Multi-core processing and scheduling performance in CMS

<u>José Hernández</u> (CIEMAT), Dave Evans (FNAL), Steve Foulkes (FNAL)

XIX International Conference on Computing in High Energy and Nuclear Physics

Motivation for multi-core processing

- Simplistic utilization of multi-core CPUs by HEP
  - Independent application processes per core executed each processing independent sets of events
- This model has been effective but we could better exploit the multi-core architecture
- RAM available in Worker Nodes is a limitation
- Experiment event processing code is memory hungry
  - Especially given increased number of collisions per event in LHC
- We might soon not be able to satisfy the job memory needs per core in the current single-core processing model in HEP
  - Most deployed WNs have up to 2 GB RAM/core
  - Event processing code straggling to keep below
  - Problems already in 2011 to use all cores at the Tier-0

Motivation for multi-core processing

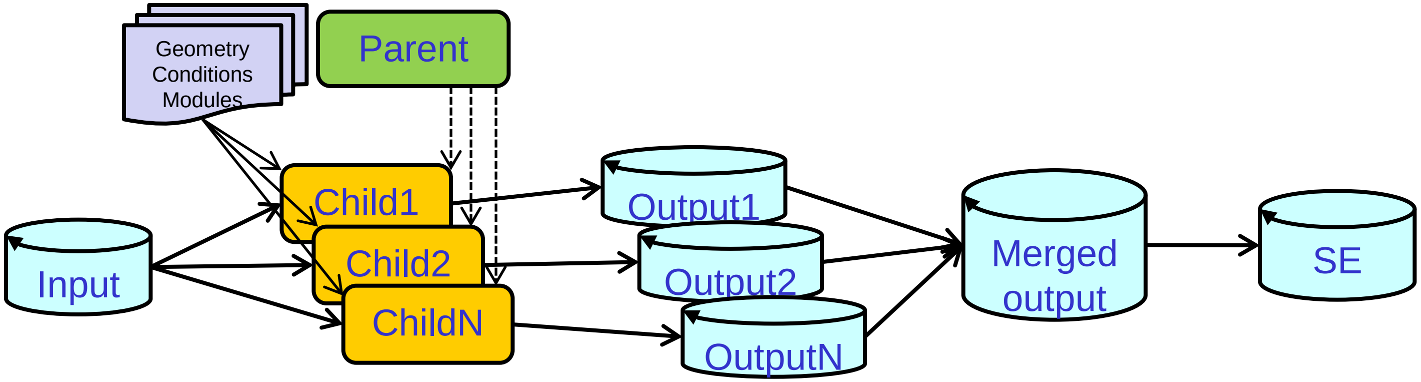
- An ever increasing number of independent and incoherent jobs running on the same physical hardware not sharing resources might significantly affect processing performance
- Experiment job management systems need to scale with the increasing number of jobs
  - CMS at the scale of ~200k jobs/day
  - Significant hardware and human resources
- It will be important to effectively utilize the multi-core architecture
- Need to efficiently use allocated cores since VO billed by all of them

Multicore processing in CMS

- Multi-core aware applications can improve memory sharing and processing performance
- Multi-core processing jobs share common data in memory, such us the code libraries, detector geometry and conditions data, resulting in a much lower memory usage than standard single-core independent jobs
- CMS has incorporated support for multi-core processing in the CMSSW event processing framework
- Initial simple approach: CMSSW forking
  - Main process forks a number of child processes
  - Parallelize event processing within the same job
- CMS is investigating as well the threading approach
  - Sub-event parallelization: use multiple cores per event
  - See contribution 194: Study of a Fine Grained Threaded Framework Design

CMSSW forking implementation

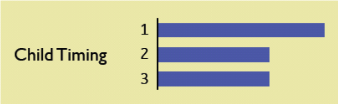
- Parent process reads configuration, loads processing modules, pre-fetches geometry, calibrations and other conditions data
- Parent forks a number of child processes
- Child processes share parent (read-only) memory
- Children process a fraction of the input file
- When child processes are done, parent merges results and stages out the output files into mass storage



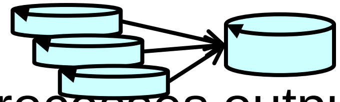
Forked multi-processing overheads

Expect some small overhead in current implementation of forked multi-processing w.r.t. single-core processing

- Processing time dispersion



  - Number of events to process by each child set up front
  - Will result in idle time of N-1 cores waiting for all cores to finish
- Merging of output files



  - Job wrapper needs to merge child processes output files
  - Local merging largely minimizes asynchronous merging present in all CMS data processing workflows
  - Implementation choice (it could be skipped or parallelized)
- Stage-out of output files into mass storage
- The parent process also consumes some RAM
- Processing dispersion and merging overheads could be overcome by using a more complex multi-processing scheme

Multicore scheduling in CMS

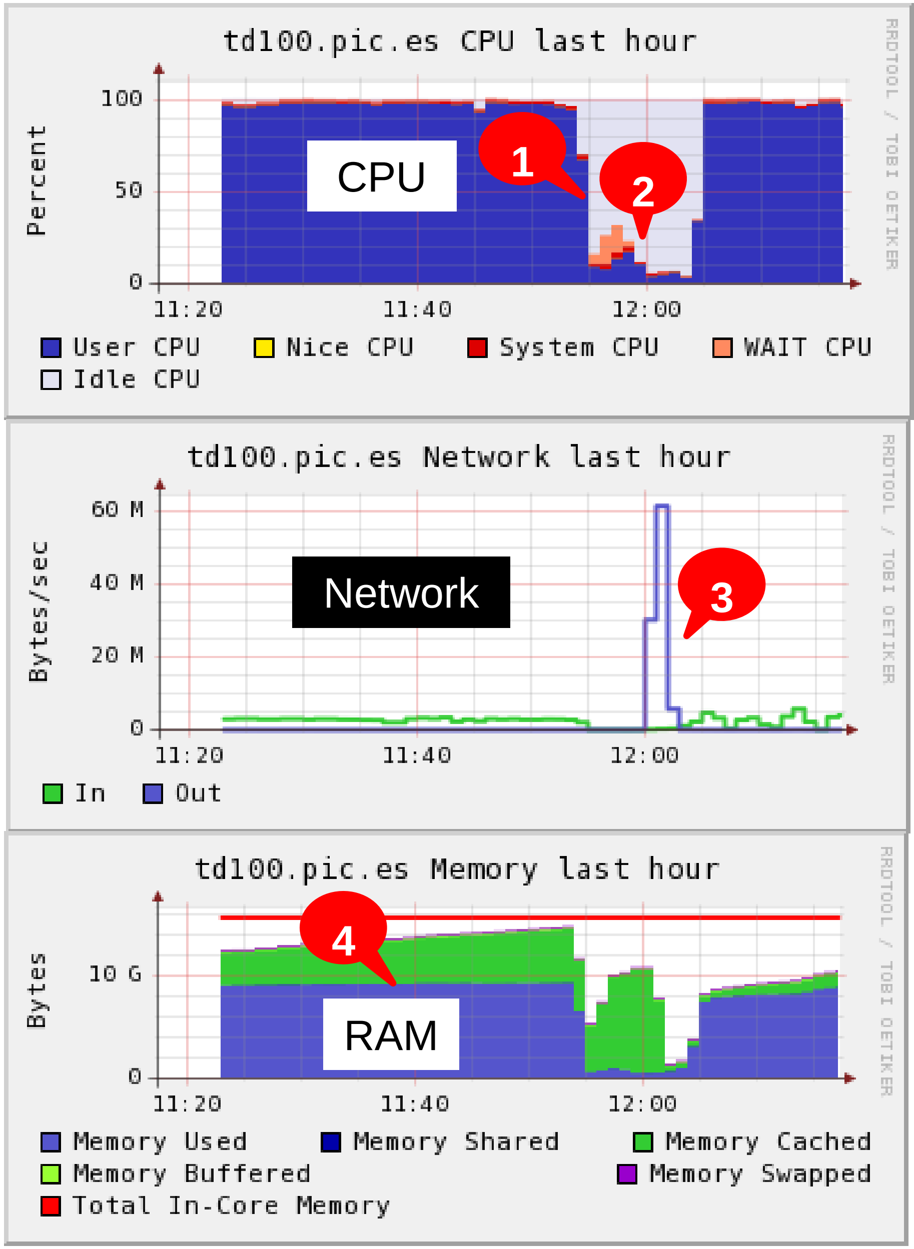
- Exploiting this new processing model requires a new model in computing resource allocation, departing from the standard single-core allocation for a job.
- The experiment job management system needs to have control over a larger quantum of resource since multi-core aware jobs require the scheduling of multiples cores simultaneously
- CMS has incorporated support for multi-core scheduling in its workload management system
- CMS has explored two approaches for multi-core allocation:
  - Dedicated resources of whole-nodes where all cores of a node are allocated to a multi-core job
    - Dedicated whole-node queues with few nodes at all 7 T1s
  - Dedicated queues that provide nodes with a fixed number of cores (not necessarily the whole node) from a shared farm
    - Shared queues at KIT Tier-1 and Purdue Tier-2

Multi-core allocation

- Whole-node queues was the initial approach in CMS
  - In the context of the WLCG whole-node task force
  - The idea was to share with other VOs these dedicated resources
  - Allows experiment to manage the whole node
  - Sites did not like partitioning of resources
- Queues that give access to nodes with a fixed number of cores from the shared farm
  - Shared resources with standard single core queues
  - LRMS drains nodes for multicore allocation
- Dynamic resource allocation
  - LRMS schedules a dynamic number of free cores
  - Jobs (or pilots) specify requirements (#cores, RAM, whole-node)
  - LRMS informs jobs of allocated number of cores
  - In line with recommendation of WLCG WM TEG
  - Shared resources with standard single core queues

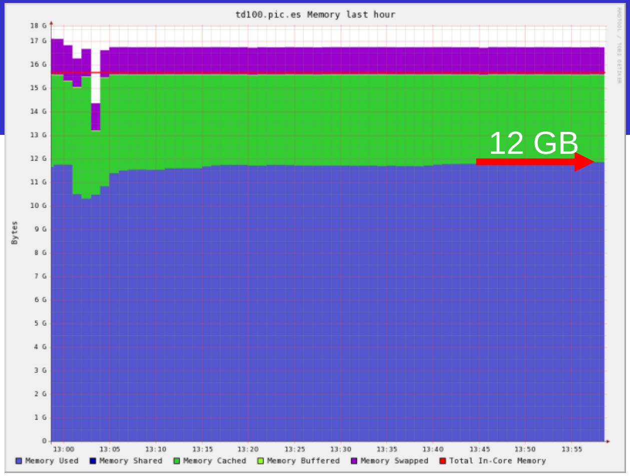
Multi-core CPU, RAM, I/O

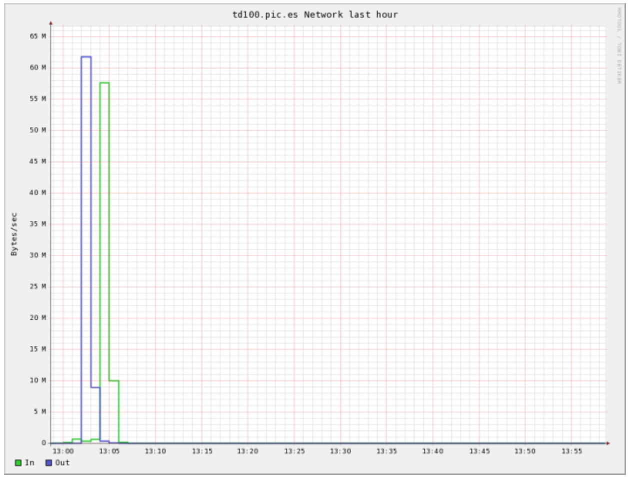
- Example typical 8-core reconstruction job
  - 7 outputs: AOD, RECO, DQM, 4 alcareco's
- 1 Processing dispersion
  - Small overhead (~1 min)
- 2 Merging output files
  - Small overhead (~5 min)
- 3 Stageout output file (~2 min)
  - Fast (few GB/min)
- 4 ~9 GB RAM used by the machine

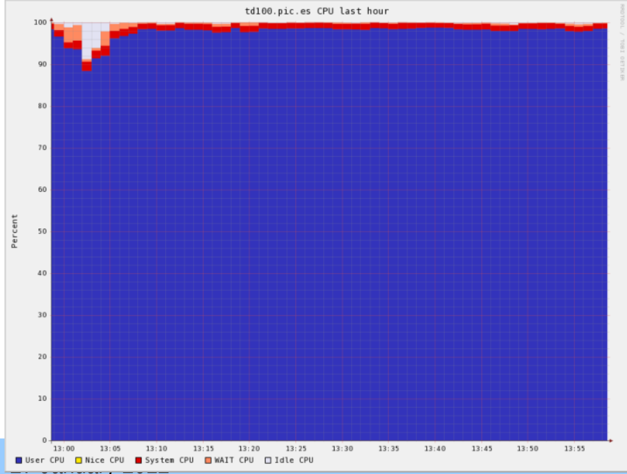


8 x jobs single core

- Running 8 simultaneous single-core jobs for comparison
  - Fair comparison with one 8-core job in the same machine
- Higher memory consumption (~25%)
  - 12 GB RAM used by machine
  - Machine even uses some swap
- Almost no idle CPU time on the cores of the node
  - Small dips when a job finishes
- No overhead by local merging (but merging has to be asynchronously run afterwards)





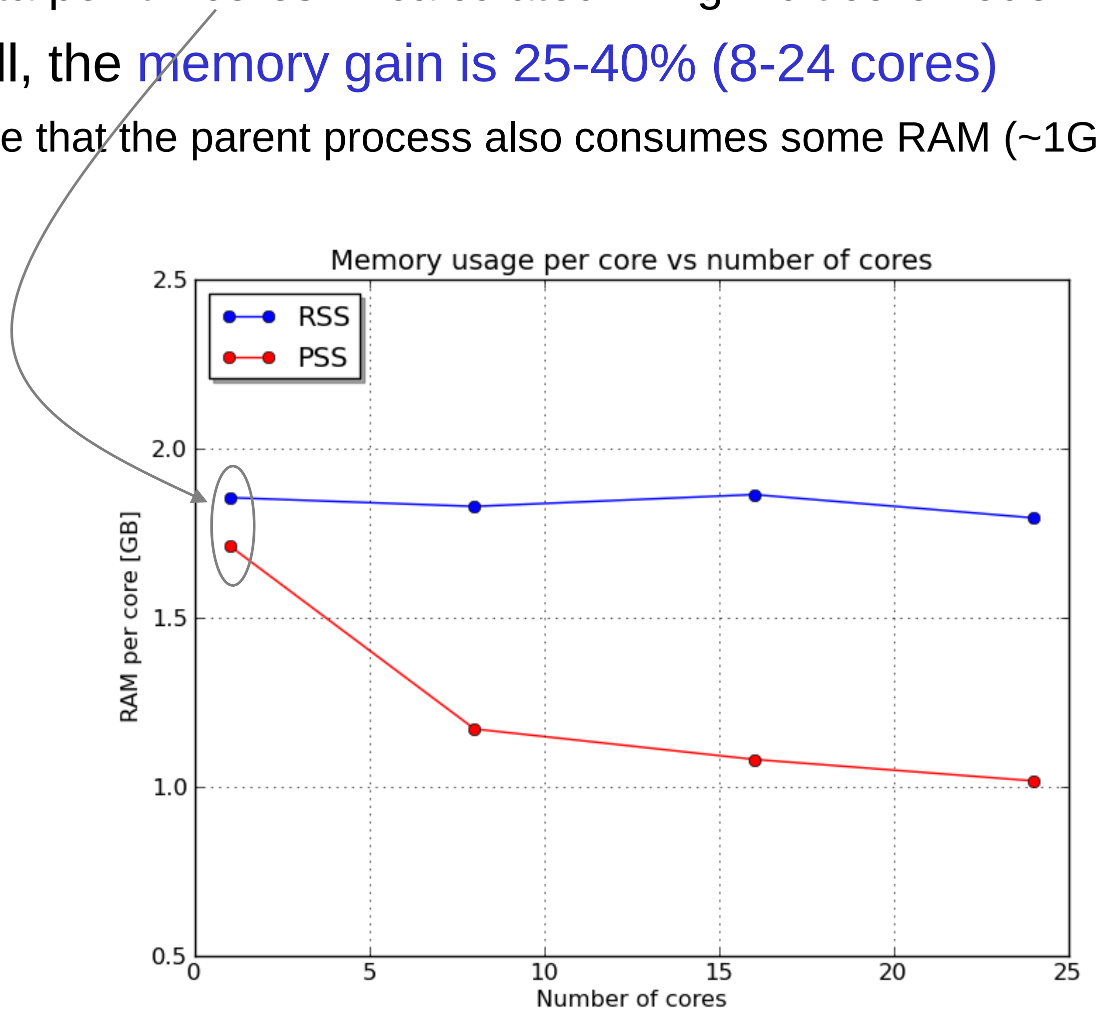


PSS memory

- Because large portions of physical memory are typically shared by processes, the standard measure of memory resident set size (RSS) will significantly overestimate memory usage
- PSS (Proportional set size) instead measures each application's "fair share" of each shared area to give a realistic measure
- The PSS of a given process (or sub-process of a multi-core process) depends on the other processes running
- The CMS framework measures the PSS value of each sub-process at the peak RSS value
- Good indicator of memory consumption by the multi-core application

PSS per processing child

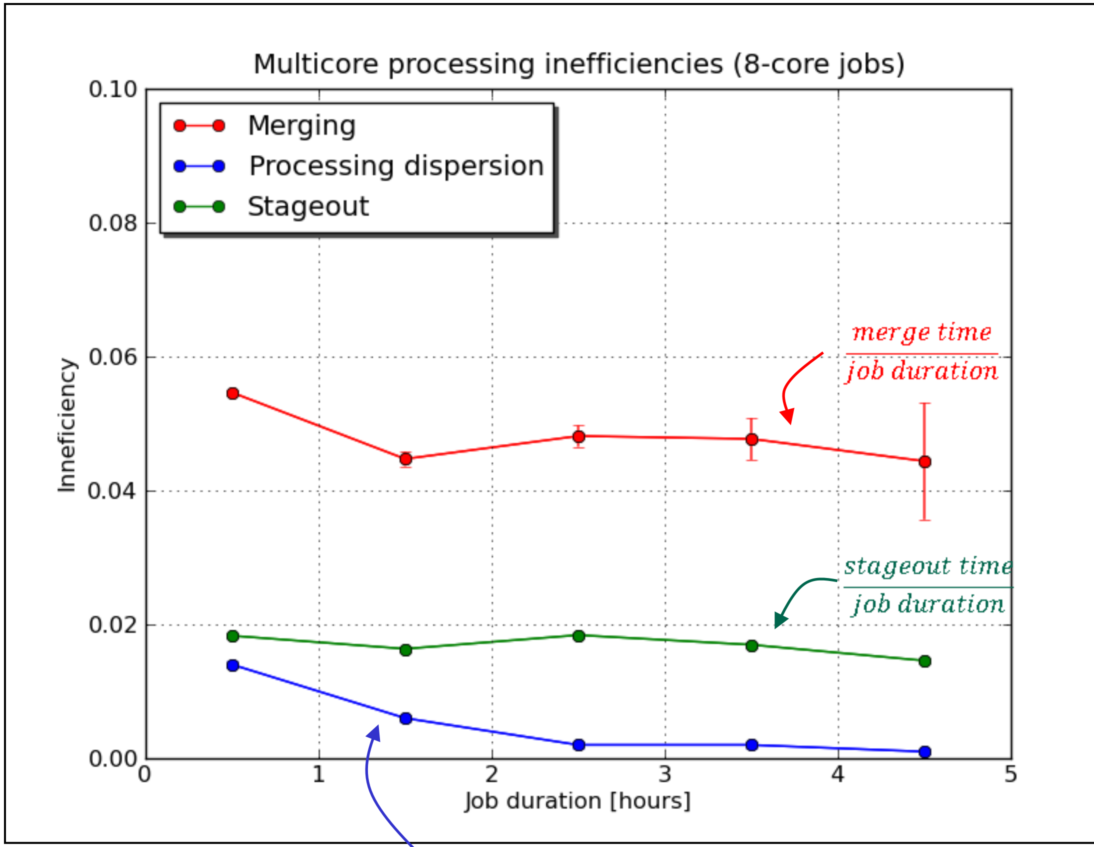
- Significant PSS reduction by running the application multi-core
  - Data point Ncores=1 calculated filling multicore node with single-core jobs
- Overall, the memory gain is 25-40% (8-24 cores)
  - Note that the parent process also consumes some RAM (~1GB)



Multicore processing inefficiencies

Idle time in cores due to:

- Processing dispersion due to fluctuations in event processing time
  - Parent process waiting for all sub-processes to finish
  - Small relative inefficiency and decreases with job length
- Merging of output files from each sub-process
  - ~Constant with job length
- Stageout of merged output files
  - ~Constant with job length

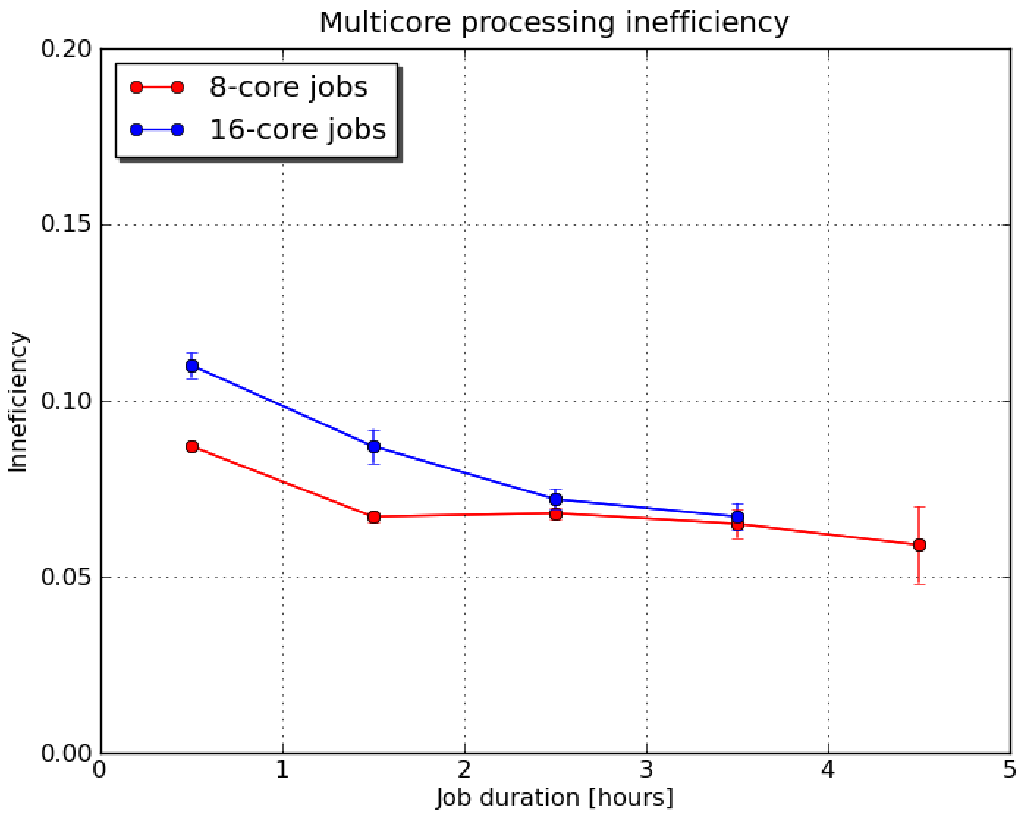


$$1 - \frac{Sum\ children\ processing\ time}{(Max.\ child\ processing\ time) \times (Number\ of\ children)}$$

Multicore processing overhead

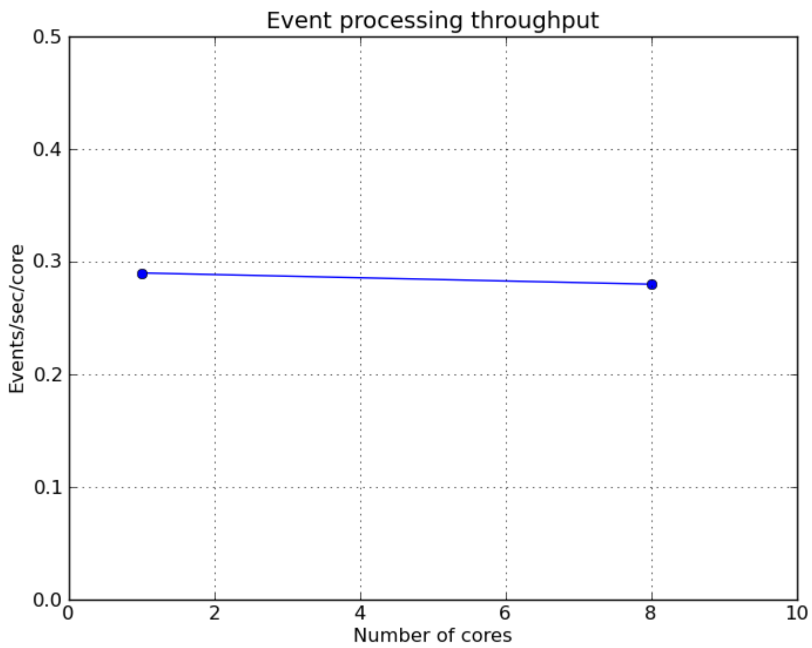
Adding up processing dispersion, merging and stageout overheads:

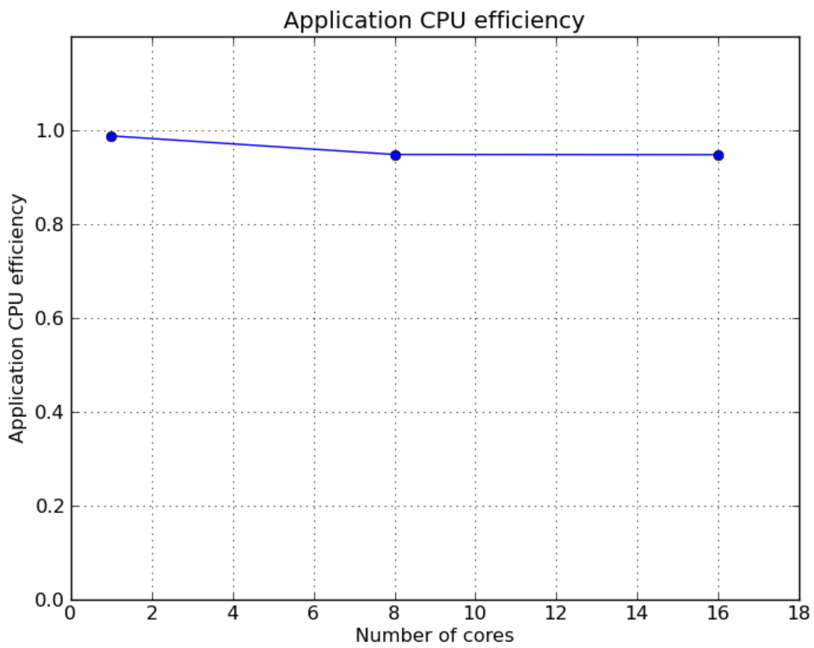
- Asymptotical overhead of ~6% for sufficiently long jobs (due to merging)
  - Typical production jobs are ~8-12 hour long
  - Merge could be skipped or parallelized
- Slightly higher overhead for short jobs when larger number of cores used
  - Due to higher processing dispersion



Multicore throughput

- Same throughput (events/sec/core) at steady processing for single- and multi-core processing modes
- ~Same high CPU efficiency (CPU time over wallclock time)





Prospects

- The CMS workload management system fully supports multi-core scheduling and execution
- Extensively tested but only at a modest scale
  - Few dedicated whole-nodes at all 7 Tier-1 sites and some access to shared resources with single-core jobs (at Purdue and KIT)
  - Up to 100 multi-core jobs running in parallel
- Plan to increase the resources available for multi-core processing at the Tier-1s
  - Shared with single-core jobs (LRMS allocates N cores)
- Potential gain as well for the Tier-0 where resources in 2011 were limited by memory consumption

Summary and conclusions

- Significant memory gain (~25-40% for 8-24 cores)
  - Important to keep reconstruction application memory footprint below 2 GB/core
- Small CPU overhead in current implementation of multicore processing
  - ~ 6% for > 2-hour long jobs
  - Essentially due to the merging of output files from each sub-process
- Merging step in the reconstruction workflow very much suppressed
- Number of processing jobs very much reduced
  - Allows to scale down our WMS
- CMS ready to go multi-core for data processing workflows