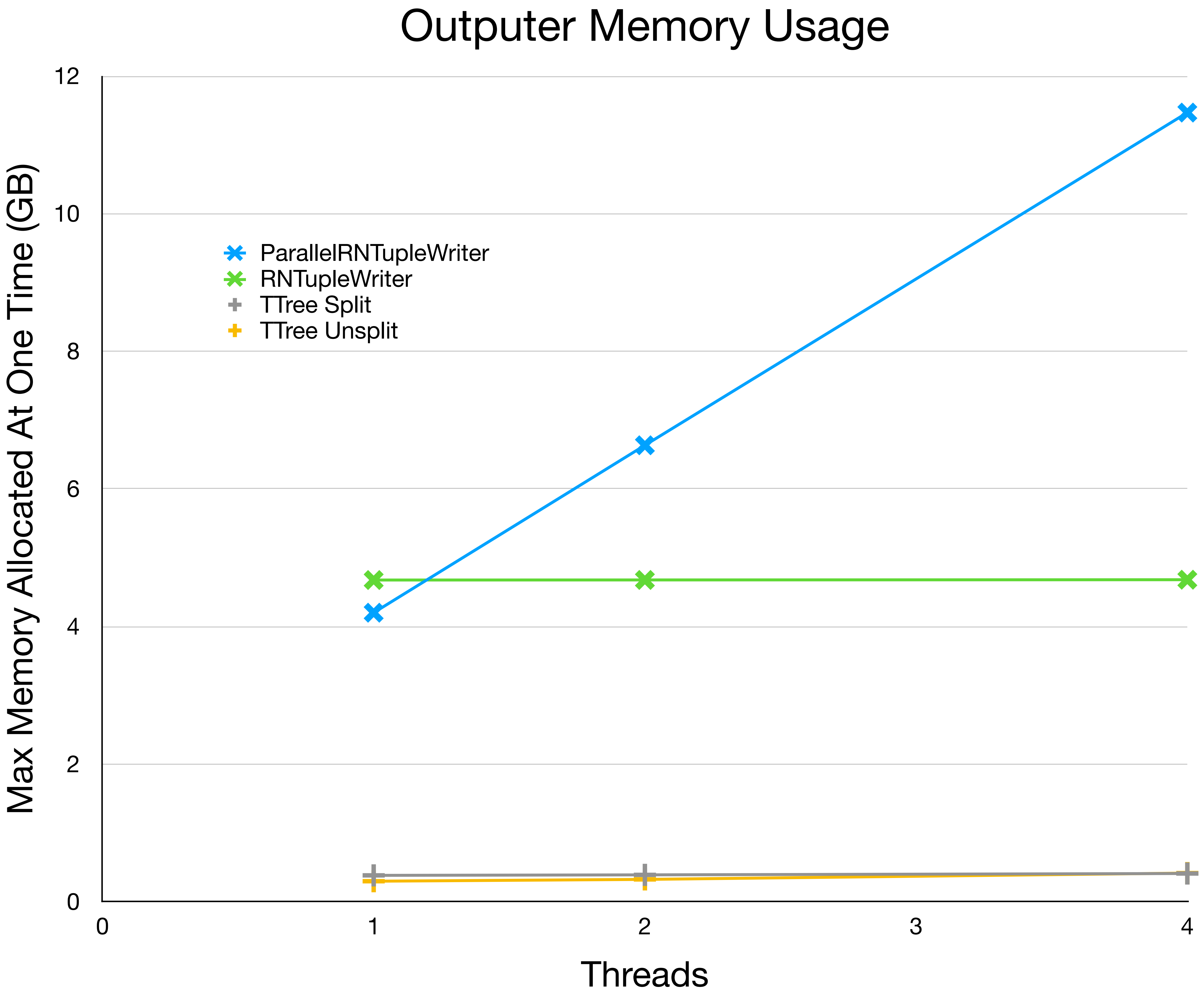
- Used CMS’ allocation measurement library
  - Records information about each new/malloc call
  - E.g. records to number of bytes requested by all allocations

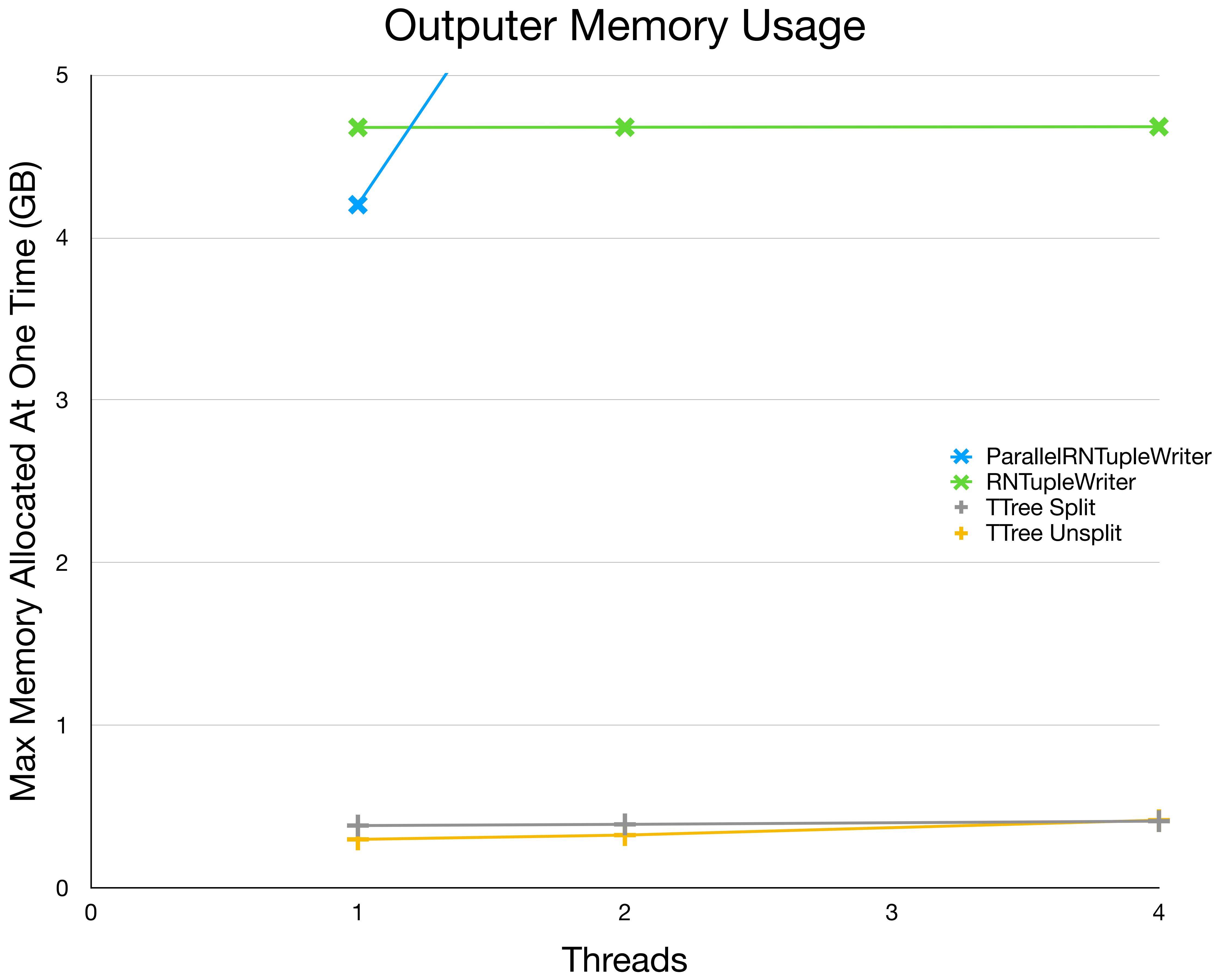
- Used same job set as the throughput measurement

- Ran jobs without any Outputted
  - Used as baseline memory

- Plots on next page subtract the baseline memory to try to find just memory used by the Outputers

Memory Comparisons





ParallelRTupleOutputer requires 2.4GB per thread

RNTuple requires ~ 10x more memory than TTree

This is allocation requested, actually max RSS size is much smaller