Thread Scaling of Different Output Methods

Dr Christopher Jones
CCE - IOS

21/10/2020

Goal

- Use realistic CMS data files

- Measure thread scaling of writing those files
  - Write ROOT format
  - Write a simple format

CMS Data Formats

- CMS Uses several different ROOT based data formats
  - Formats differ by exactly what data products are stored

- RECO
  - ~ 3MB/event
  - ~ 20% of data as it is taken is written to this format
  - most files not kept beyond 90 days

- AOD
  - ~500 kB/event
  - ‘Big’ analysis format
  - data and MC are stored in this format

- miniAOD
  - ~ 50 kB/event
  - ‘medium’ analysis format
  - Used for most analysis

Measurements

- Machine Used
  - AMD Opteron(tm) Processor 6128
  - 4 CPUs with 8 Cores per CPU

- Testing procedure
  - Number of Events processed in a job is directly proportionally to number threads used
    - Exception is when jobs stop scaling with threads, then fix number events processed
  - Unless otherwise noted, number of concurrent Events == number of threads
  - Machine was always fully loaded
    - #threads per job * # concurrently running jobs == 32

- Read first 10 events from the file and replay objects over and over
  - No dependency on storage device read speeds on measurements

- No file actually written
  - Output goes to /dev/null
  - Avoids dependency on speed of storage device in measurement

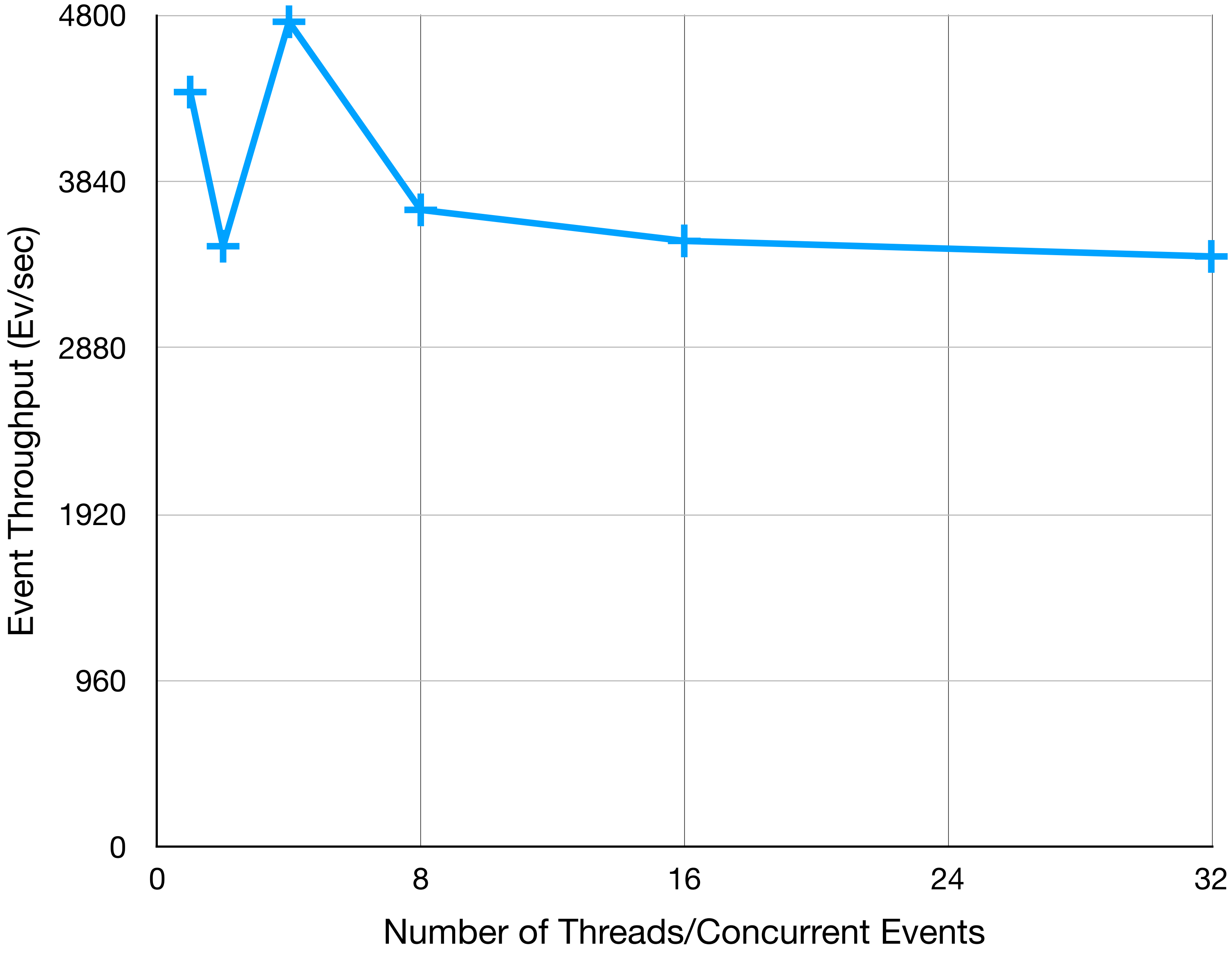
RECO Format

RECO: Data Product Reading Only

- Upper limit on processing for the testing framework using this file

- No scaling

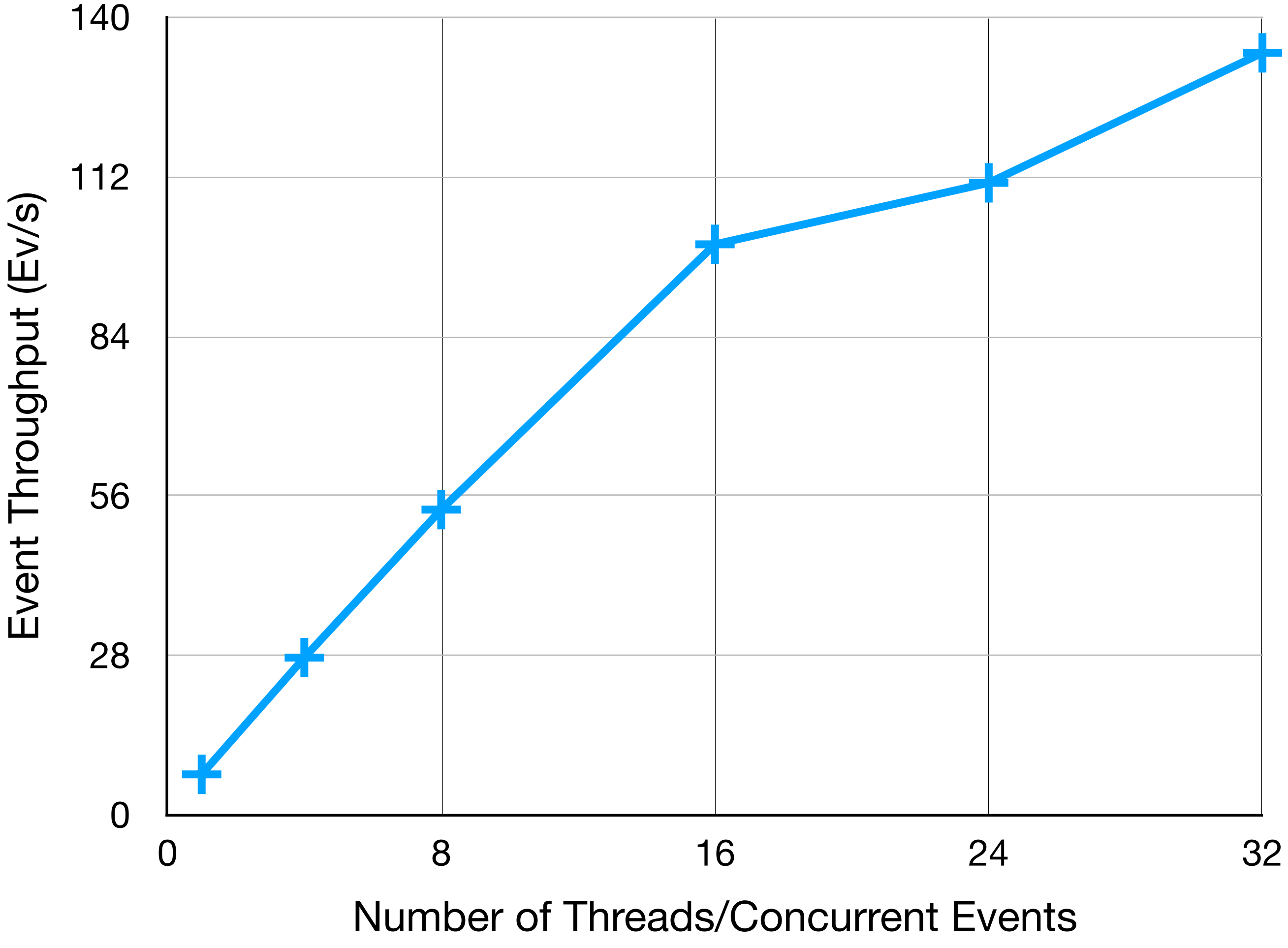
  Standard CMS processing rate is 0.1 Ev/sec/thread

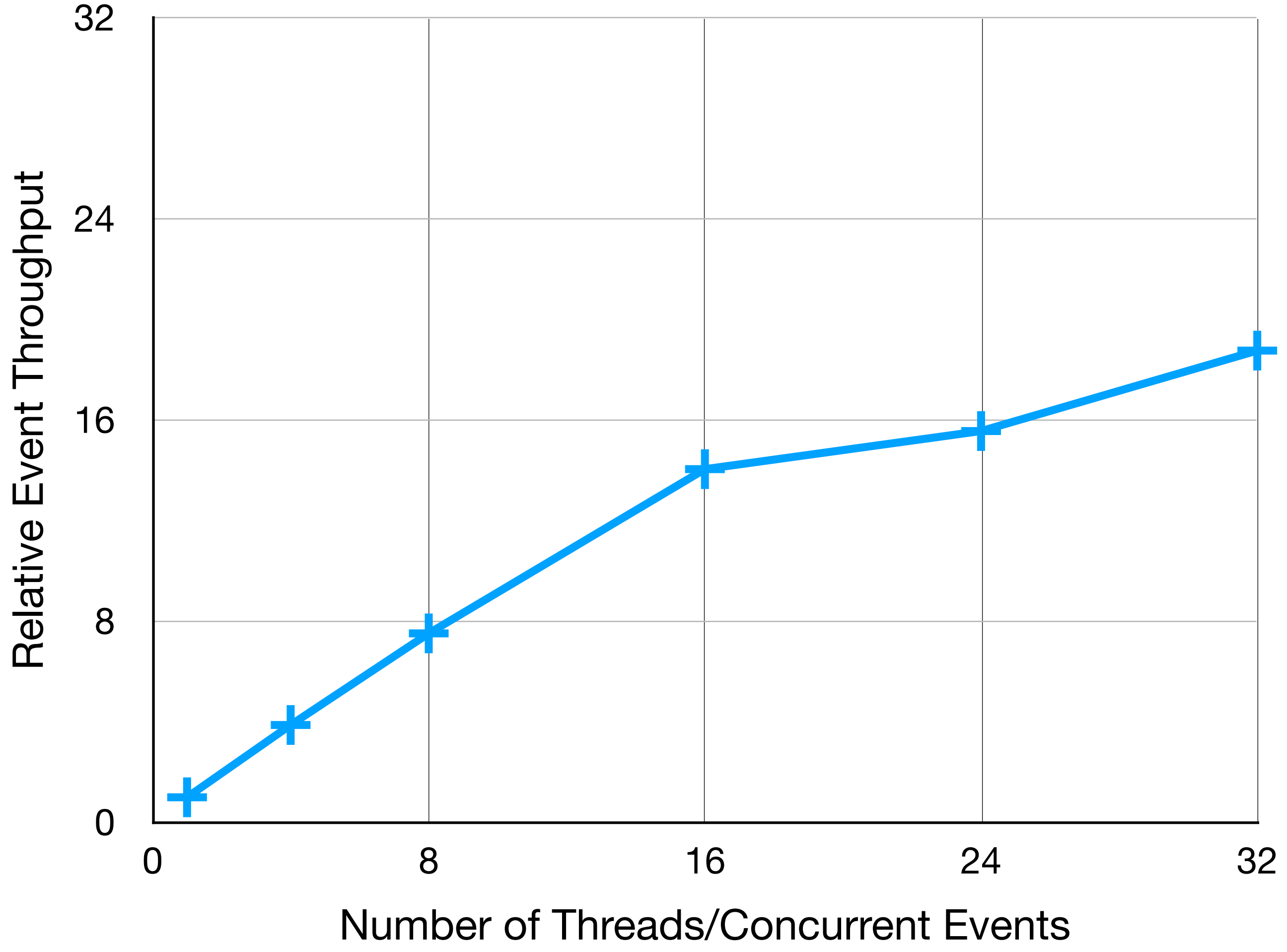


RECO: Use ROOT Serialization

- Serialize the data products read from the file
  - Each data product can be serialized simultaneously
  - Events are processed simultaneously

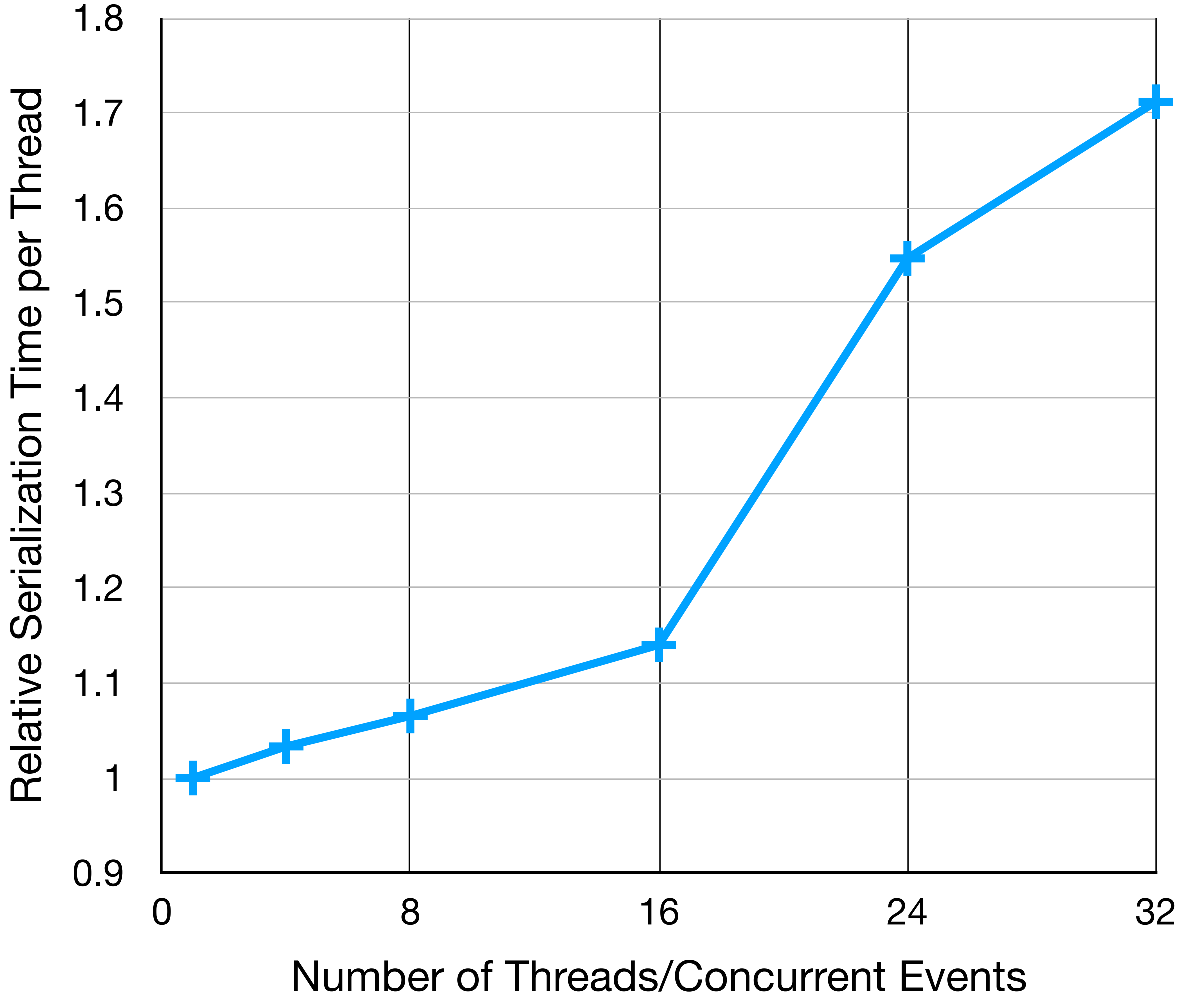
- Good scaling up to 8 threads
  - breaks down around 16 threads

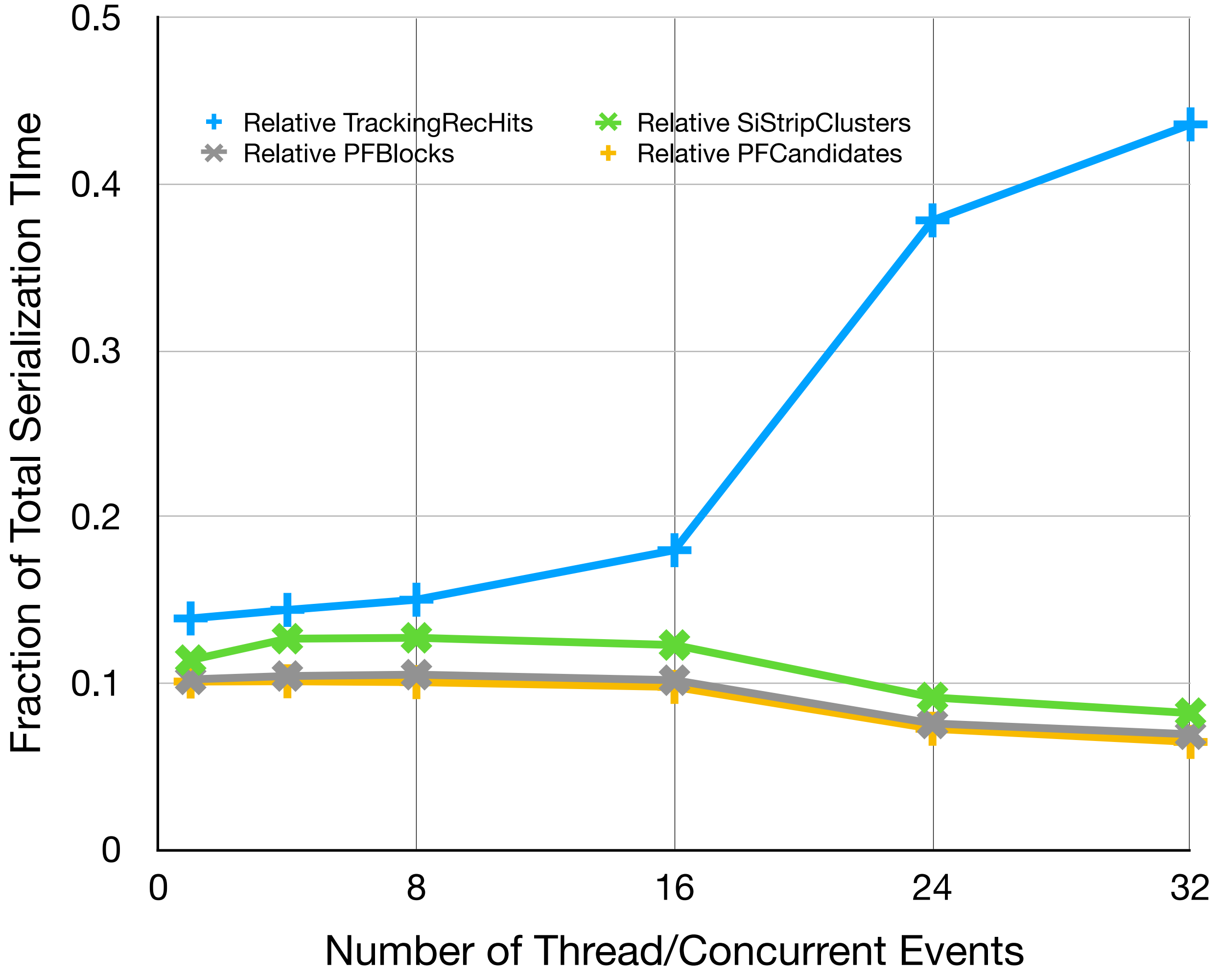




RECO: Use ROOT Serialization (continued)

- ~50% of serialization time comes from 4 data products
- Thread scaling difficulties comes from 1 of the data products not scaling well





Summary of ROOT File Writing

- ROOT TFile API requires only 1 thread to call at a time
  - NOTE: can call methods of different TFiles concurrently

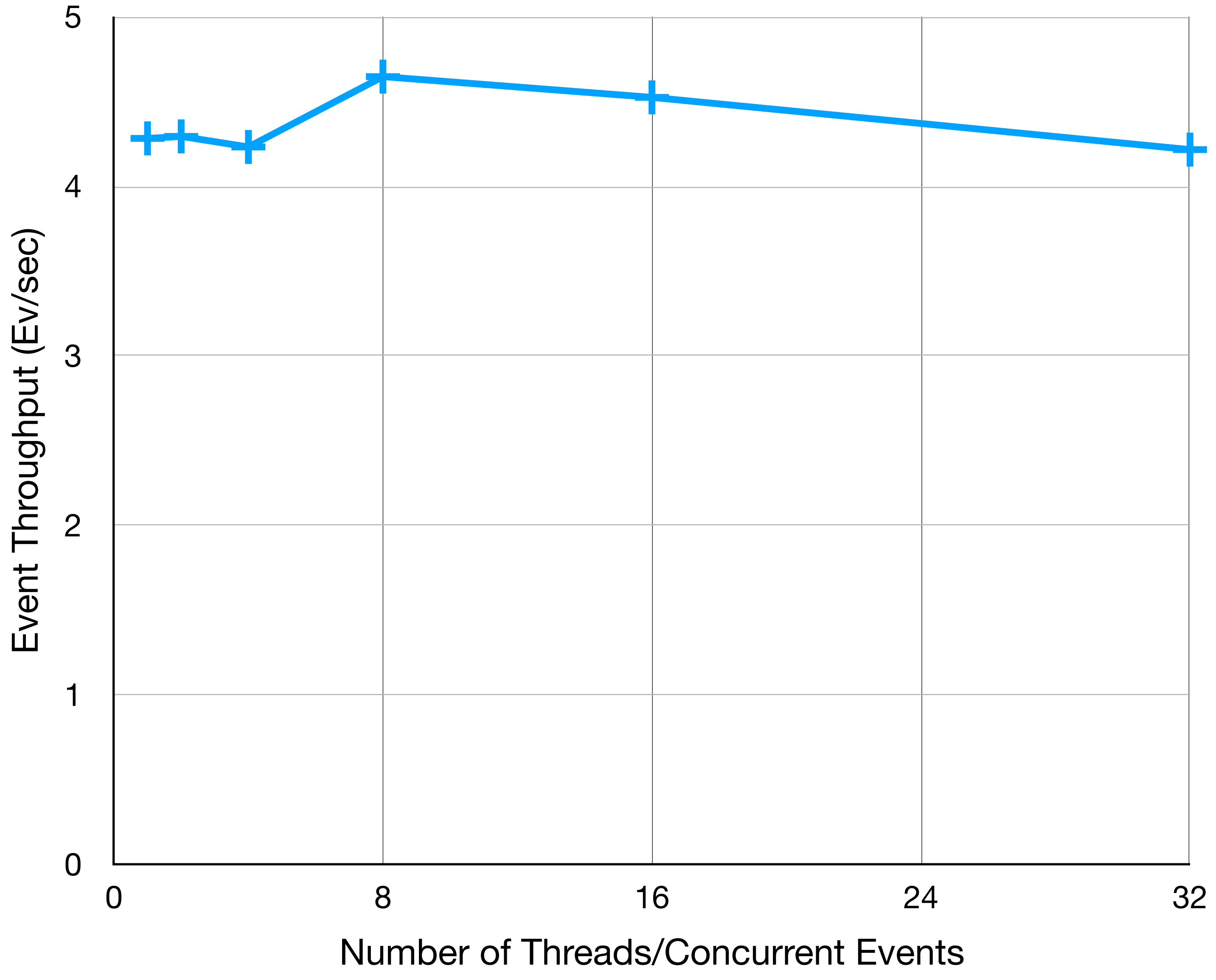
- ROOT can internally use threads to work on files
  - Called *Implicit Multi-Threading* or **IMT**

- Writing a ROOT file can compress different data product buffers concurrently
  - serialization of the C++ objects is still done sequentially

RECO: Write ROOT File Fast

- Write to /dev/null
- Disable use of compression

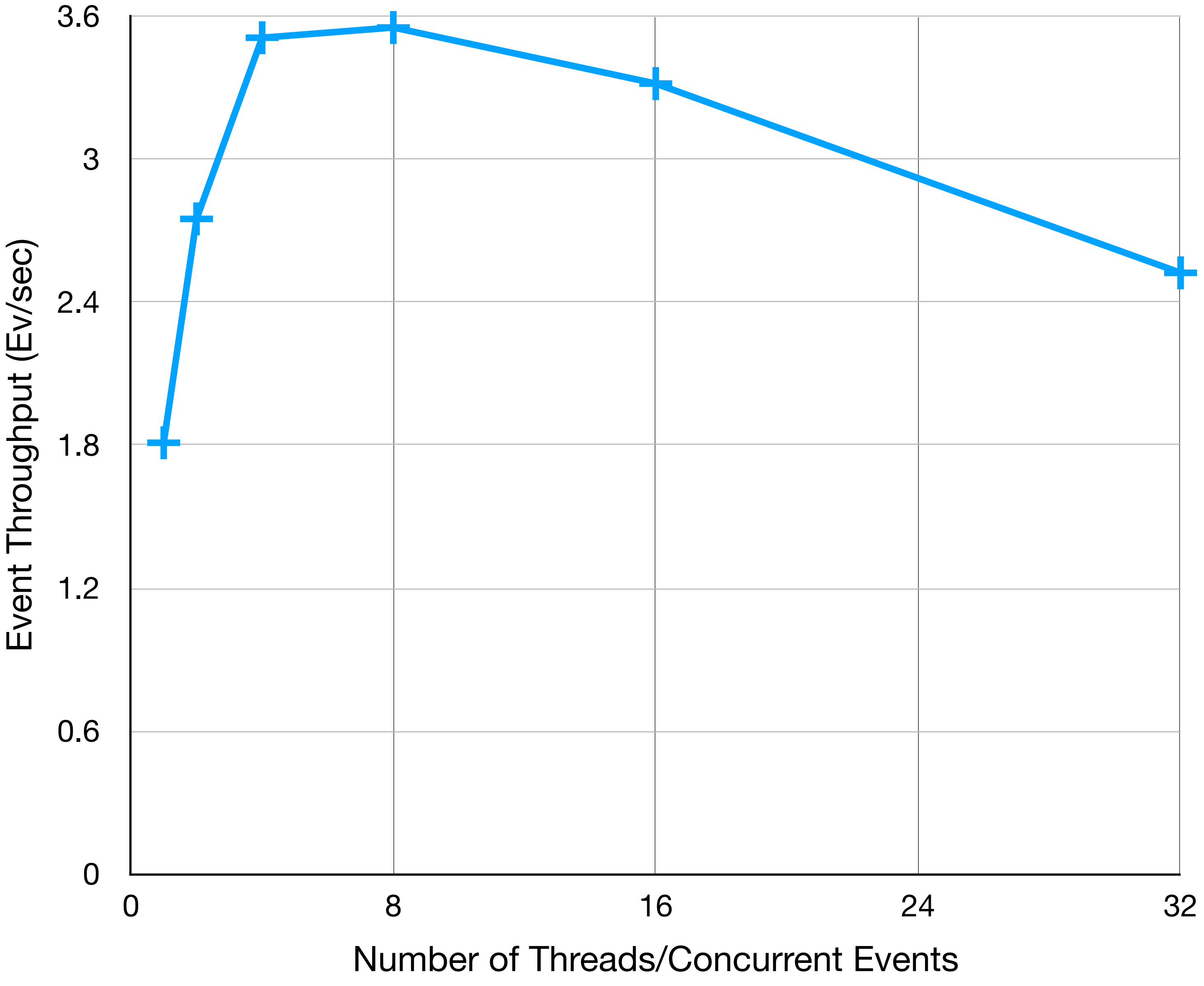
- No scaling
  - This was expected as no IMT used



RECO: Write ROOT File ‘Realistic’

- Write to /dev/null
- Use LZ4 compression
- Use IMT

- See modest scaling
  - not enough parallelization opportunities in data product compression



A Simple Data Format

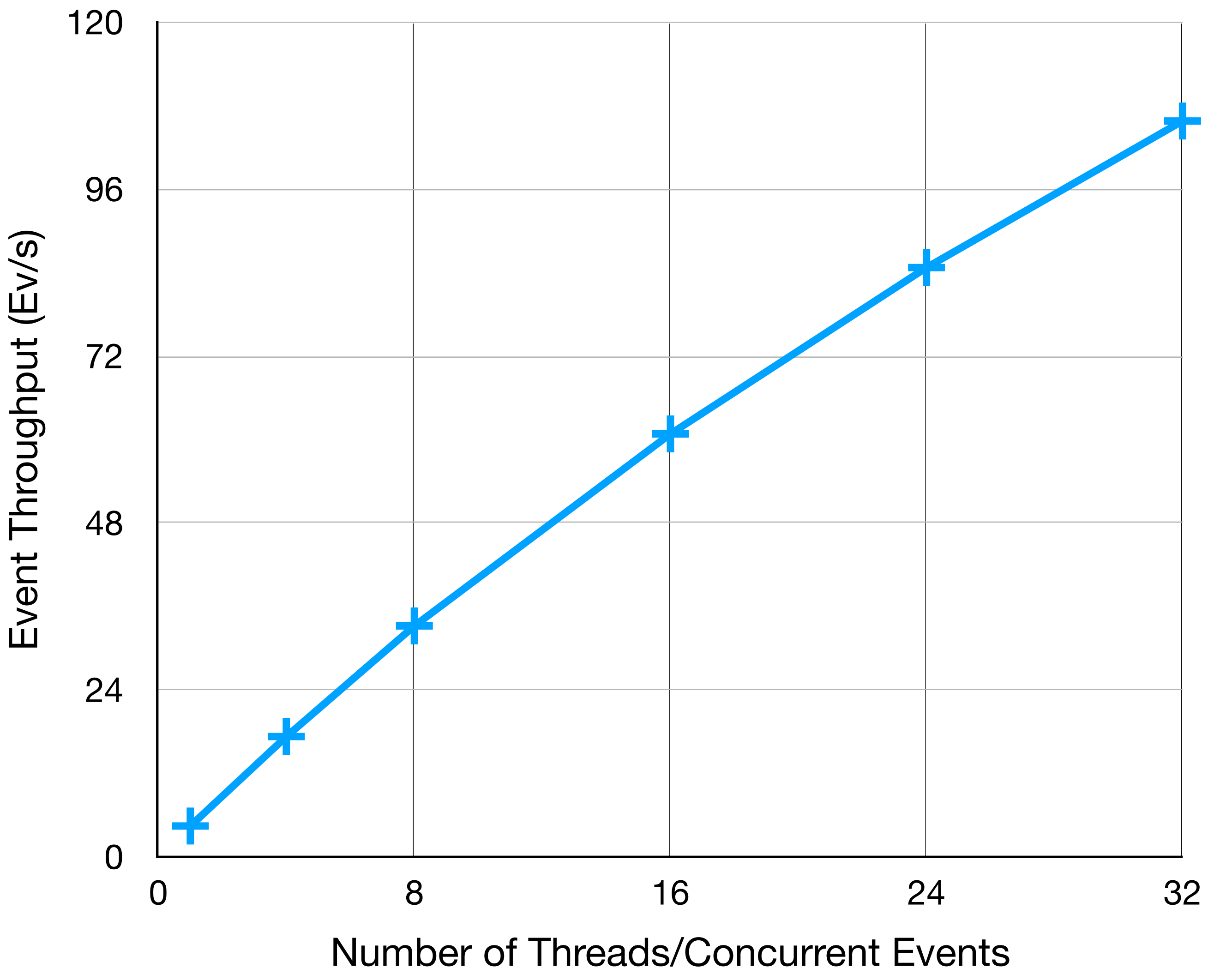
- Repurposed a file data format used by CLEO collaboration

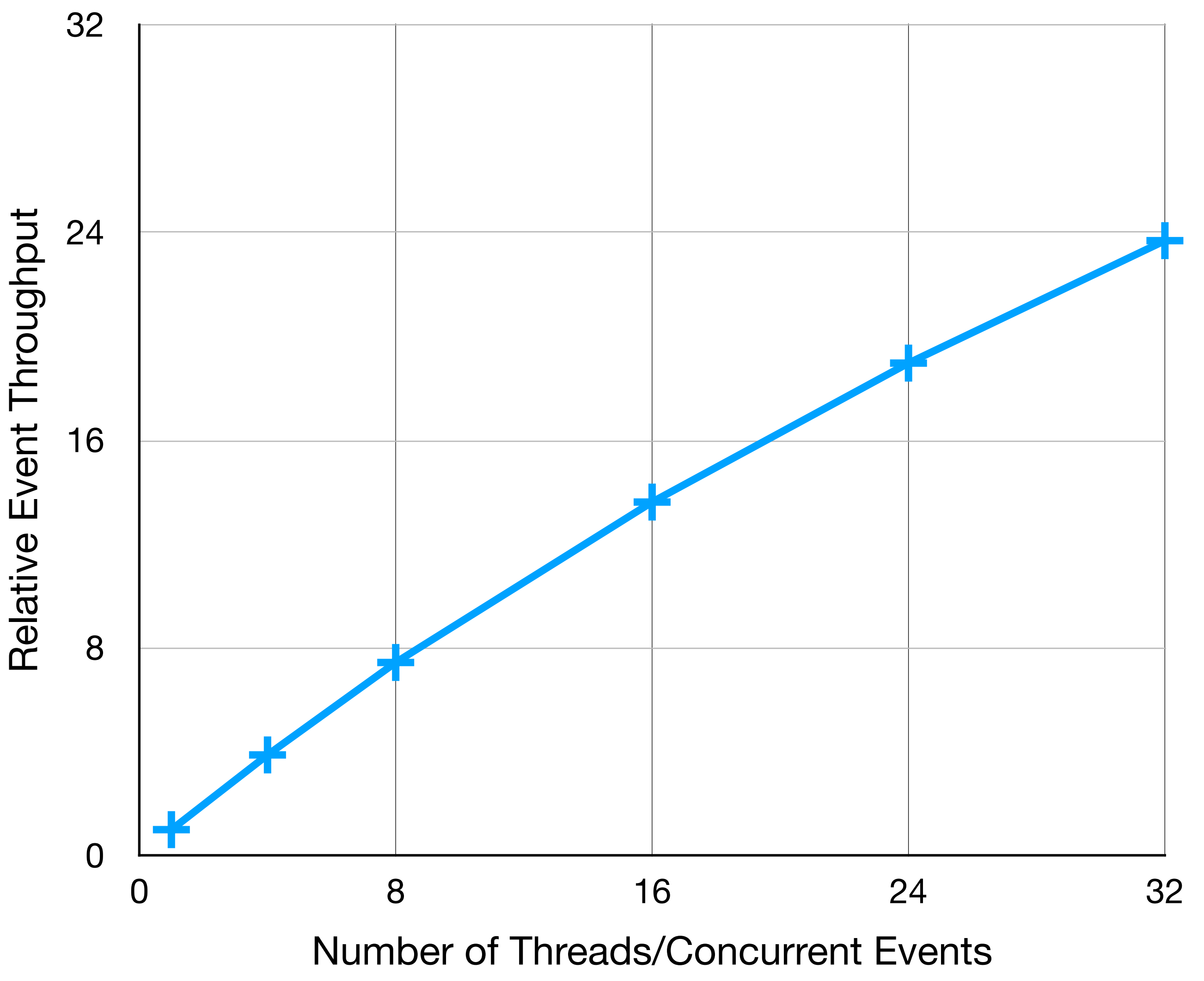
- Design
  - Each Event is written to the file atomically
    - no coupling across Events
  - Events on disk are just a collection of serialized data products
    - no coupling across data products

- Implementation
  - Data products can be concurrently serialized using ROOT serialization
  - Once all data products for an event are serialized the Event is compressed
    - Different Events can be concurrently compressed
    - Use LZ4 compression algorithm
  - Compressed Event is written sequentially to disk
    - No attempt to concurrently write different Events

RECO: Simple Data Format

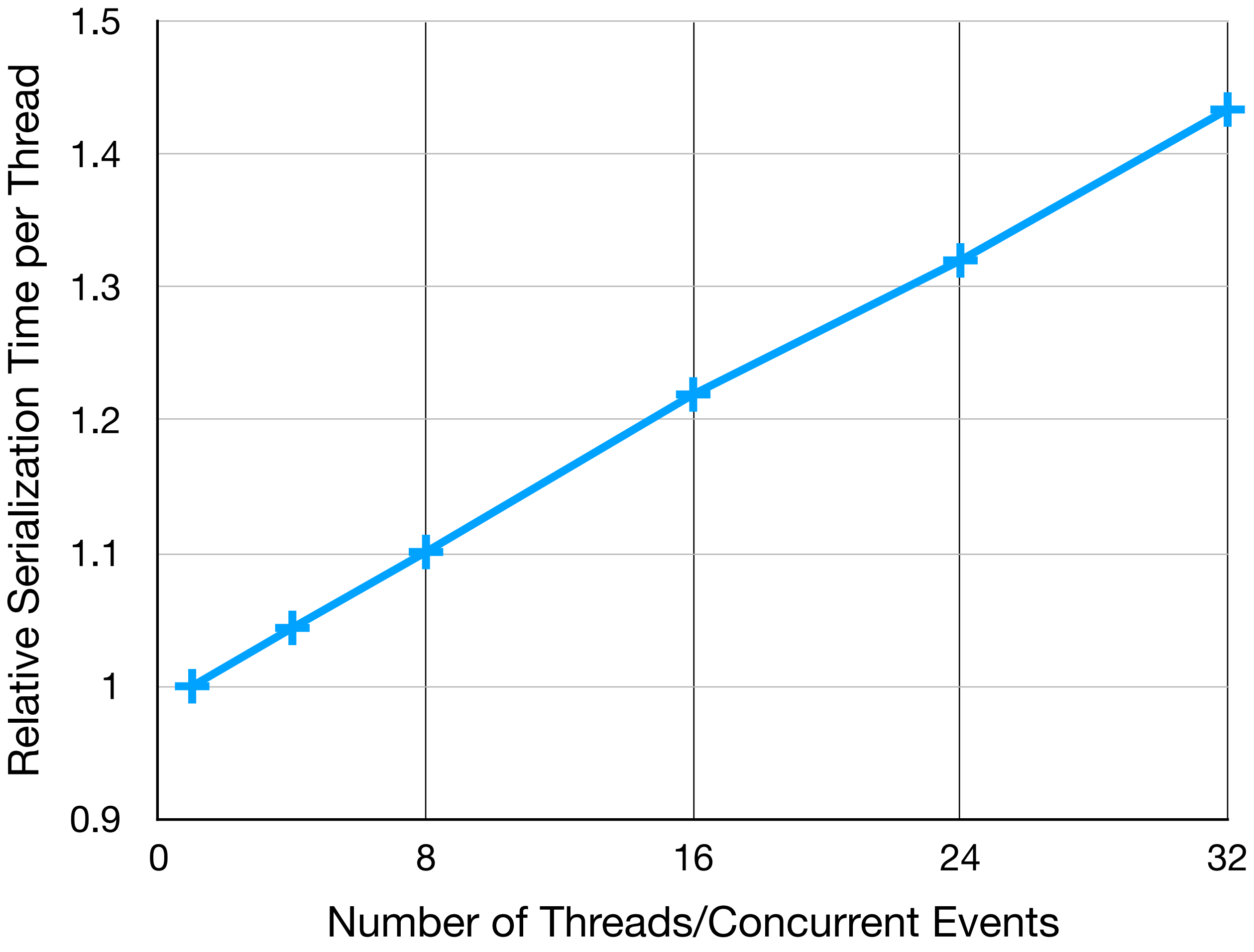
- Very fast with good thread scaling
  - 2.4x faster than ROOT format at 1 thread
  - 40x faster than ROOT format at 32 threads

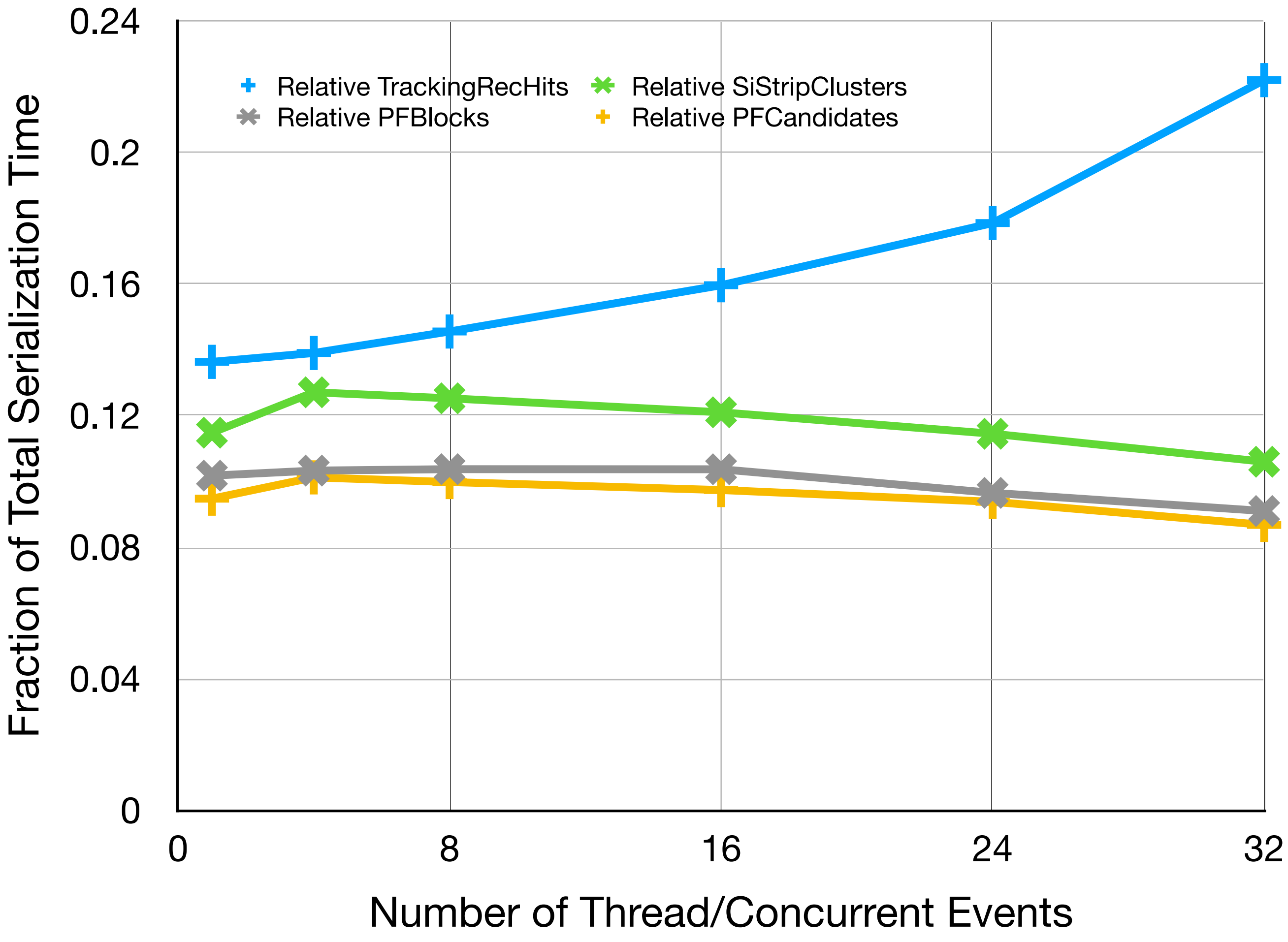




RECO: Simple Data Format (continued)

- Loss of scaling primarily due to serialization not scaling perfectly



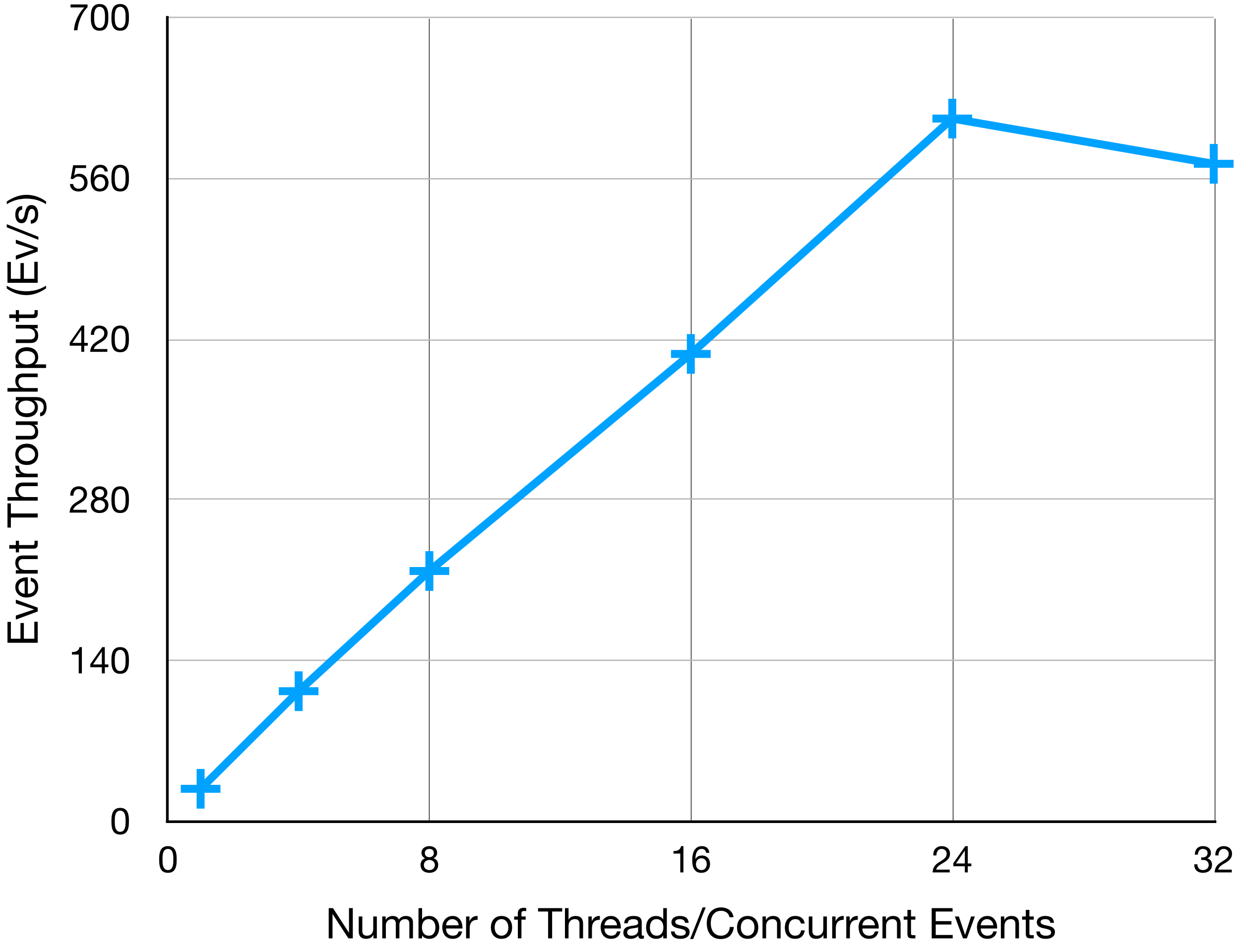


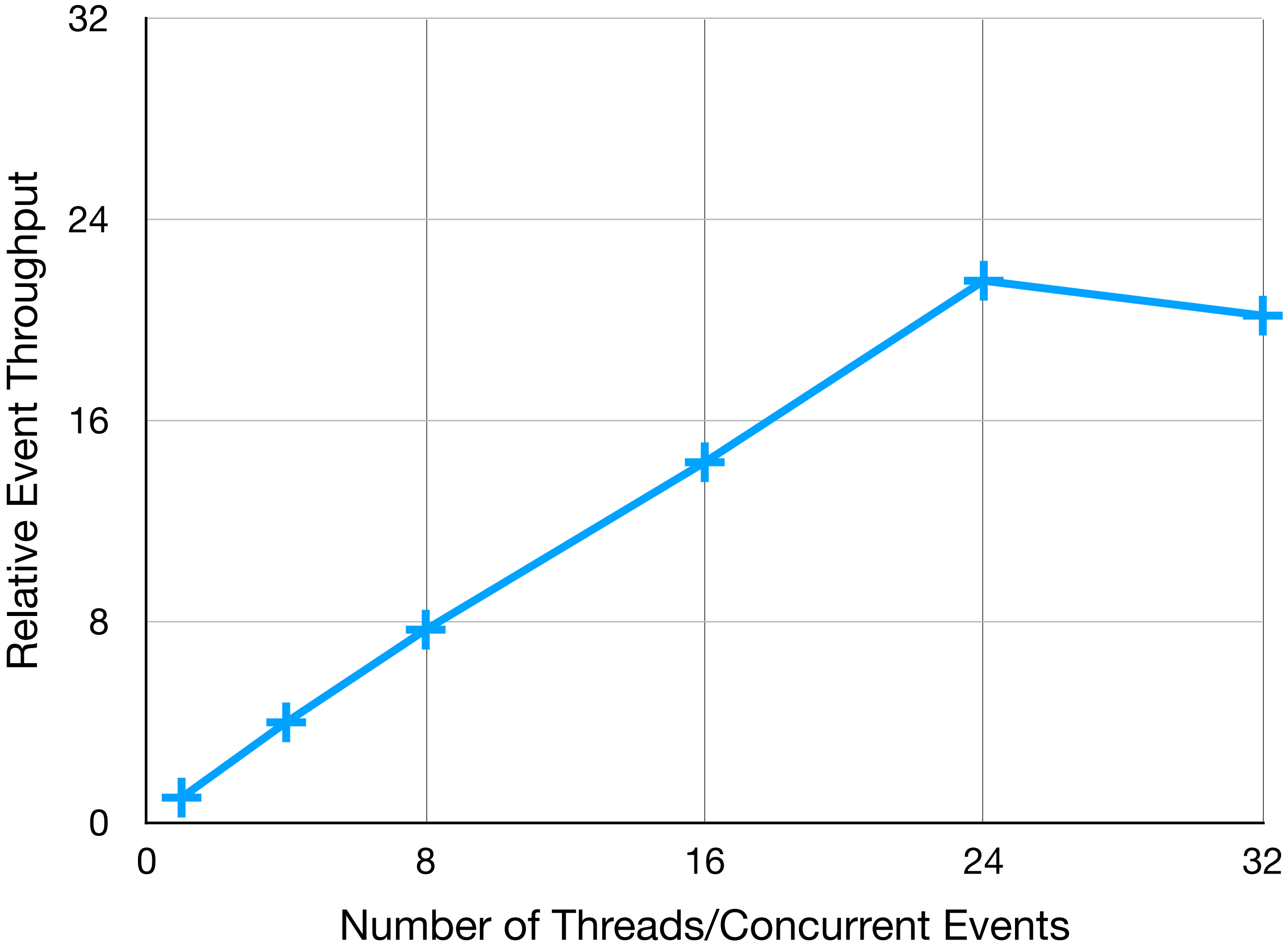
AOD Format

AOD: Use ROOT Serialization

- Serialize the data products read from the file
  - Each data product can be serialized simultaneously
  - Events are processed simultaneously

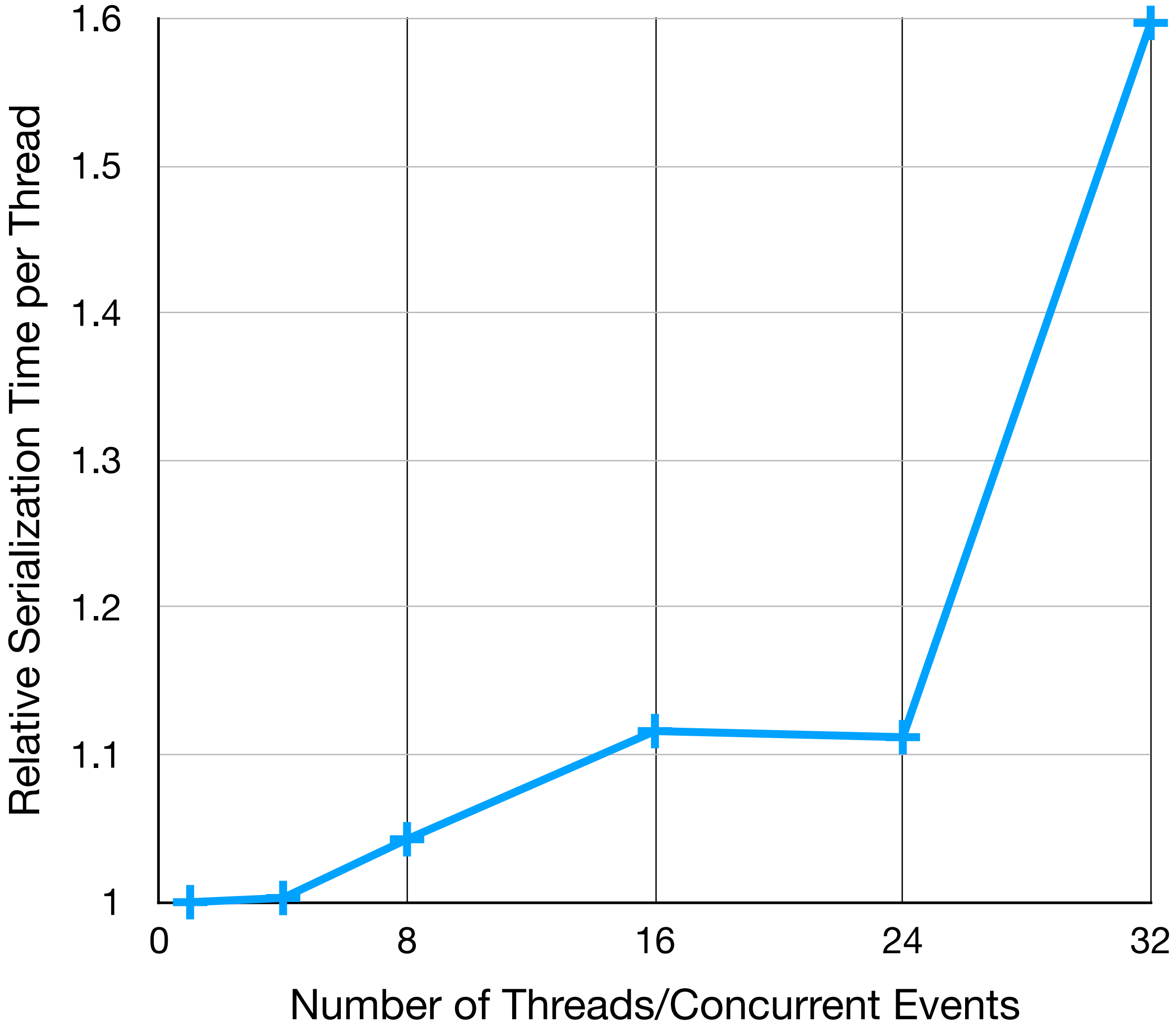
- Very good scaling
  - breaks down around 24 threads

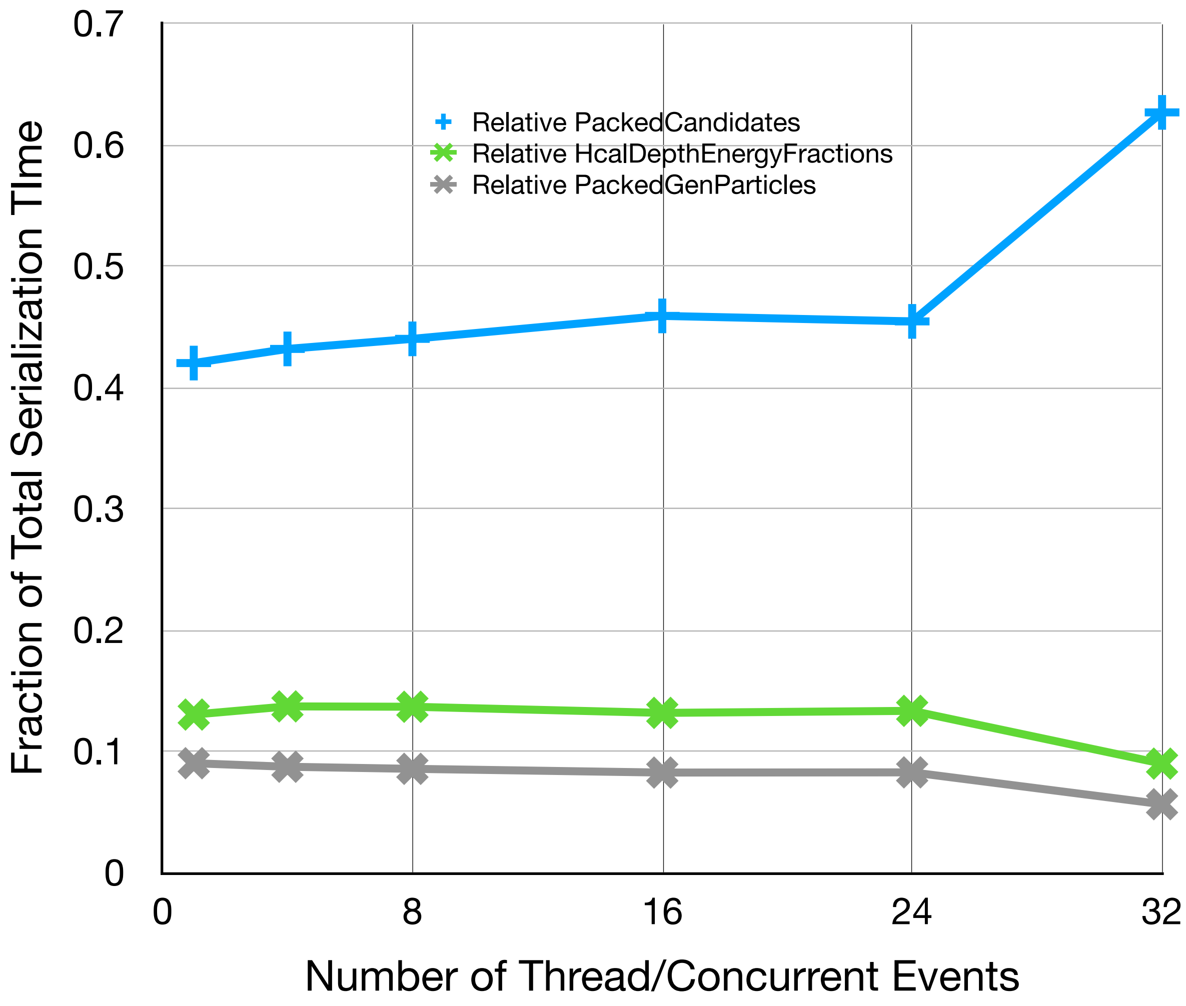




AOD: Use ROOT Serialization (continued)

- ~60% of serialization time comes from 3 data products
- Very good scaling until 32 threads where 1 data product stops scaling

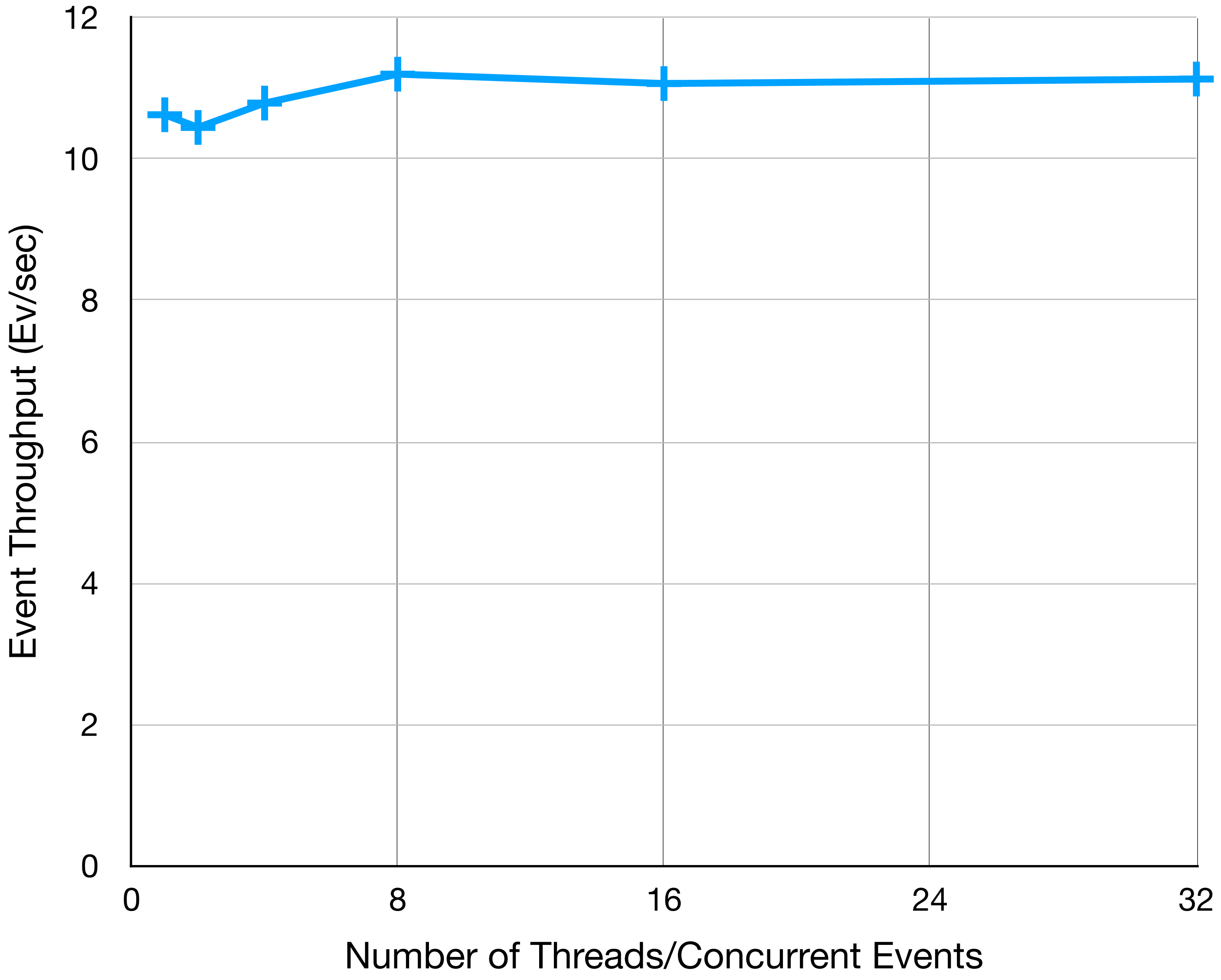




AOD: Write ROOT File Fast

- Write to /dev/null
- Disable use of compression

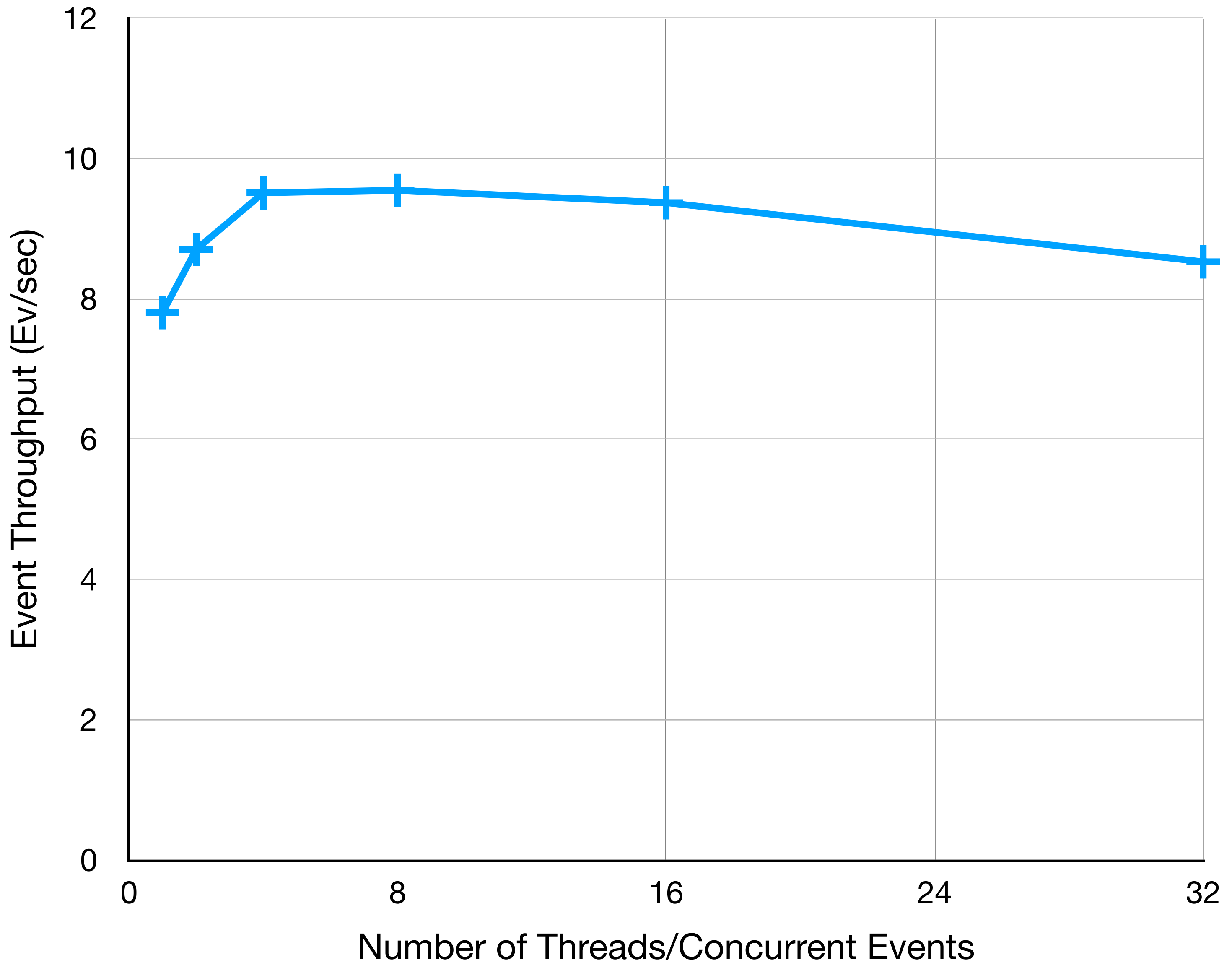
- No scaling
  - This was expected as no IMT used



AOD: Write ROOT File ‘Realistic’

- Write to /dev/null
- Use LZ4 compression
- Use IMT

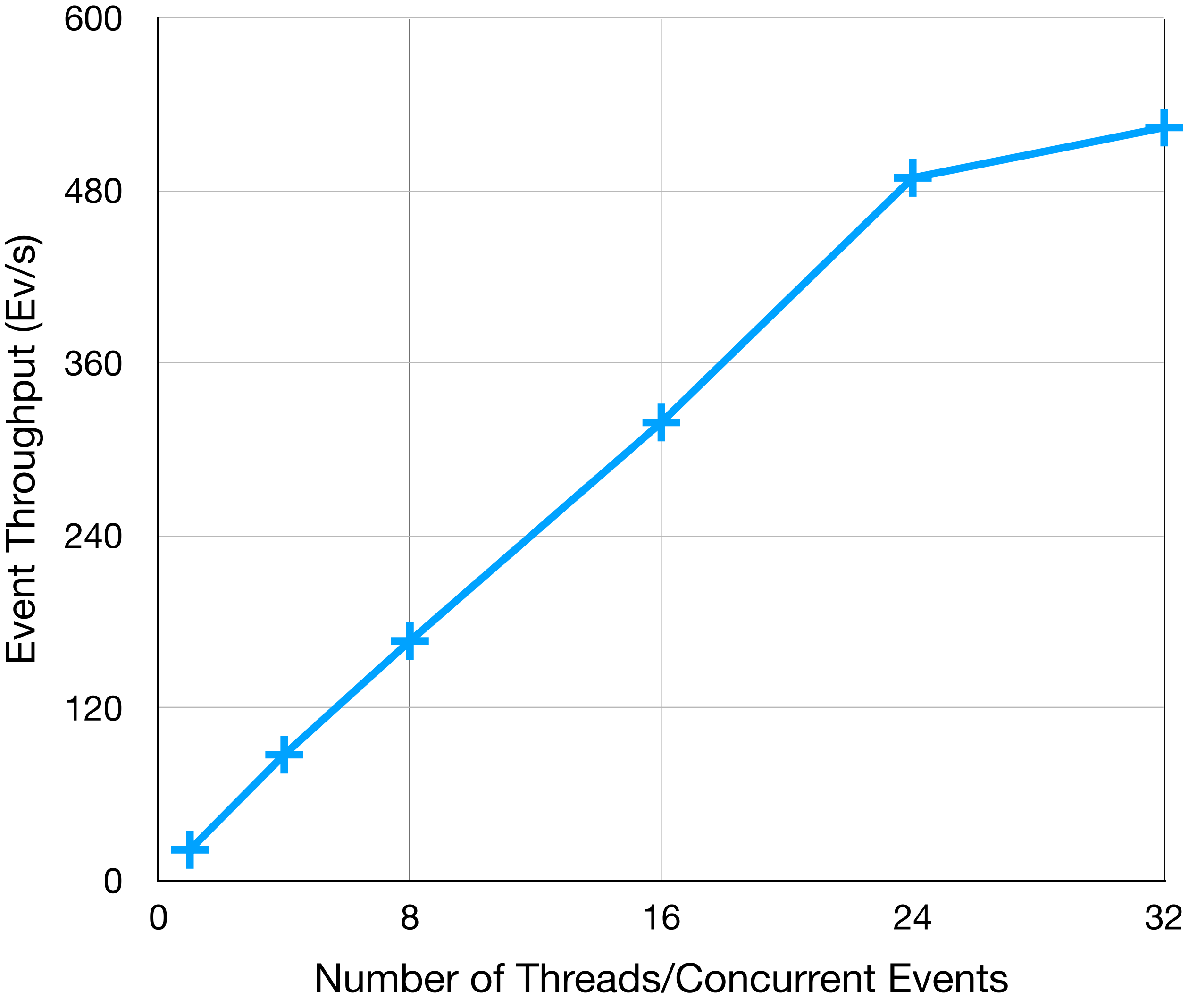
- Very limited scaling
  - Time in compression is minimal

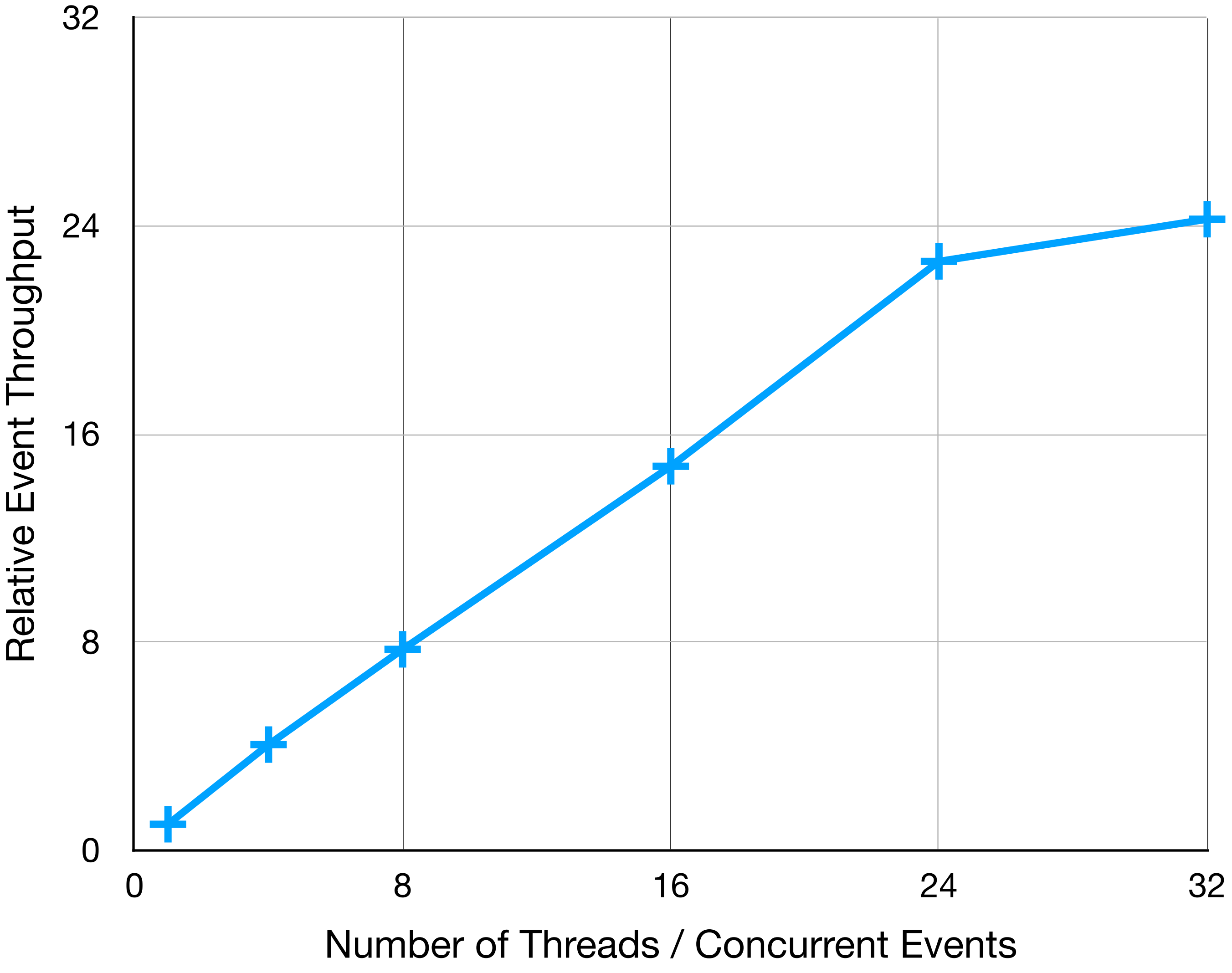


Fermilab

AOD: Simple Data Format

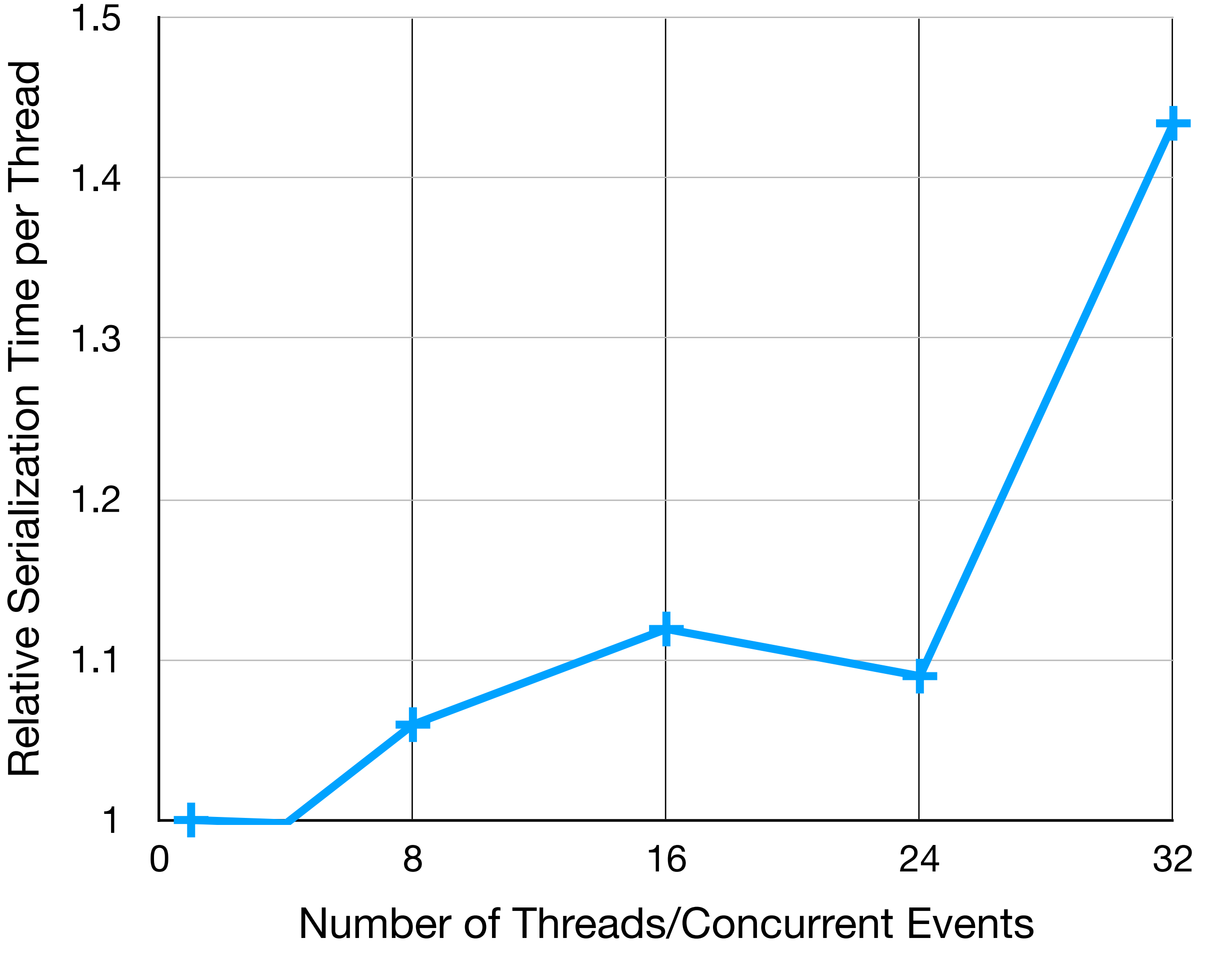
- Very fast with very good thread scaling
  - 2.6x faster than ROOT format at 1 thread
  - 58x faster than ROOT format at 32 threads

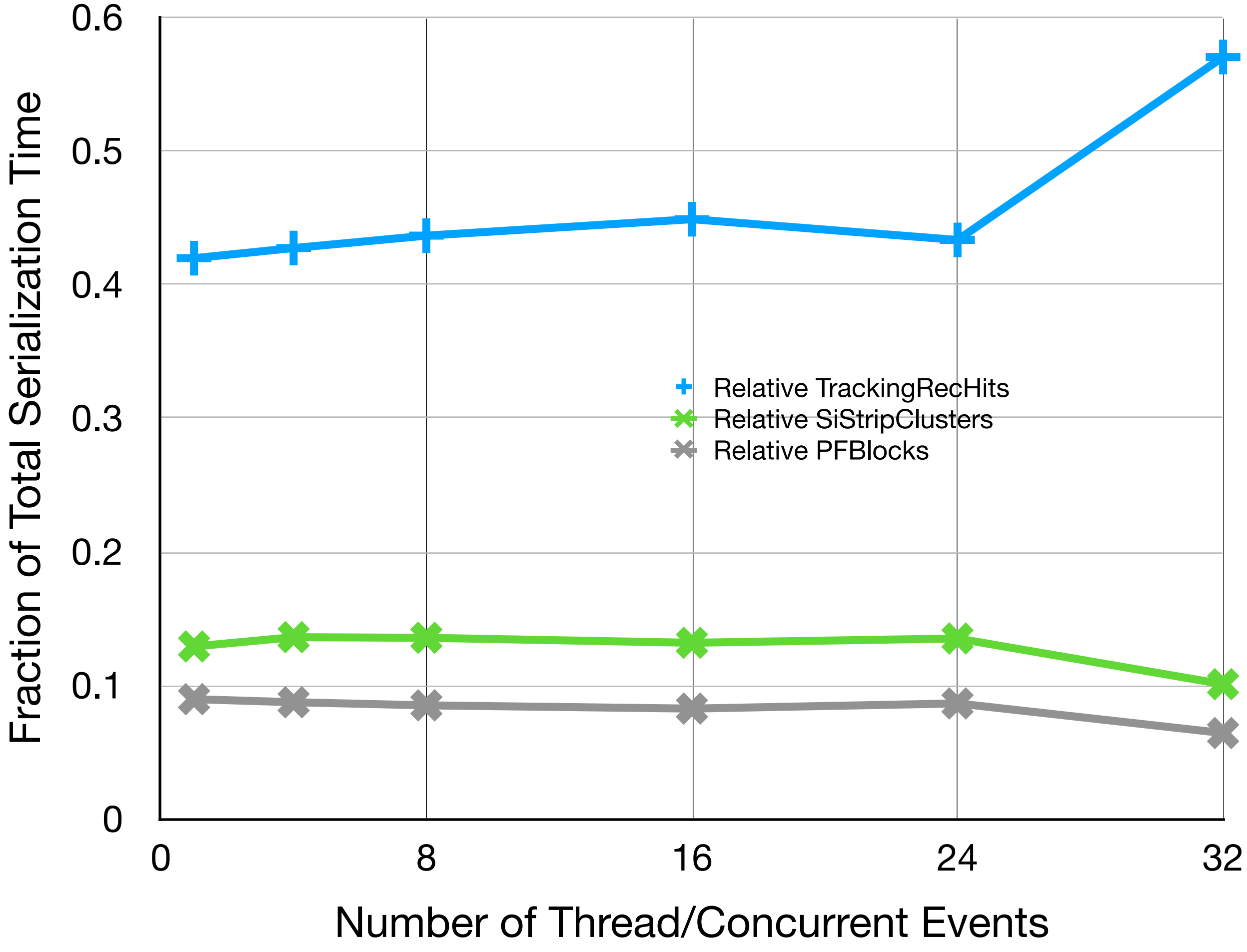




AOD: Simple Data Format (continued)

- Good scaling due to serialization scaling well up to 16 threads



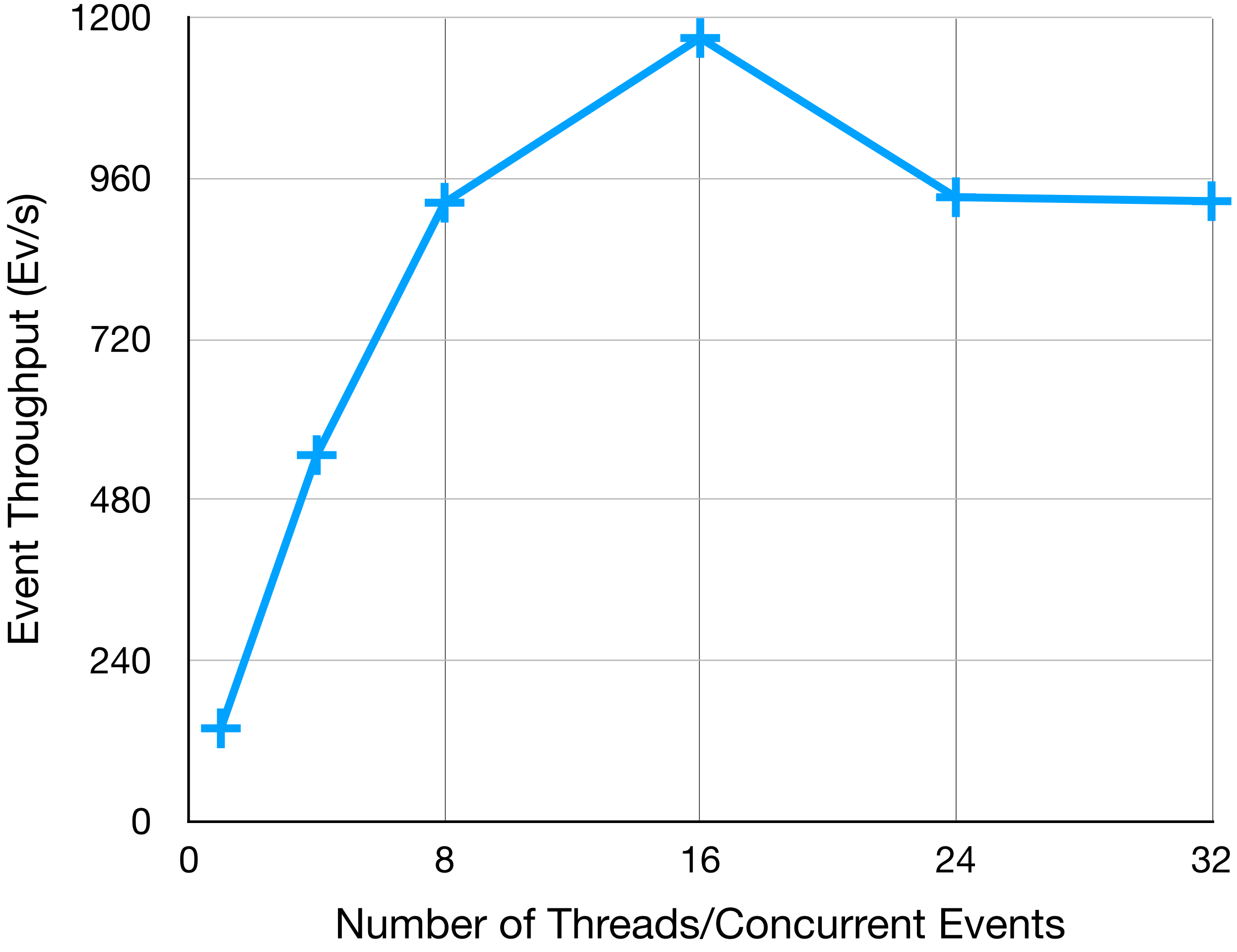


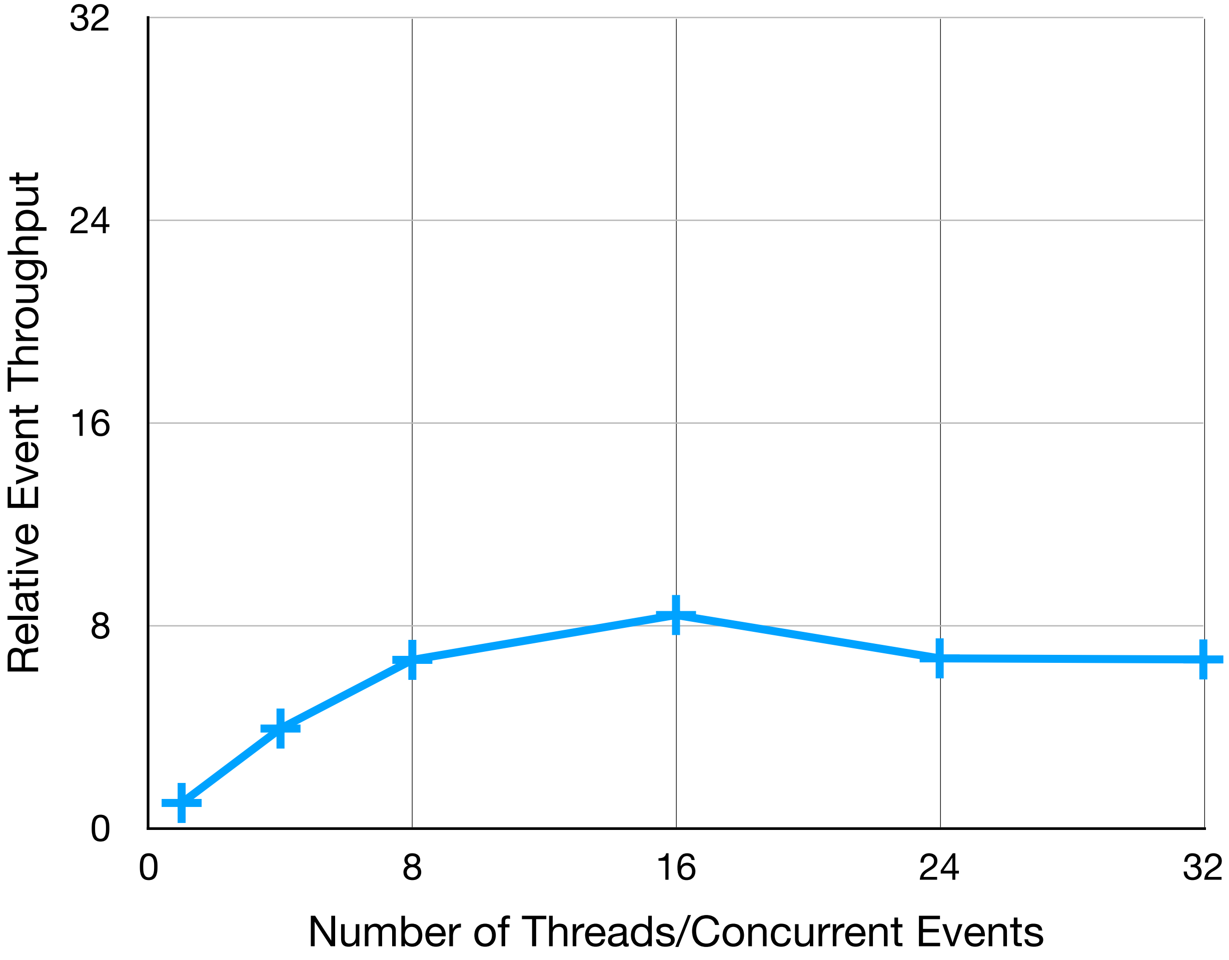
MiniAOD Format

MiniAOD: Use ROOT Serialization

- Serialize the data products read from the file
  - Each data product can be serialized simultaneously
  - Events are processed simultaneously

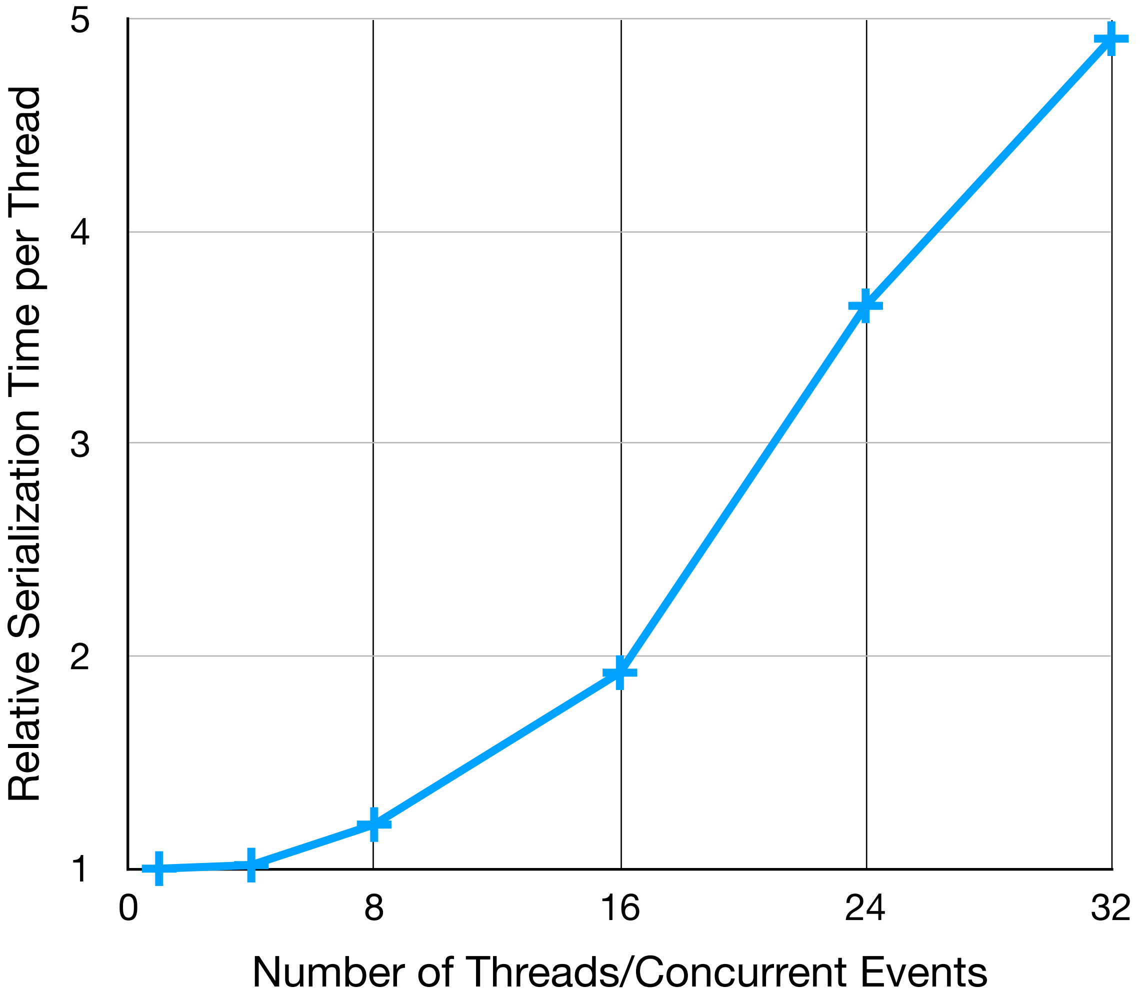
- Limited scaling
  - breaks down around 8 threads

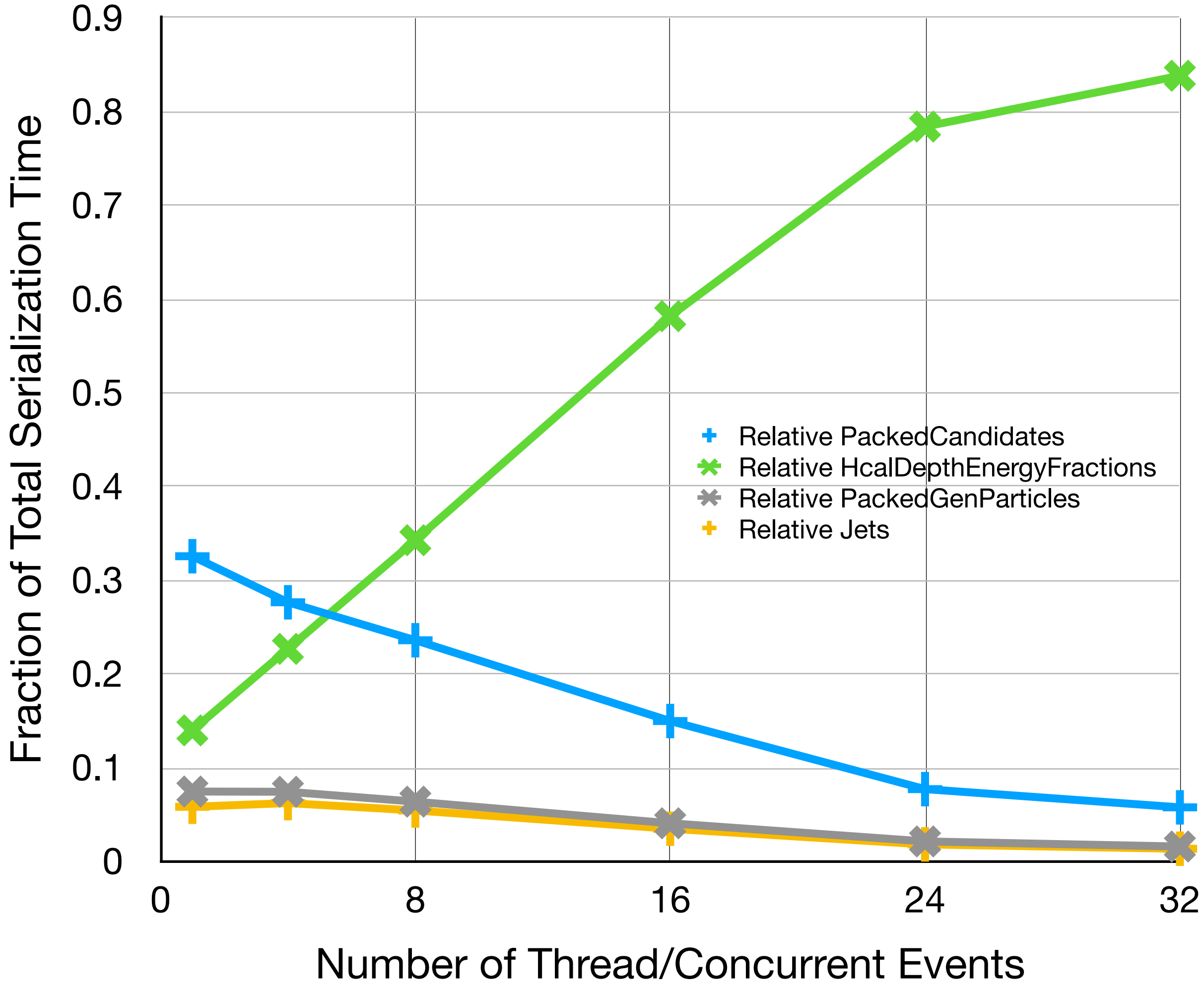




MiniAOD: Use ROOT Serialization (continued)

- ~70% of serialization time comes from 4 data products
- Thread scaling difficulties comes from 1 of the data products not scaling well

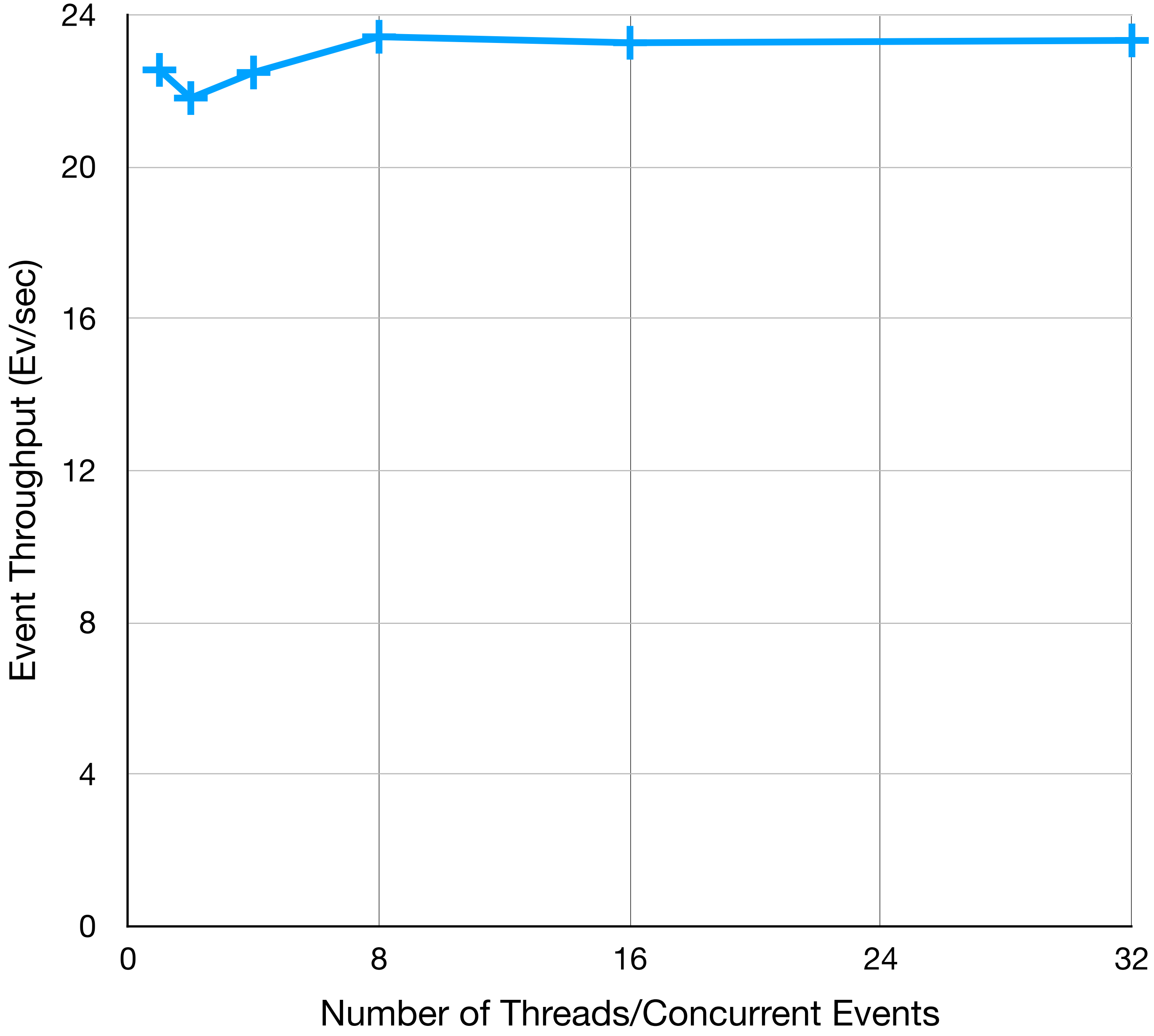




MiniAOD: Write ROOT File Fast

- Write to /dev/null
- Disable use of compression

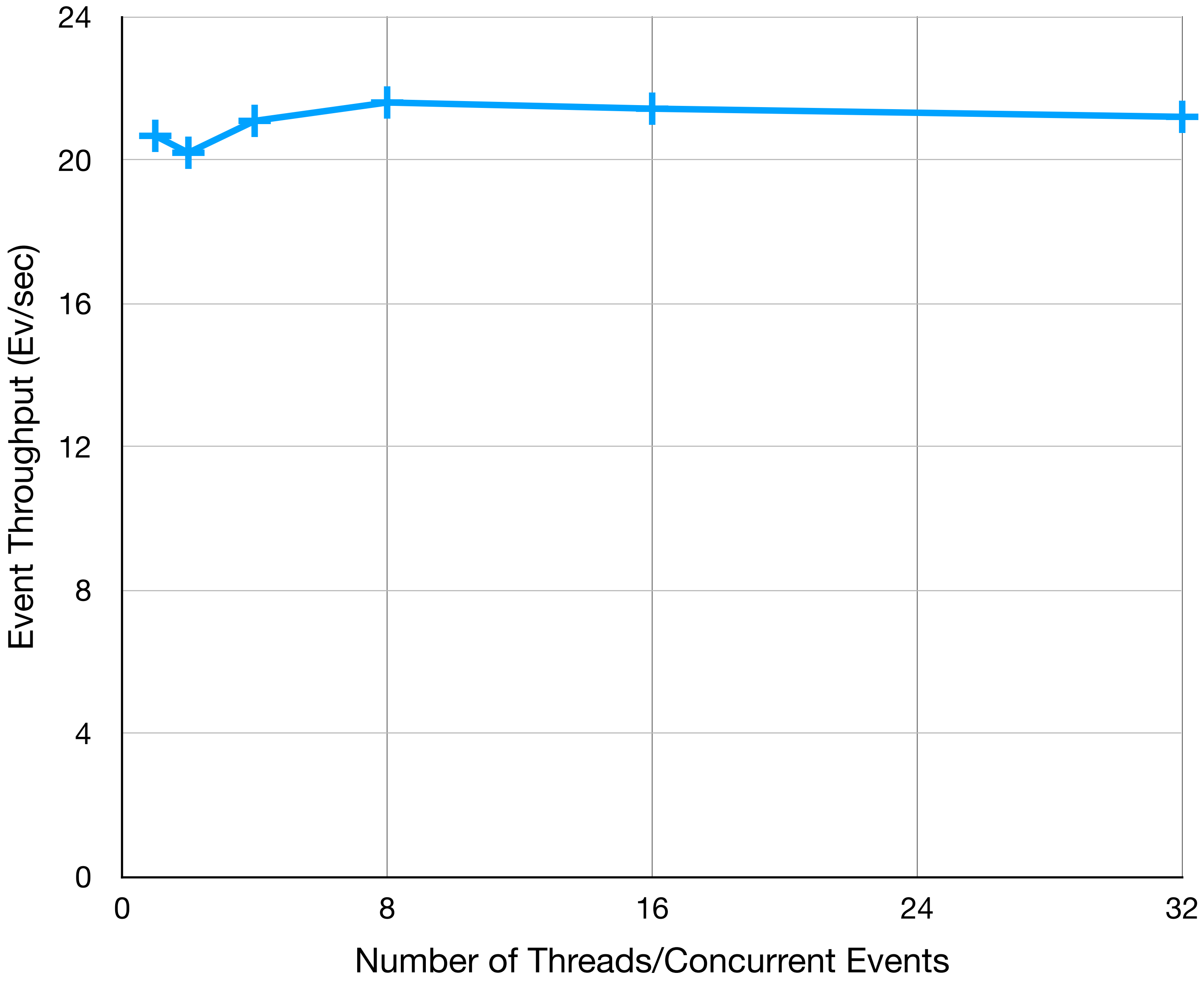
- No scaling
    - This was expected as no IMT used



MiniAOD: Write ROOT File ‘Realistic’

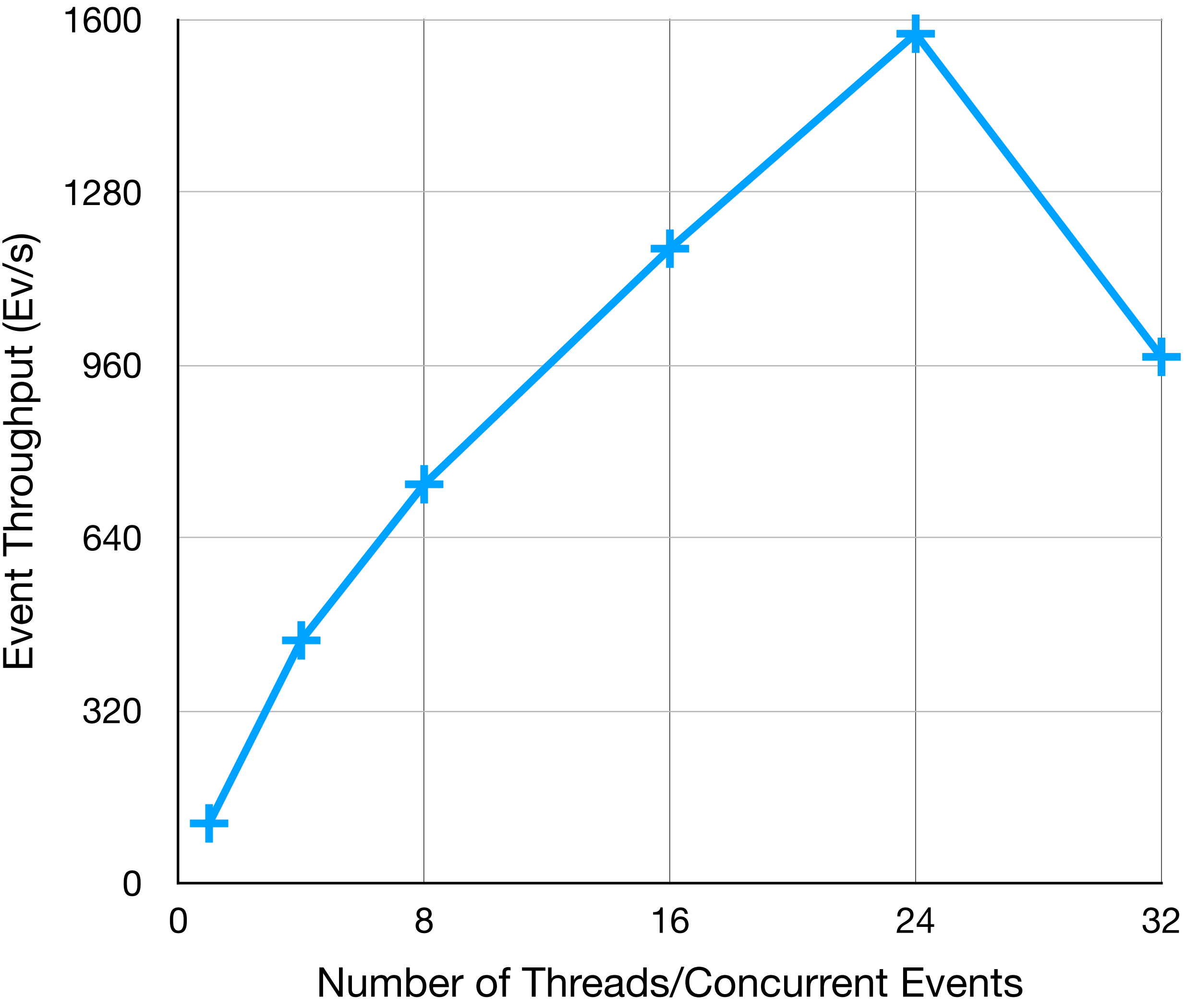
- Write to /dev/null
- Use LZ4 compression
- Use IMT

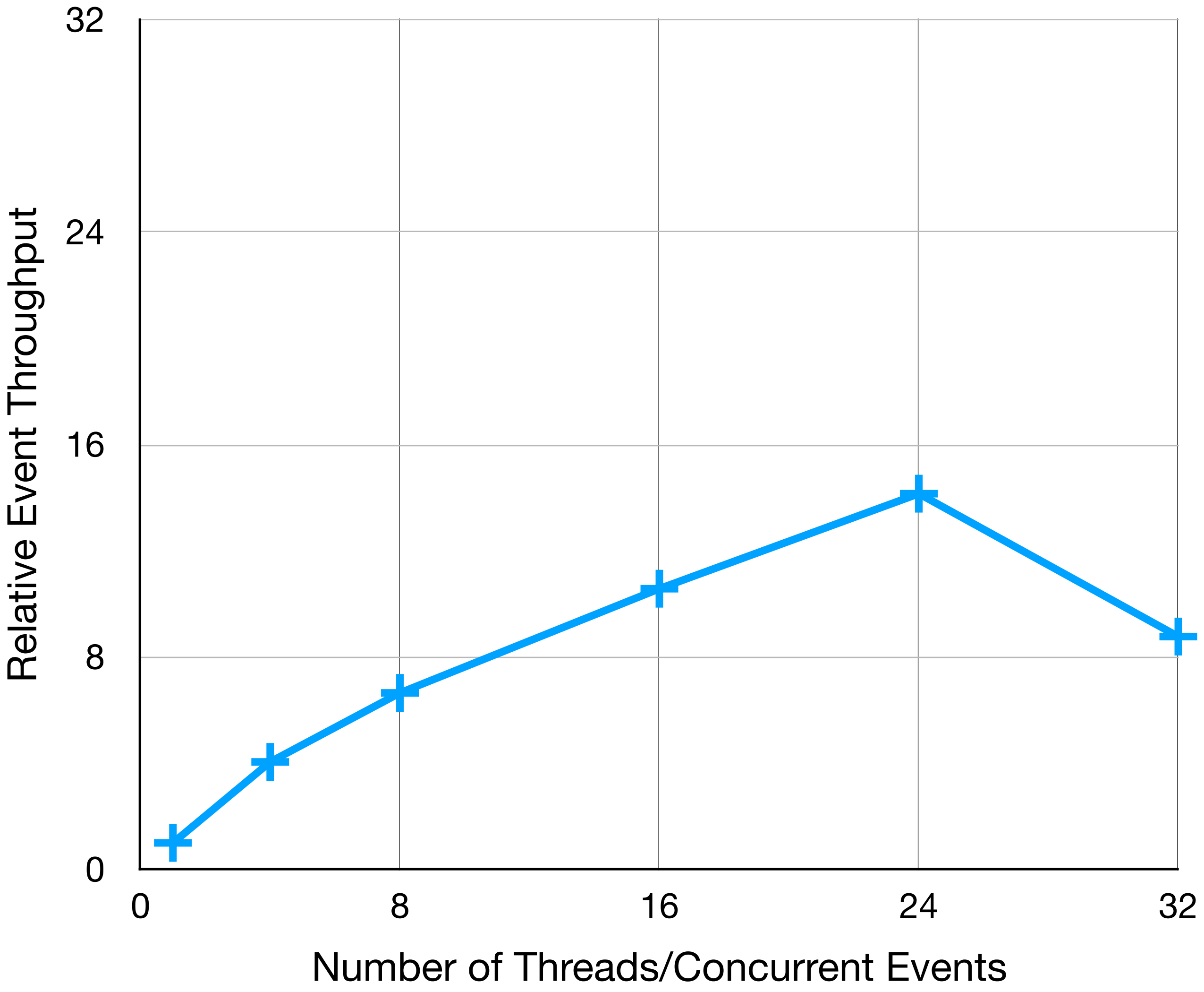
- No scaling
  - Time in compression is too little to make appreciable difference



MiniAOD: Simple Data Format

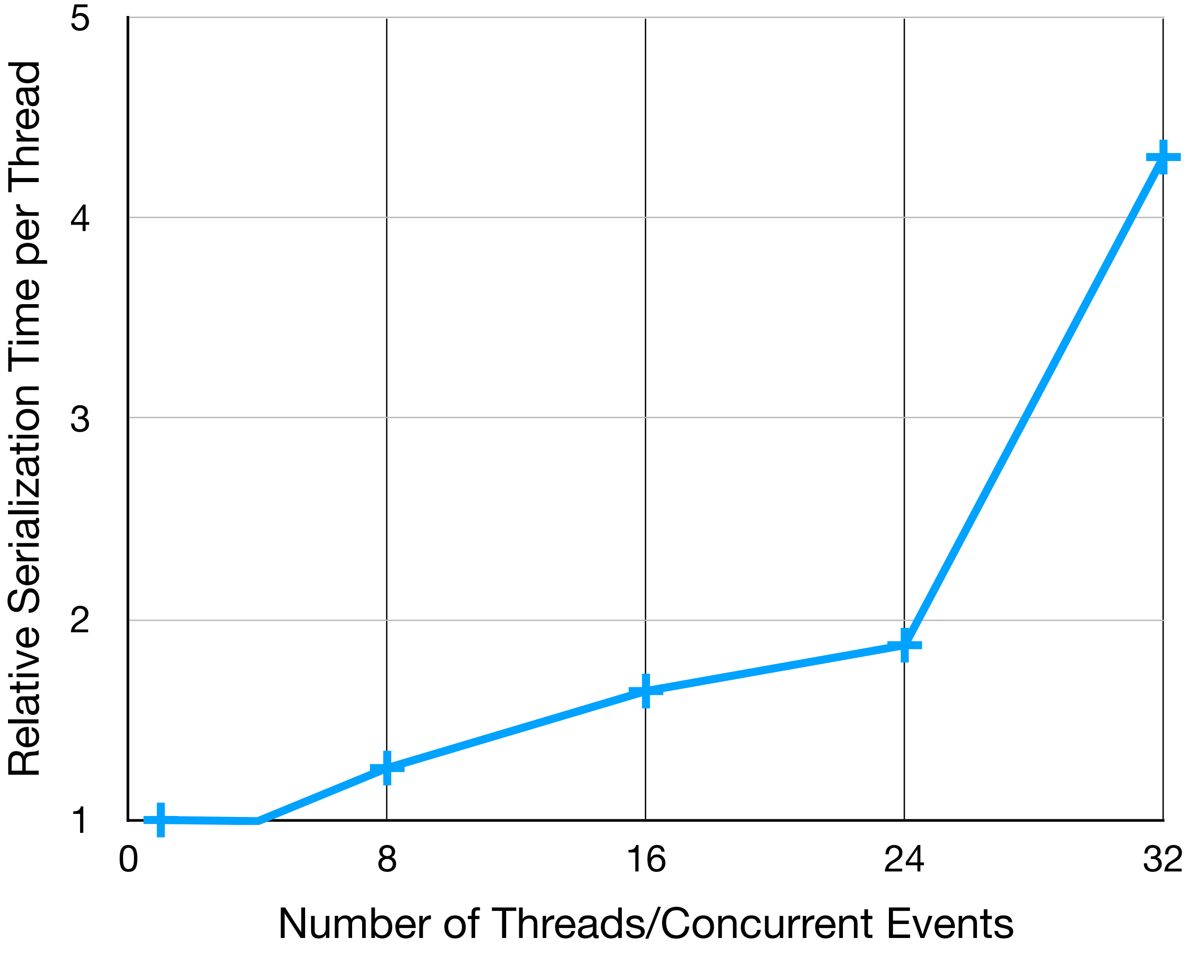
- Very fast with modest thread scaling
  - 5x faster than ROOT format at 1 thread
  - 45x faster than ROOT format at 32 threads

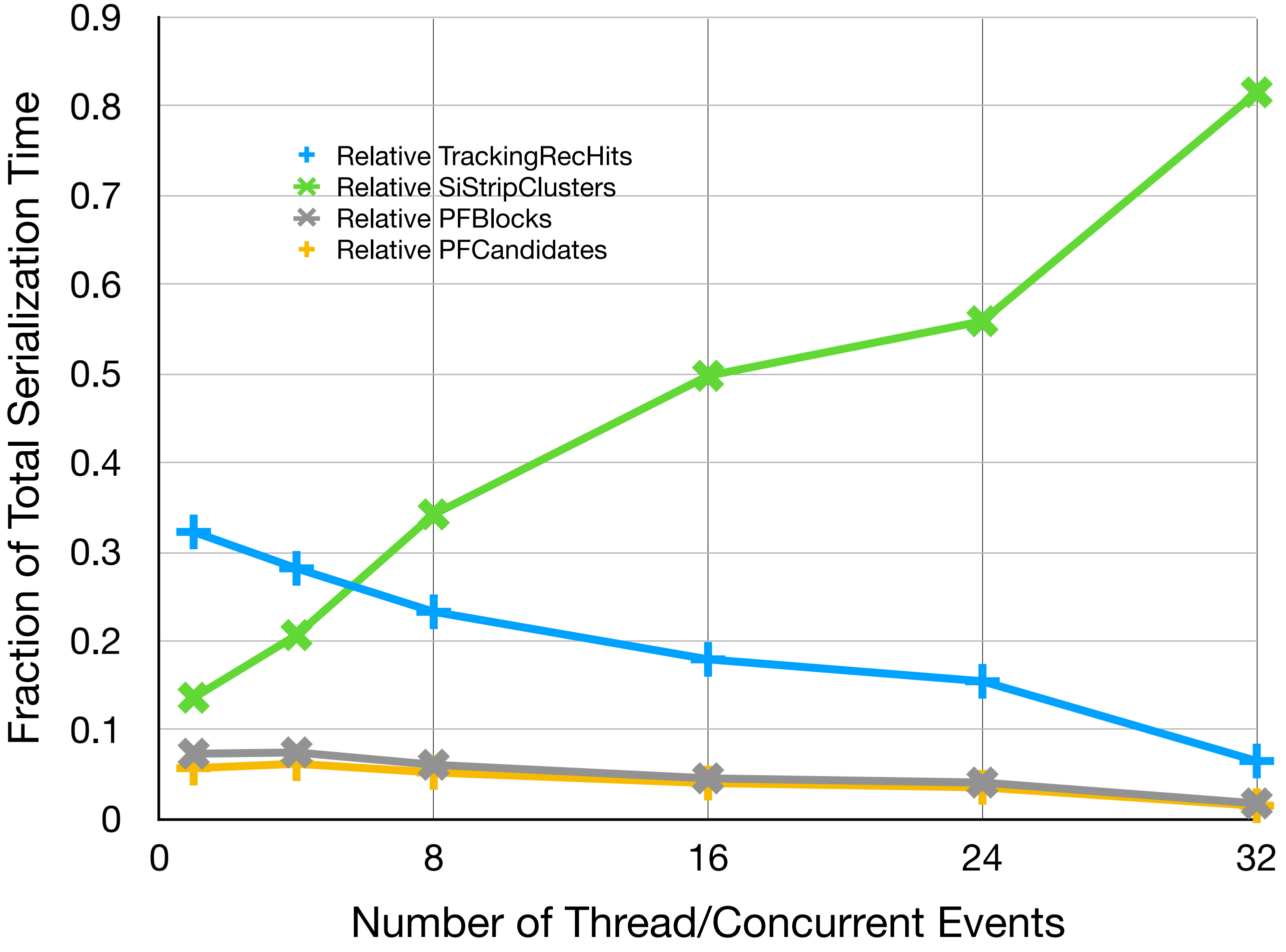




MiniAOD: Simple Data Format (continued)

- Loss of scaling due to serialization not scaling perfectly





Conclusions

ROOT to Simple Format Comparison

- Only looked at throughput comparison

- Other important factors
  - Amount of memory used
    - simple format is presently using a buffer per data product per Event
  - Resulting file size
    - simple format compresses Event by Event
    - ROOT compresses by data product (ish)
      - larger similarities in values allows better compression
  - Write performance
    - all tests done by writing to /dev/null
  - Read performance

ROOT to Simple Format Comparison (continued)

- Create CMS files with 100 events and converted to ROOT and Simple format using the testing framework

	File Event Size in MB		
	RECO	AOD	MiniAOD
ROOT	4.490	0.646	0.105
Simple	12.370	2.869	0.494
Size Ratio	2.76	4.44	4.70

Future Directions

- Write to actual storage
  - Need access to a *representative* node
  - Could also do a multi-node scale test to see cross-node effects

- Run tests using Saba's HDF5 based format
  - Can benefit from concurrent data product serialization

- Test with other experiment's data files
  - Best done by an experiment expert