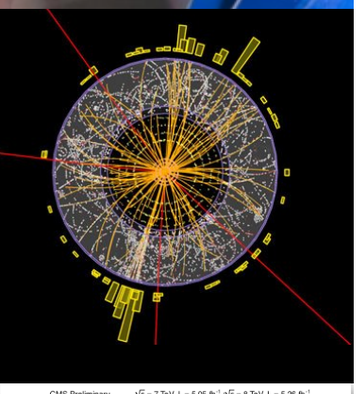
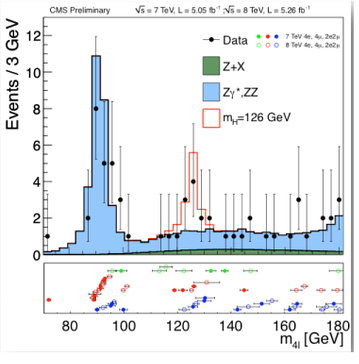
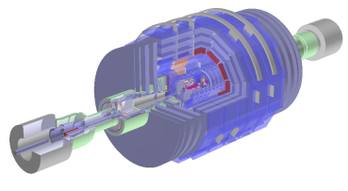




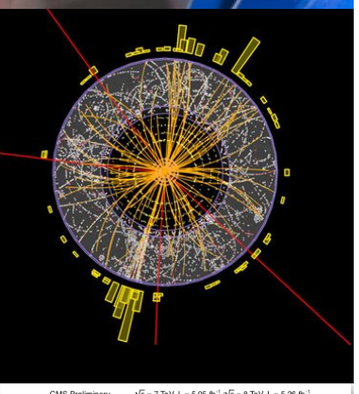


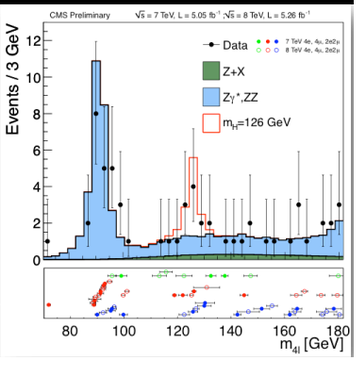
ROOT I/O Performance In Multihread Environment

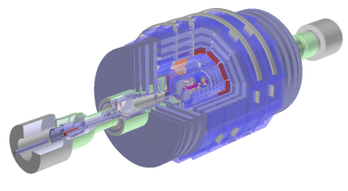
Philippe Canal
Fermilab

Philippe CANAL
root.cern.ch

Overview







- Support for I/O in multi-thread environment
- CMS Event
- CMS Condition Database
- Conclusion

Support

- With be only for C++11 but both in v5 and v6.
  - Relies on std::atomics

- In v5, limited to non-interactive sessions
  - High deadlock risk when starting the command line.

- Cost of atomics (and thread_locals) about 5% of streaming time (mitigated by C++11 being 2% faster).

Removal of unnecessary serialization

- Update TClassRef
  - From a linked list of ref per TClass object update at each creation/deletion of TClassRef
  - To a single pointer to TClass* per TClass object shared by the TClassRef
    - TClass* is allocated once per TClass and never changes
- TClass::GetClass
  - Move code around to reduce length the lock is held.
- TClass::Get/FindStreamerInfo
  - Remove use of lock in the common case by caching in an atomic the most recently found for each Tclass
- TThread::Self
  - Remove linear search doing string comparison by using thread local storage.
- Remove locks in TBaseClass by caching information.

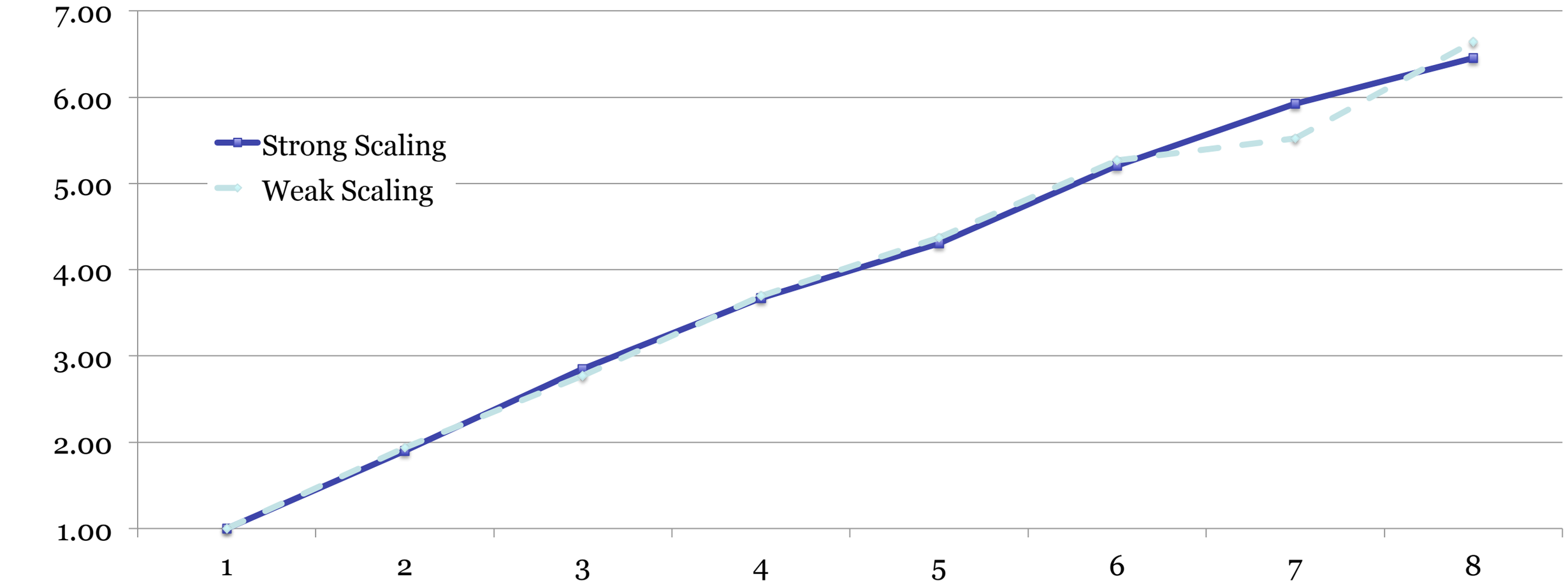
Test Machine

- Cpu: Intel Xeon X5570 @ 2.93GHz
  - 2 CPUs for 8 cores (16 including hyperthreading).
  - Cache size: 8192 KB

- 12GB of DDR3 at 1333 MHz

CMS Event

- CMS Event file 133MB, 100 entries.
- One TFile and TTree per thread.
- Use TTreeCache and slightly modified MakeProject lib.
- Less than 5% serialization

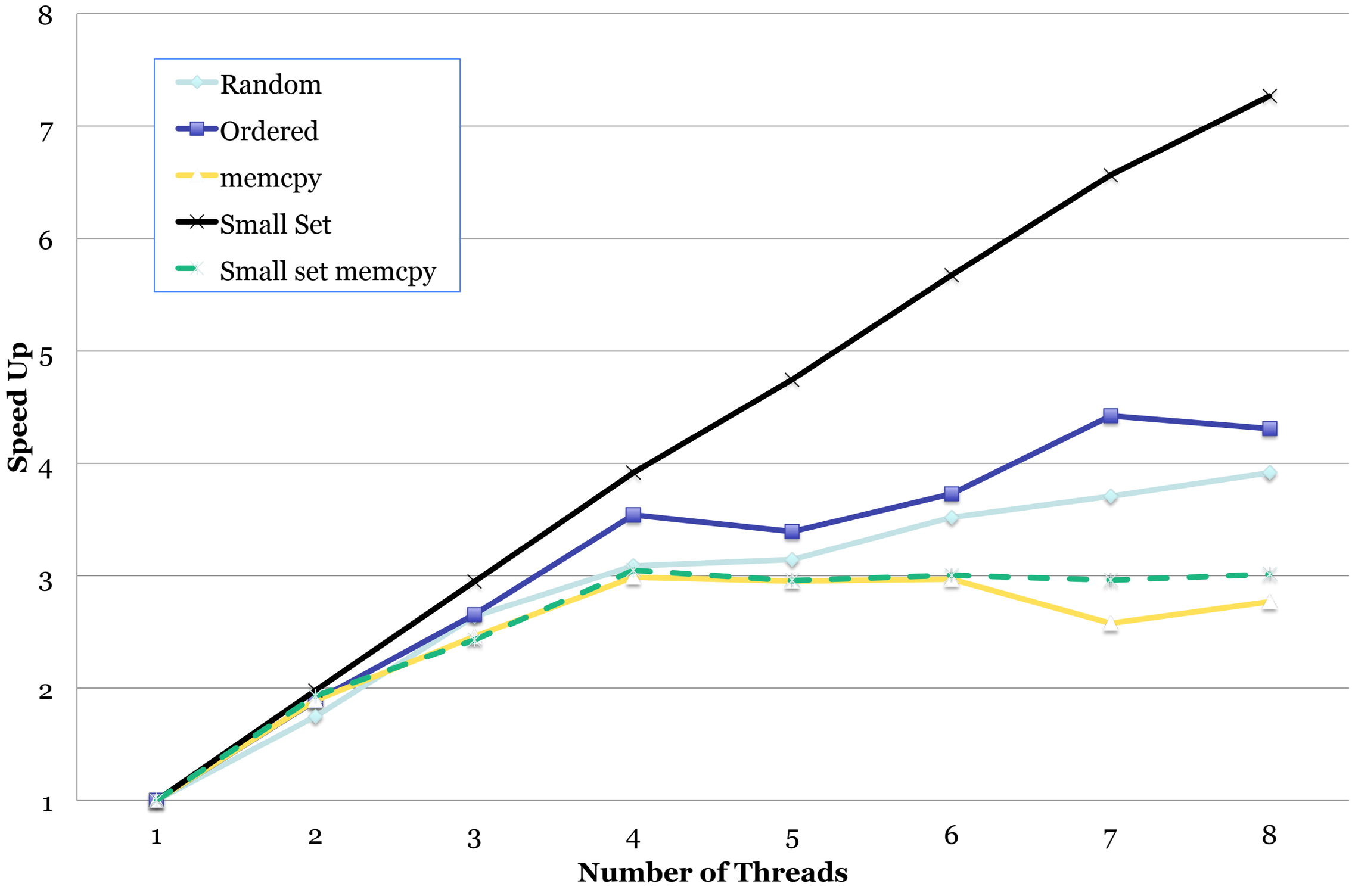


Test Description

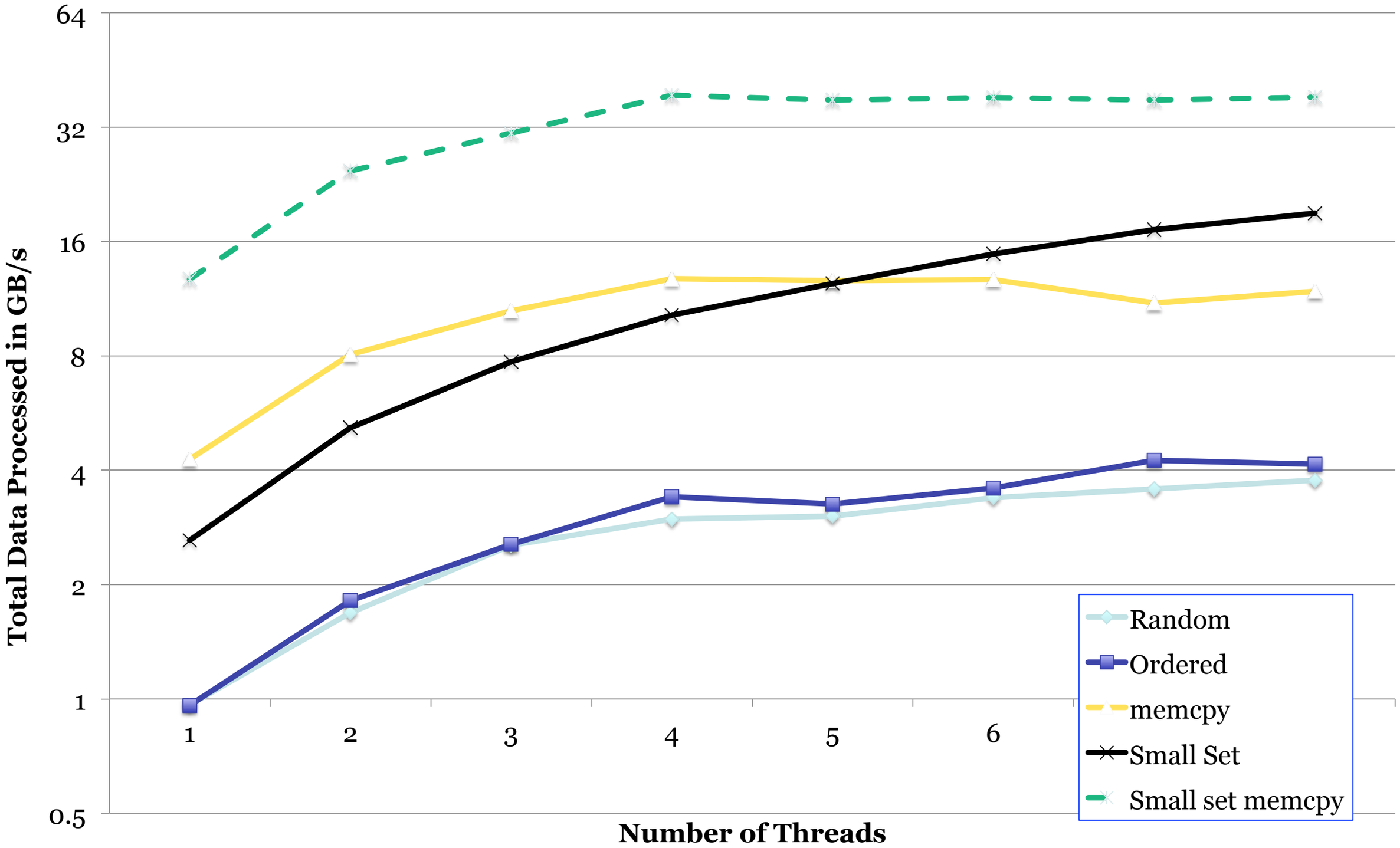
- 226 distincts CMS condition database objects
- 233MB of data

- Run example with no thread enabled then 1 through 8 threads.
- Load the data (amount varies) into 1 TBufferFile per thread.
  - Small Set: first 3 objects for 393KB.
- Each thread deserialize the content multiple times

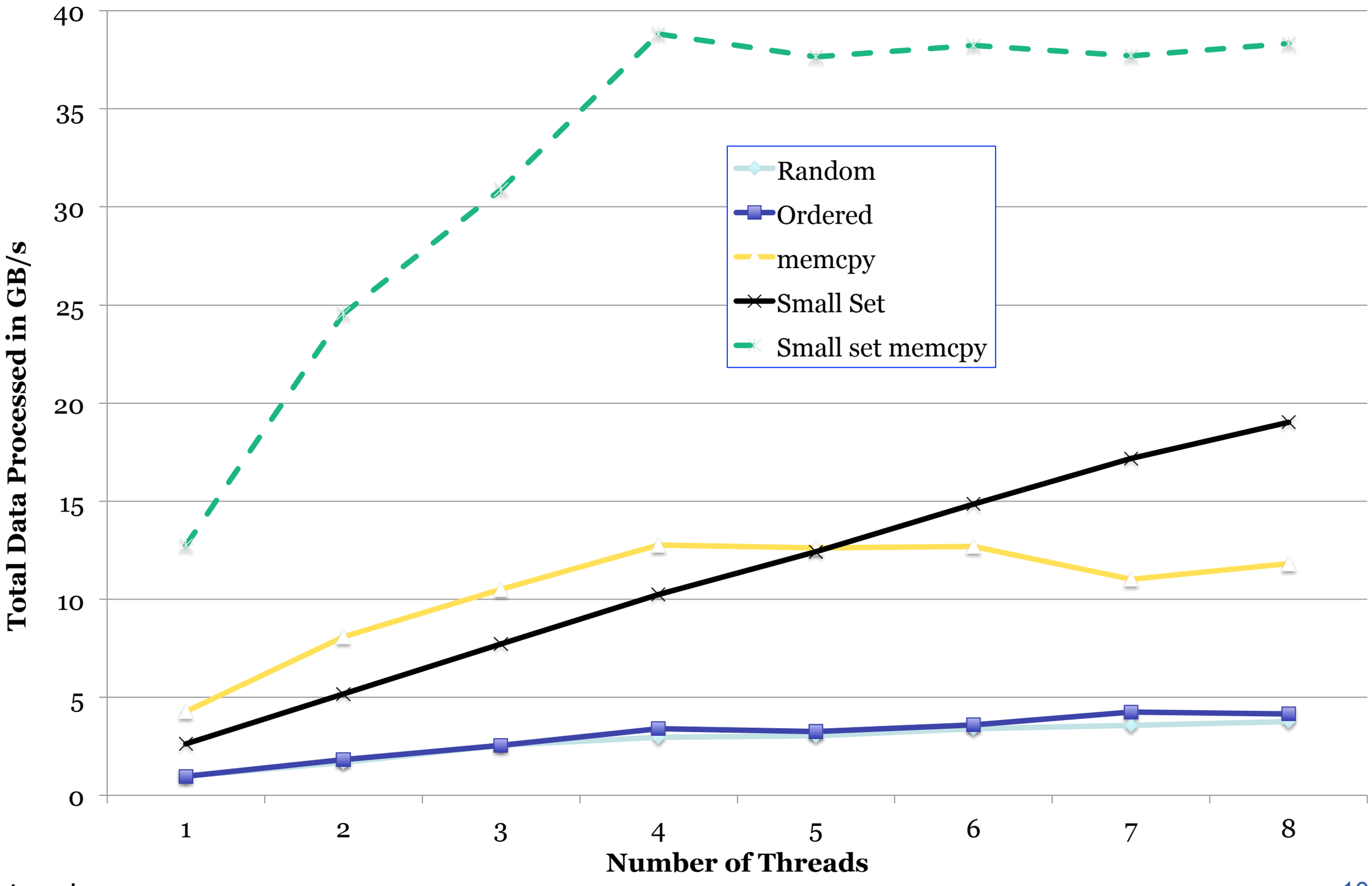
CMS Condition Database



Data Bandwidth



Data Bandwidth



Observations

- Average streaming performance nicely (1 GB/s)
- But varies a lot

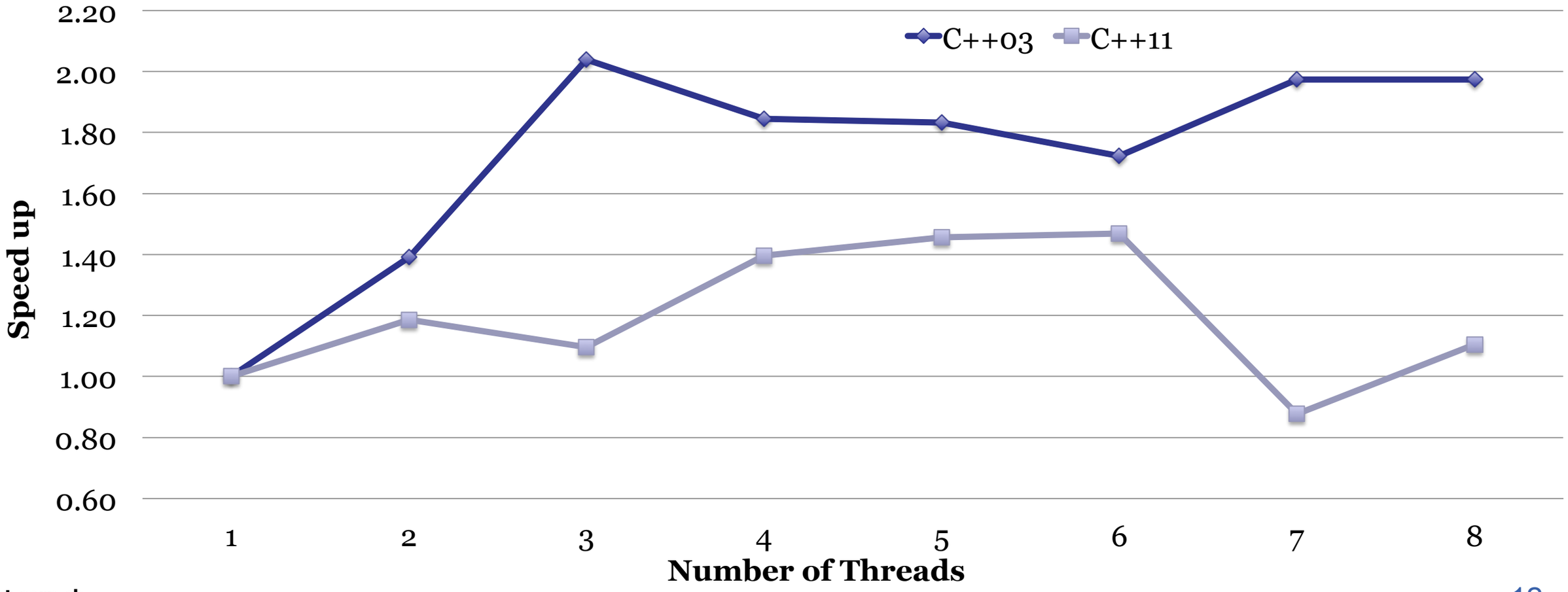
- Condition database objects very varied.
  - Top 6 objects takes 60% of the time but 15% of the size.
  - This will limit the amount of possible parallelism.

- For example
  - IdealGeometryRecord / PGeometricDet
    - 8MB (3% of total)
    - 23557 std::string in an object in a vector
    - Read @ 325 MB/s … 12% of total
  - SiPixelGainCalibrationOffline
    - 65MB (28% of total) … Read @ 4680 MB/s … 6% of total
  - L1MuDTPtaLut
    - 448Kb (0.2% of total) … Slowest read @ 25 MB/s … 8% of total …
    - Contains a vector<map<short,short> >.

Worst case.

- Class L1RPCConfig
  - 22% of time, 3% of space
- Most time consuming object.
- Contains vector of 93160 objects
  - which contains an array of 6 objects.
  - Each of those contains 2 bytes!

- High serialization
  - Atomics and lock play a role but not enough to explain behavior
- But no clear explanation
  - Maybe try running in Vtune



Conclusion

- ROOT I/O is now thread friendly
  - It works!
  - Less than 5% serialization on CMS Events TTree.
  - Less than 15% serialization on all CMS cond db objects.
- Some object layout lead to poor performance and poor scalability.
- More can be done to optimize
  - Reduce number of ‘class/version/checksum’ searches.
    - To reduce the number of atomic and thread local uses.
  - Change byte swap order (increase memcpy case)
  - Continue refactoring of the I/O internals
    - Increase vectorization, reduce branches, etc.