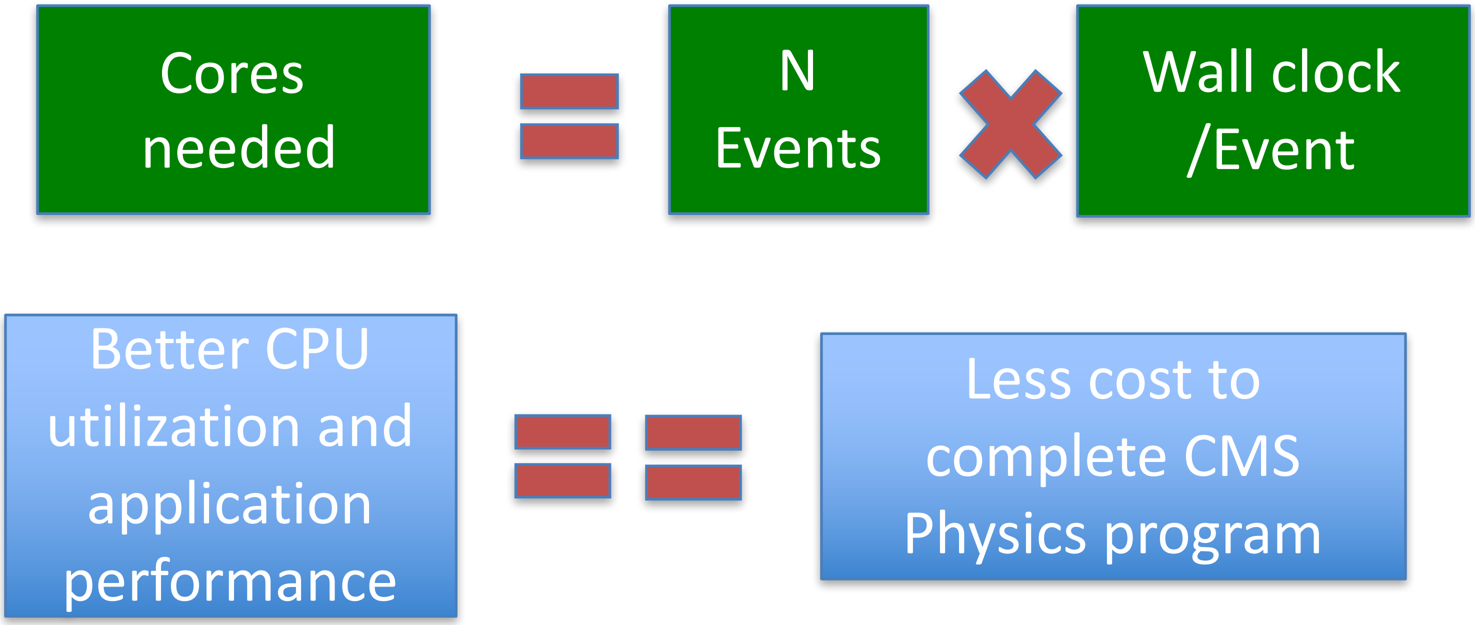
CMS software and infrastructure performance and efficiency gains

David Lange

October 10, 2016

Motivation: Technical improvements can pay off

- Time required to process drives CPU requirements for keeping up with LHC data



Cores needed = N Events × Wall clock /Event

Better CPU utilization and application performance == Less cost to complete CMS Physics program

Where do we lose efficiency

Pilots:

- **Retirement** / draining of multicore pilots. If we split 8-core pilots into 8 single-core payloads, we must wait for the last single-core payload to end.
- **Negotiation**: Time between jobs and time-to-first-match.
- **Failed validation**: Pilot starts in unusable environment.

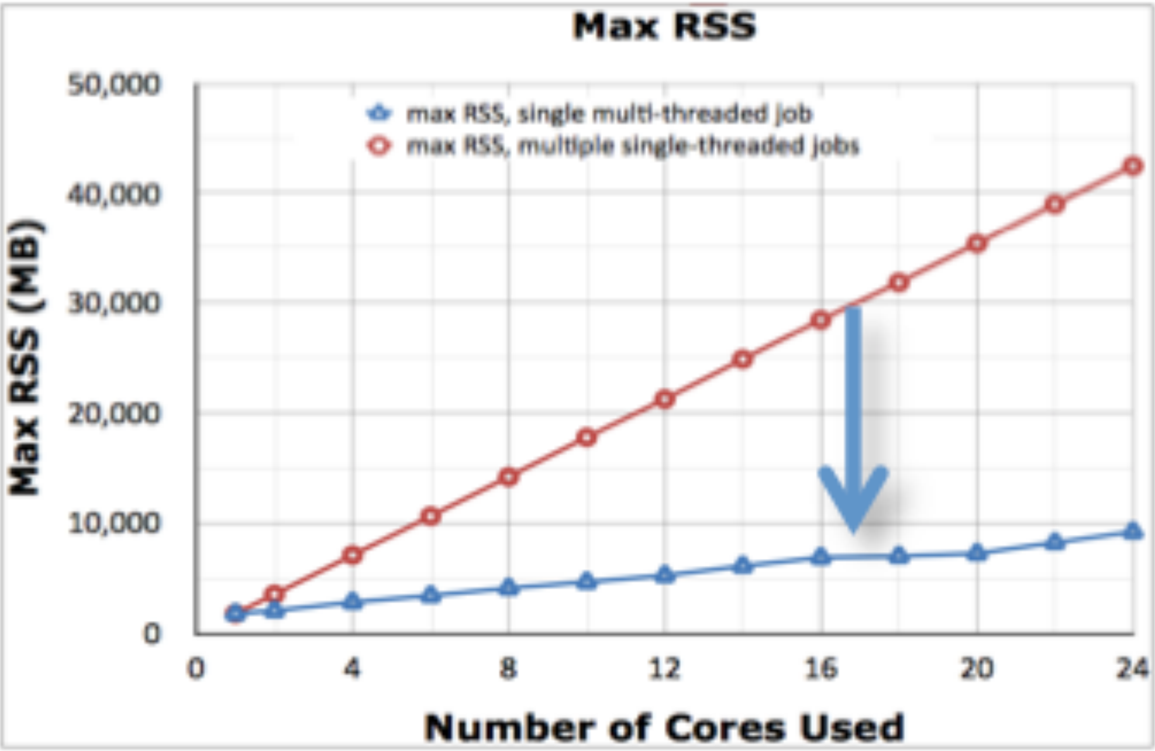
Payloads:

- **Job startup / shutdown**: Stage-in and -out. Exacerbated by short-running jobs
- **CPU efficiency**: Lots of IO per job (typically DIGI with pileup).
- **Algorithm performance**: Tasks that take longer than they should

Pilot efficiency considerations

- **Pilot scheduling efficiency is closely monitored. Generally ~90%**
  - Job pressure variation (e.g., contracting global pool) impacts efficiency
  - Mix of single and multi threaded jobs fragments pilots
- **Going more and more multithreaded is helping too**
  - 2015+ Tier-0 and data re-reco is multithreaded
  - 2016+ DIGI-RECO MC now transitioned to multithreaded
  - 2017+ GEN-SIM will be multithreaded
  - Legacy configurations to remain single threaded (in most cases). These will naturally reduce over time as analysis is more focused on 2016+ given large data samples accumulated this year

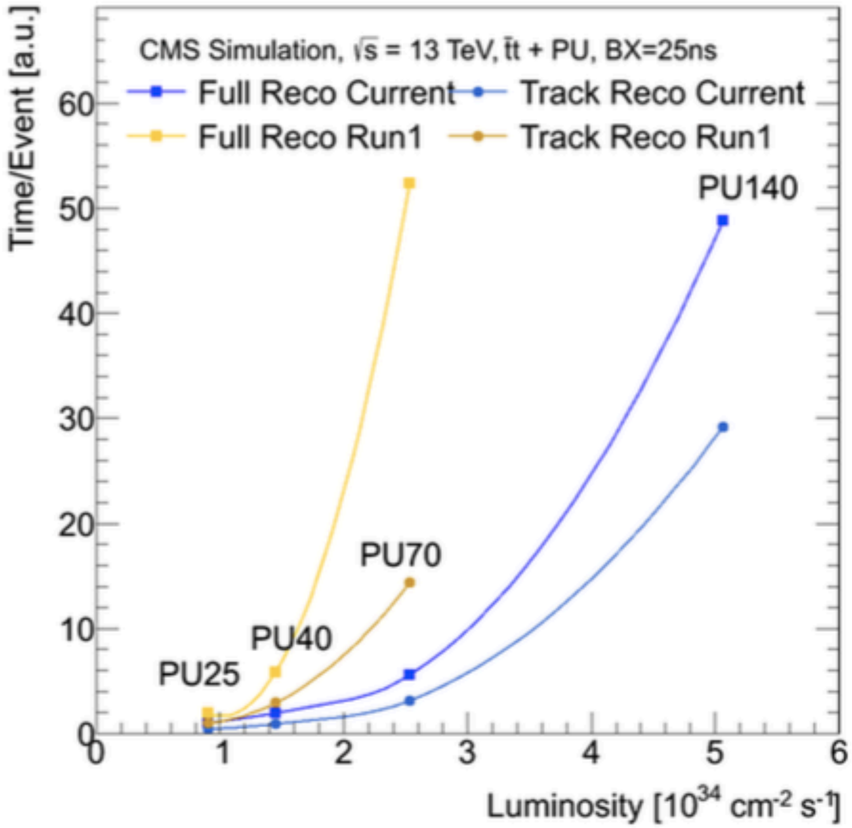
Mulithreaded framework in production for Run2



- Big challenge remains:
  - How do we upgrade all algorithms to be thread friendly
  - How do we teach developers to program in this new regime?

Code performance: Processing time scales drastically with event complexity

- Resources needed by combinatorically driven reconstruction algorithms increase non-linearly with interactions/crossing
  - Dedicated analyses of performance at high pileup resulted in scaling improved in recent years



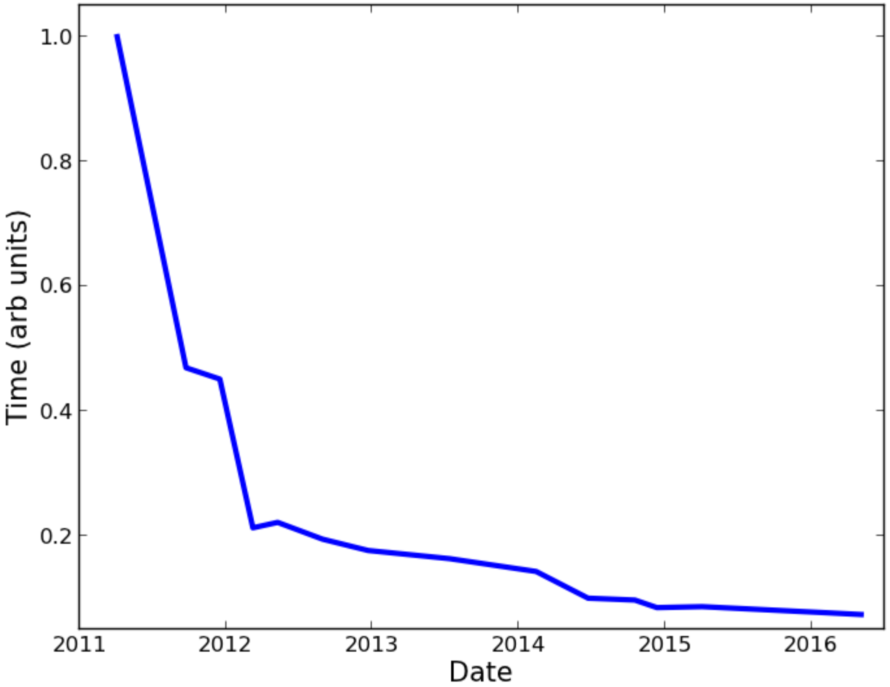
Constantly monitoring and improving CPU performance across our production software stack

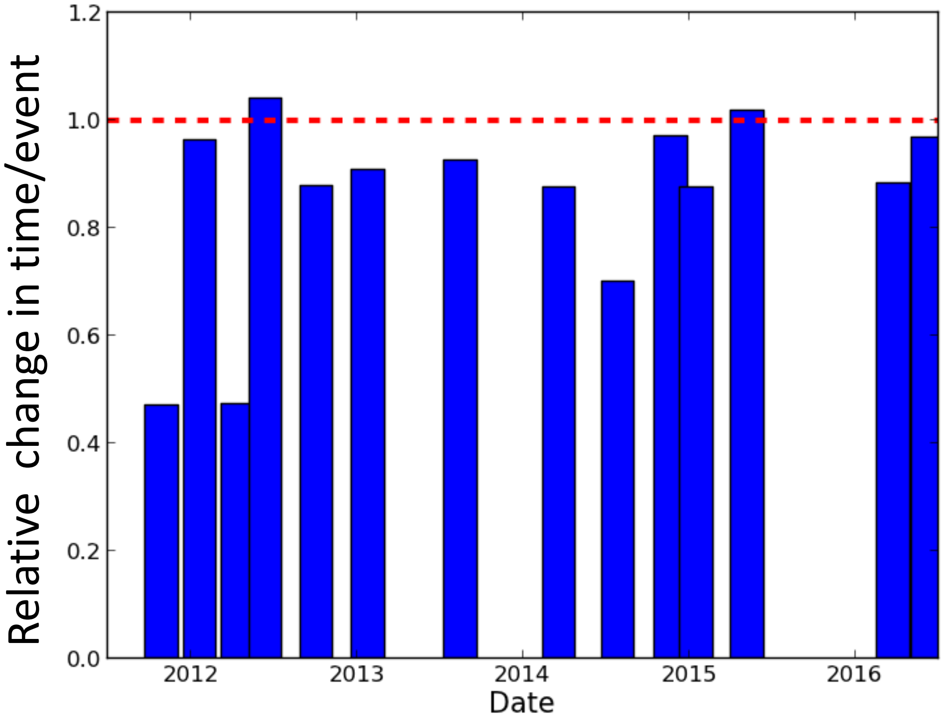
Recent targets in code performance

Large individual targets are gone. Tool like igprof particularly important to identify hot spots for investigation at this stage

- Redundant calculations: Remove
- Numerical precision: Use whats needed
- Math functions: VDT where possible
- Push persistency overhead away: Use original objects in algorithms
- Take advantage of C++11 constructs
- Vectorization: Ensure loops are vectorized by compiler (but we miss automatic methods)
- Stay on top of latest compilers: We aim to use gcc6 for 2017 production software

Reconstruction performance against high pileup data





Incremental progress continues through Run 2 while maintaining the physics

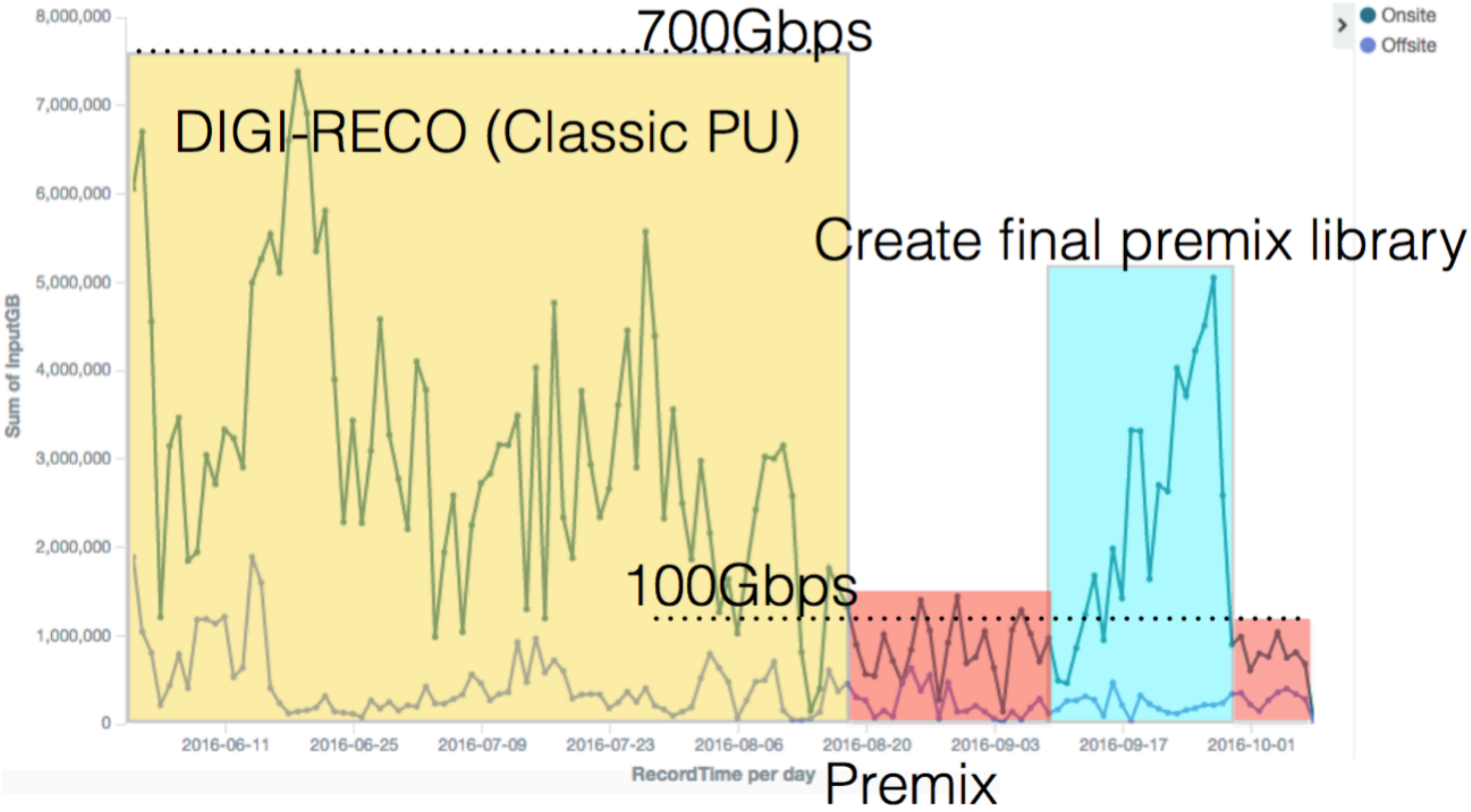
I/O impacts job performance

- More important to monitor threaded jobs:
  - Reads from ROOT soruces are serialized, so impact of any latencies scales with N threads.
    - Recently parallelized FlushBaskets saving ~10% of total time for processing Tier-0
- Time for random (ROOT) file reads can dramatically improve if file is written in optimal way (to let you read and uncompress just what you need)
- Impact of saturating site bandwidth on CPU efficiency is big
  - Improving I/O of pileup simulation via “premixing”

Pileup simulation: Premixing now deployed in CMS production

- Premixing concept: Instead of doing pileup simulation during DIGI-RECO processing, we do it once to make a “premixed data set”.
  - Process ~0.25B heavy I/O events in a single dedicated production instead of ~6B during DIGI-RECO campaign.
  - Combine premixed data set with hard scatter events during digitization processing
- After long validation and production integration/testing period, premixing is now deployed
- Results
  - Premixing reduces the I/O from pileup in digitization step by **40x**
  - The premixed data set is big (250M, 0.5 PB)
    - We do not have a copy of it everywhere. Often use AAA to read it
    - PU scenarios needed for special studies will use old DIGI-RECO method
  - **Large increase in DIGI job efficiency (even with remote reads)**

Effect of premixing on total I/O



Conclusion

- Continue to improve time needed to process CMS data and MC events through improvements on many fronts
  - Multicore, algorithm design, I/O considerations are recent important changes
- Monitoring is improving and helps us identify where hot spots are in real operations
  - Easy to follow performance according to type of workflow, site, individual request level
- 2017 will be a challenging year:
  - Pileup expected to continue to increase
  - Major detector upgrades new lots of new code going into production.

For more details see talks during CHEP

- Monitoring efficiency: Monday 16:30, Track 7
- Threading efficiently: Monday 11:30 , Track 2
- Premixing: Tuesday 14:00, Track 2
- Multicore pilots and scheduling: Monday 14:30, Track 3