Simulating Diverse HEP Workflows on Heterogeneous Architectures

Charles Leggett, Illya Shapoval *(on behalf of the ATLAS collaboration)*
Marco Clemencic *(LHCb)*, Chris Jones *(CMS)*

ACAT 2019 Saas Fee, Switzerland
Mar 13 2019

Next Generation HPC Architectures

- In the next generation of supercomputers we see extensive use of accelerator technologies
  - Oak Ridge: **Summit** (2018)
    - 4608 IBM AC922 nodes *w/* 2x Power9 CPU
    - 3x NVIDIA Volta V100 + NVLink / CPU
  - Texas: **Frontera** (2019)
    - 8064 x2 Xeon
    - "single precision GPU subsystem"
  - Argonne: **Aurora A21** (2021)
    - exascale
    - "novel architecture" -> maybe CSA?
  - LLNL: **Sierra** (2018)
    - 4320 IBM AC922 nodes *w/* 2x Power9 CPU
    - 2x NVIDIA Volta V100 + NVLink / CPU
  - LBL: NERSC-9 "**Perlmutter**" (2020)
    - AMD Epyc "Milan" x86 only nodes + mixed CPU / "next gen" NVidia GPU
  - Similarly in Asia and Europe
- In order to meet the HL-LHC computing requirements, we need to use all available computing resources, or cut back physics projections
- **US funding agencies have indicated that we will not be able to get allocations if our code does not make use of accelerator hardware**

Offloading HEP Software To Accelerators

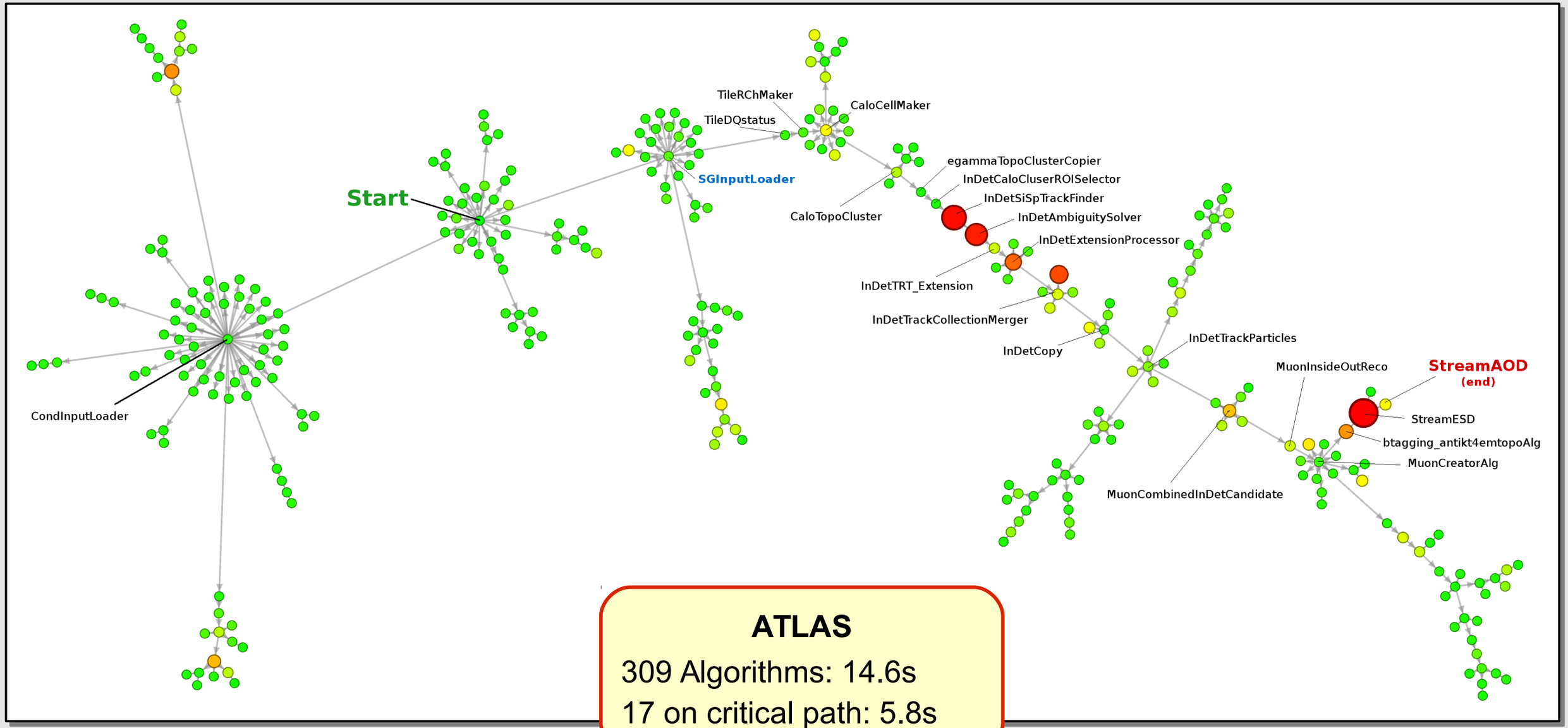
- In general, very little HEP software has been coded to run on accelerators
  - mostly tracking
  - some Geant4 EM and neutral processes
  - calorimeter cluster seeding
  - most HEP codebases don't parallelize easily
- Extensive work is being done to rewrite certain Algorithms making use of machine learning technologies
  - not easy, and time consuming
- Before expending vast resources recoding, it is essential to understand *how much* actually needs to be rewritten to make use of accelerators
  - can we identify critical bottlenecks?
- We can simulate HEP workflows and see what kind of Algorithms are most beneficial to offload

Understanding HEP Workflows

- We selected several standard reconstruction workflows from ATLAS, CMS and LHCb
  - Algorithm data interdependencies, control flow and timings have been extracted from actual data
- Ran simulation using Gaudi Avalanche task scheduler, with artificial CPU Crunchers instead of real Algorithms, allowing maximal concurrency of all Algorithms
- Generated a precedence trace, showing execution path of scheduler through Algorithm sequence
- Analyzed graph to identify **critical path**
  - Longest path through the graph, with run times taken as node weights
  - Algorithms that, with sufficient concurrency, determine event processing time
- Used critical path to determine which Algorithms to simulate offloading to accelerator. I/O is not offloaded

Workflow	Total Algorithms	Critical Path Algorithms	Critical Path without I/O
ATLAS	309 : 14.6s	20 : 8.33s	17 : 5.78s
ATLAS high-μ	309 : 127.3s	32 : 95.7s	29 : 85.7s
CMS	707 : 13.4s	147 : 8.38s	145 : 7.32s
LHCb	282 : 491ms	13 : 259ms	11 : 234ms

ATLAS Precedence Trace

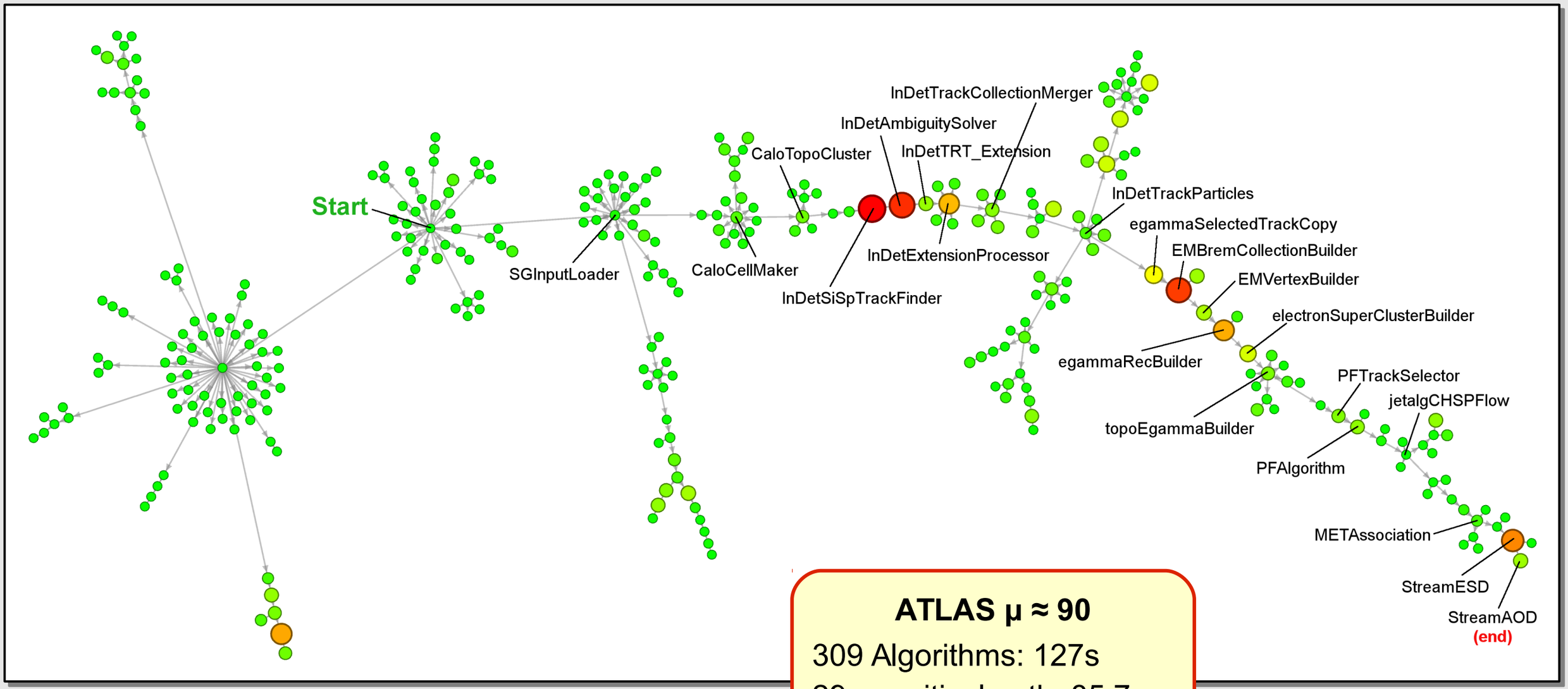


ATLAS

309 Algorithms: 14.6s

17 on critical path: 5.8s

ATLAS High μ Dataset Precedence Trace

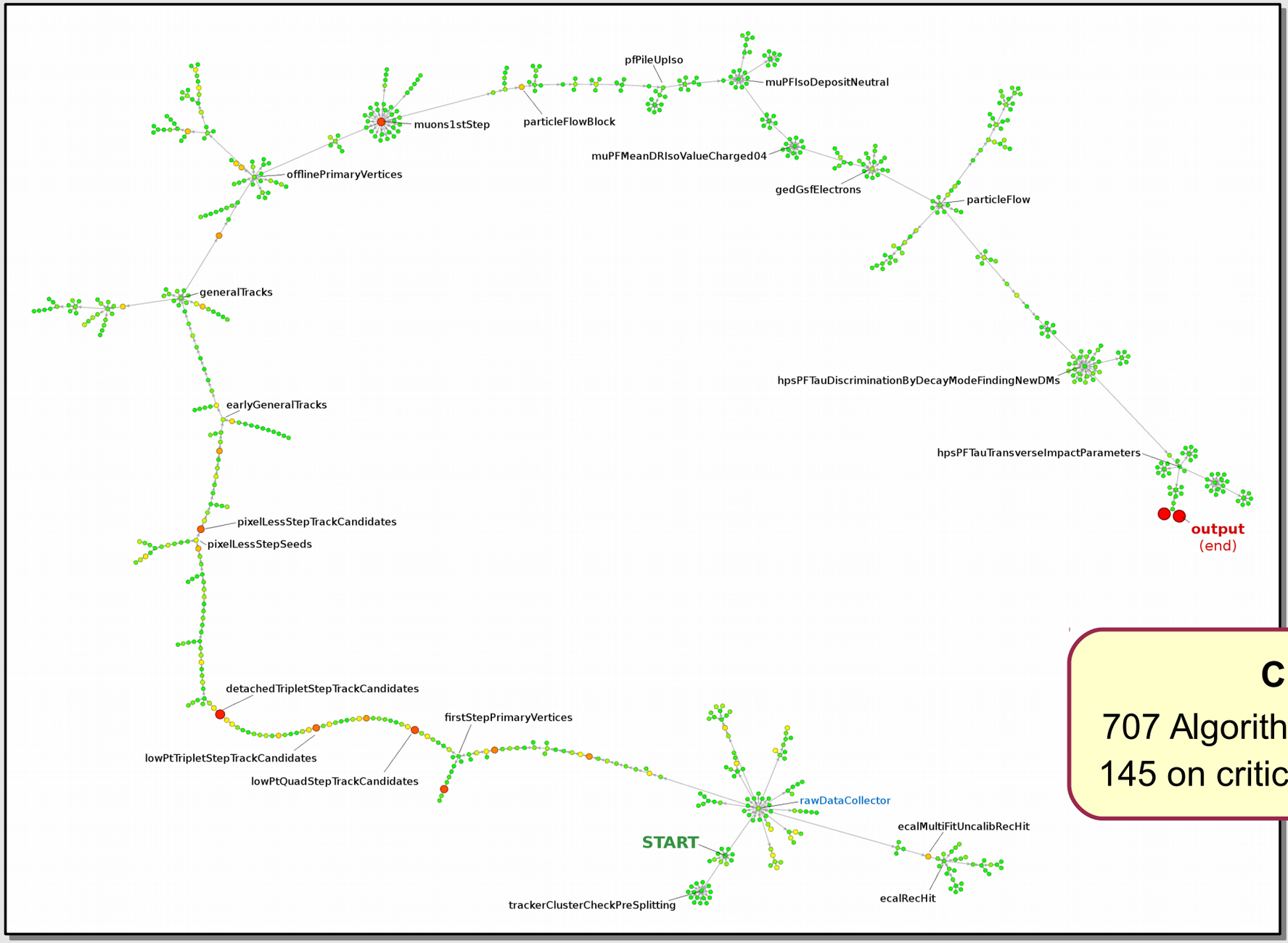


ATLAS μ ≈ 90

309 Algorithms: 127s

29 on critical path: 85.7s

CMS Precedence Trace

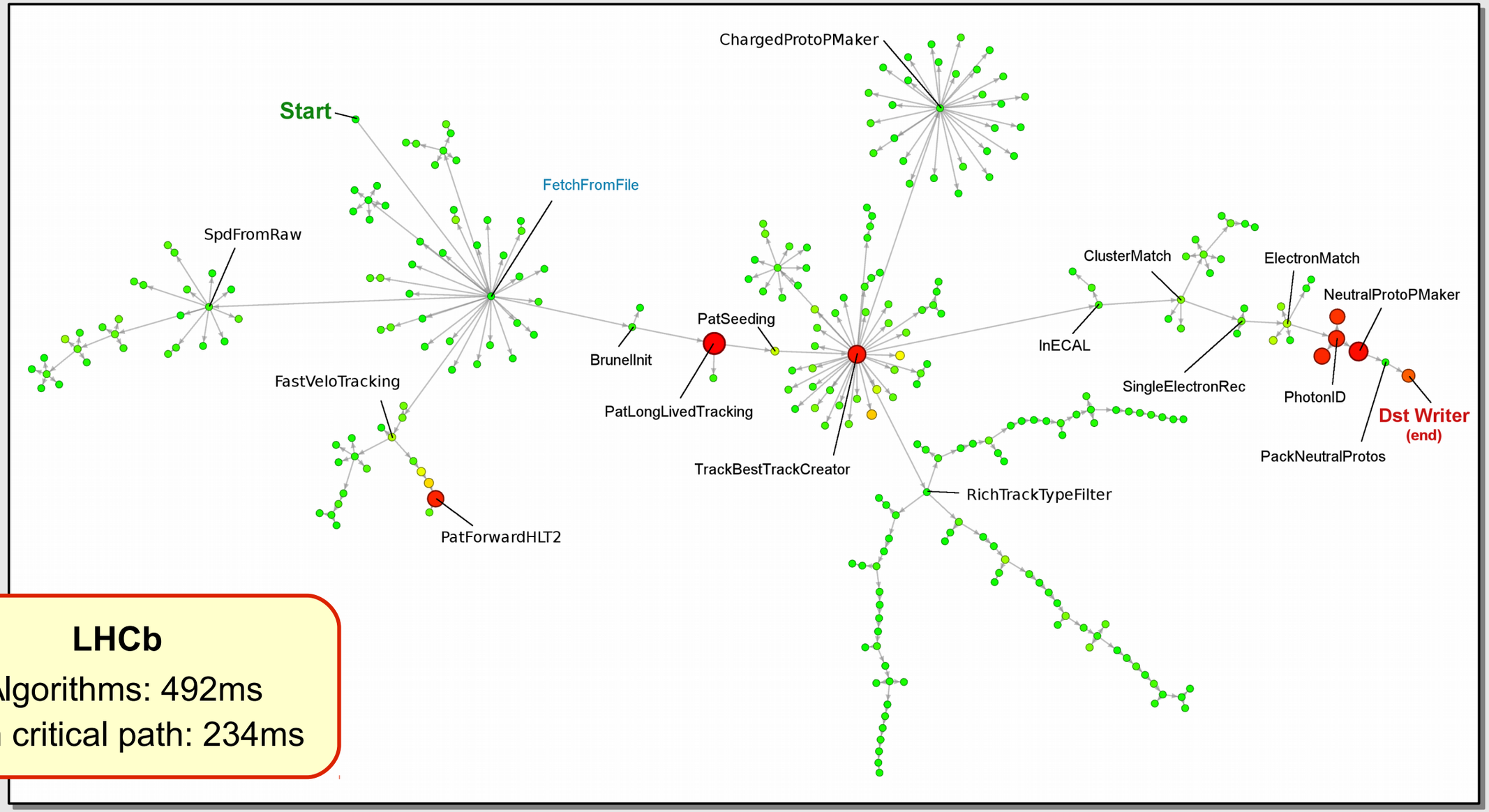


CMS

707 Algorithms: 13.4s

145 on critical path: 7.32s

LHCb Precedence Trace



LHCb

282 Algorithms: 492ms

13 on critical path: 234ms

Algorithm Offloading

- An Algorithm that offloads data to an external resource blocks its software thread
  - allow blocking thread to be pre-empted and displaced from the linux kernel run queue until it wakes up
  - hide latency by scheduling another thread if one is available
  - oversubscribe the scheduler with more threads than available hardware threads
  - for offline processing, **event throughput** is the only metric that matters

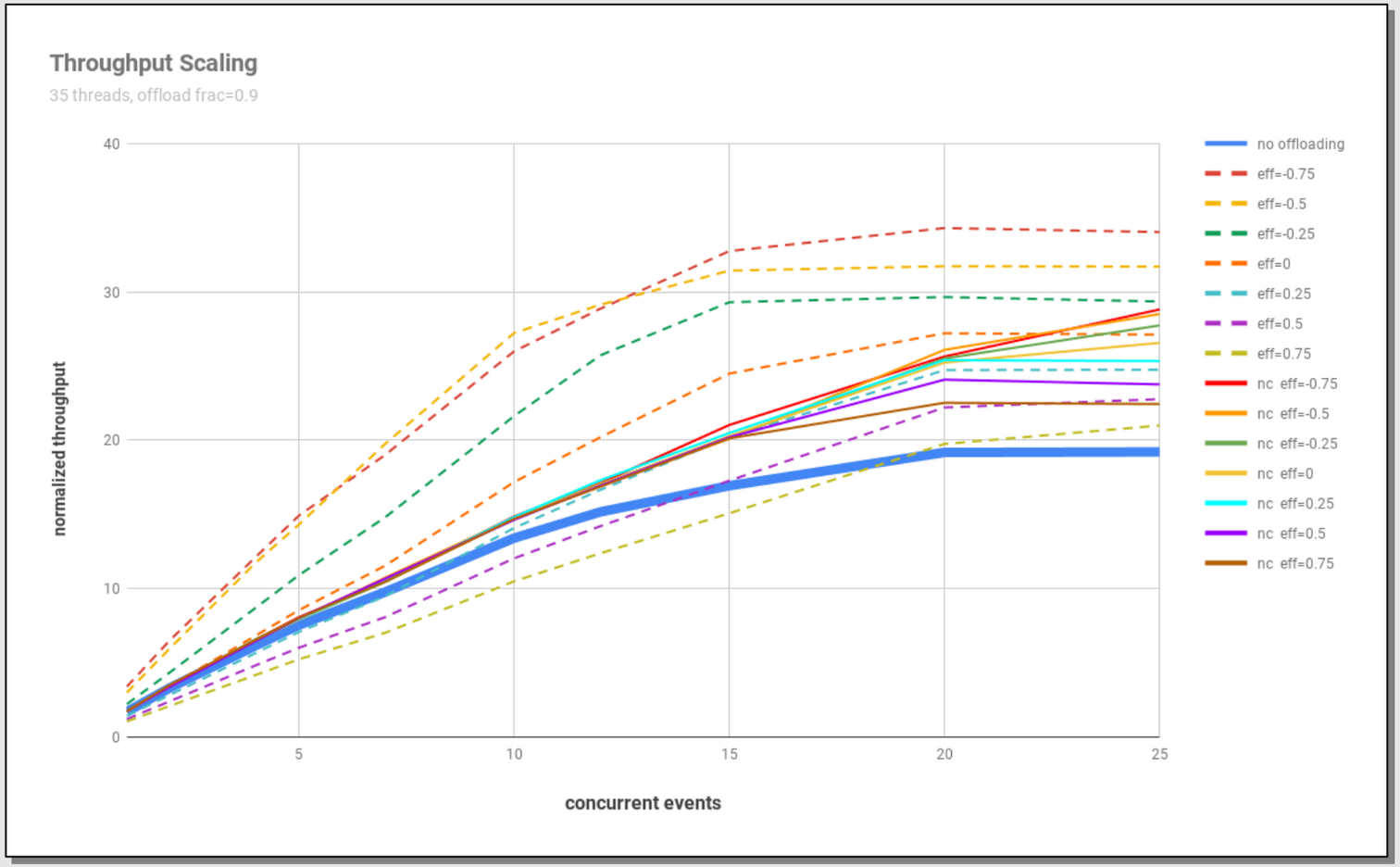
- Model offloading by modifying runtime t_{orig} of the Algorithm with 3 parameters:
  - fraction (*frac*) of Algorithm runtime that can be offloaded
  - efficiency (*eff*) of running offloaded part on accelerator (does it run faster or slower?)
  - extra time (t_{extra}) to transfer data to/from accelerator
  - the CPU will then run for t_{cpu} and the accelerator for $t_{offload}$

$$t_{cpu} = t_{orig} * (1\text{-}frac) \qquad t_{offload} = t_{orig} * frac * (1+eff) + t_{extra}$$

- Actual offload simulation performed by calling *sleep*
  - linux kernel does the rest for us

Offloading Critical or Non-Critical Path Algorithms

- Choosing which Algorithms to offload can be critical
- We can measure the throughput of the job varying the offloading fraction and efficiency

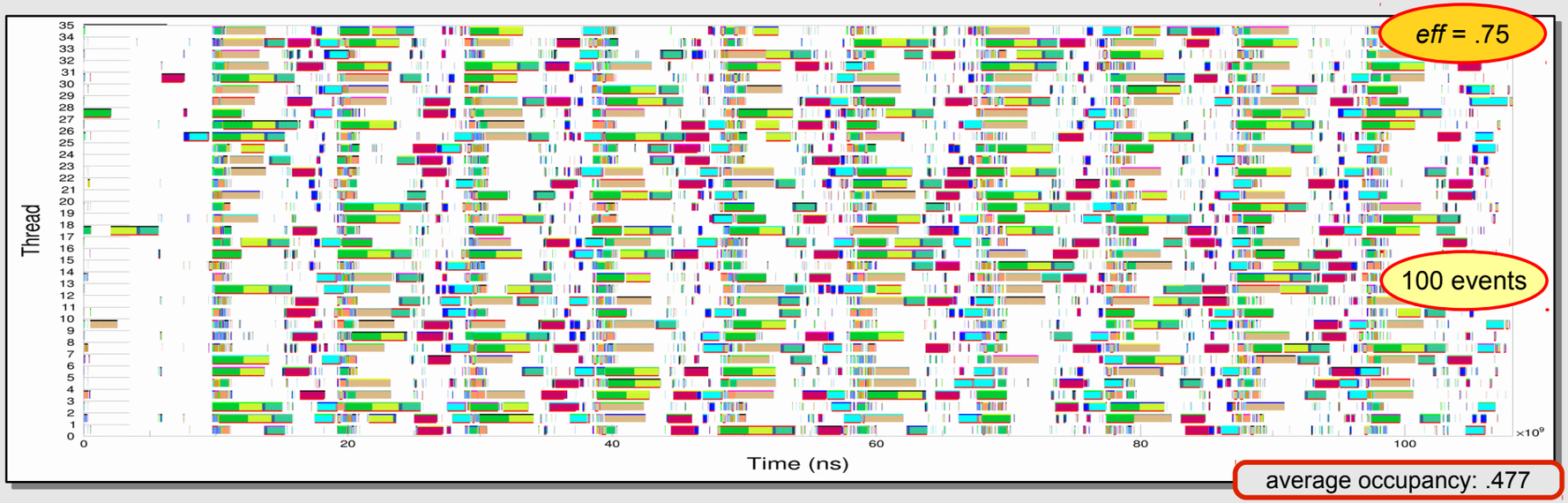


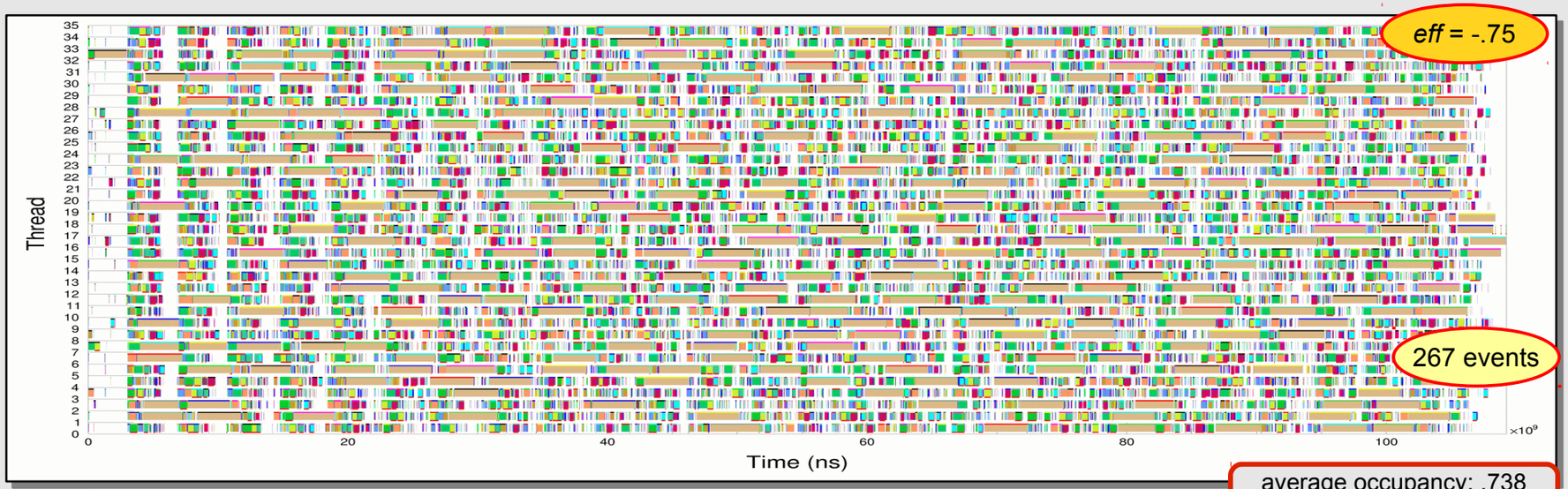
- If the accelerator takes much longer to execute the Algorithm than the CPU, it has the effect of lengthening the critical path. This can be overcome by increasing the number of concurrent events.
  - this may be limited by other system resource constraints
- While the actual algorithmic content of the Algorithm will ultimately decide whether it can be usefully offloaded, knowing that offloading Algorithms on the critical path has a larger impact on throughput will reduce the number of Algorithms to manually inspect

Comparison of Changing Accelerator Efficiency

- Offload Algorithms on the **critical path**
- Does it matter if Algorithms don't run much faster on the accelerator?
- Decreasing the accelerator efficiency (runs faster on accelerator) has the effect of increasing the occupancy, and decreasing the length of the critical path
  - throughput 2.7x higher

threads: 35
concurrent events: 10
offload frac: 0.9
offload eff: 0.75 -> -0.75





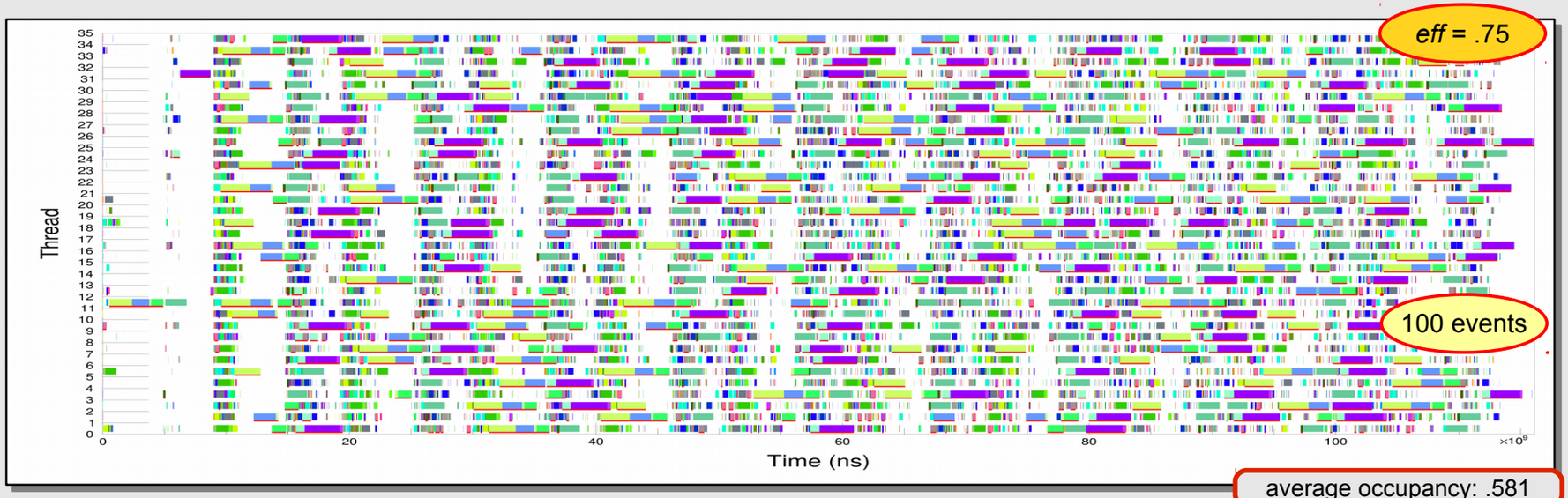
hardware: 2x Intel Xeon CPU E5-2630 v4
10 physical cores / CPU, HT enabled

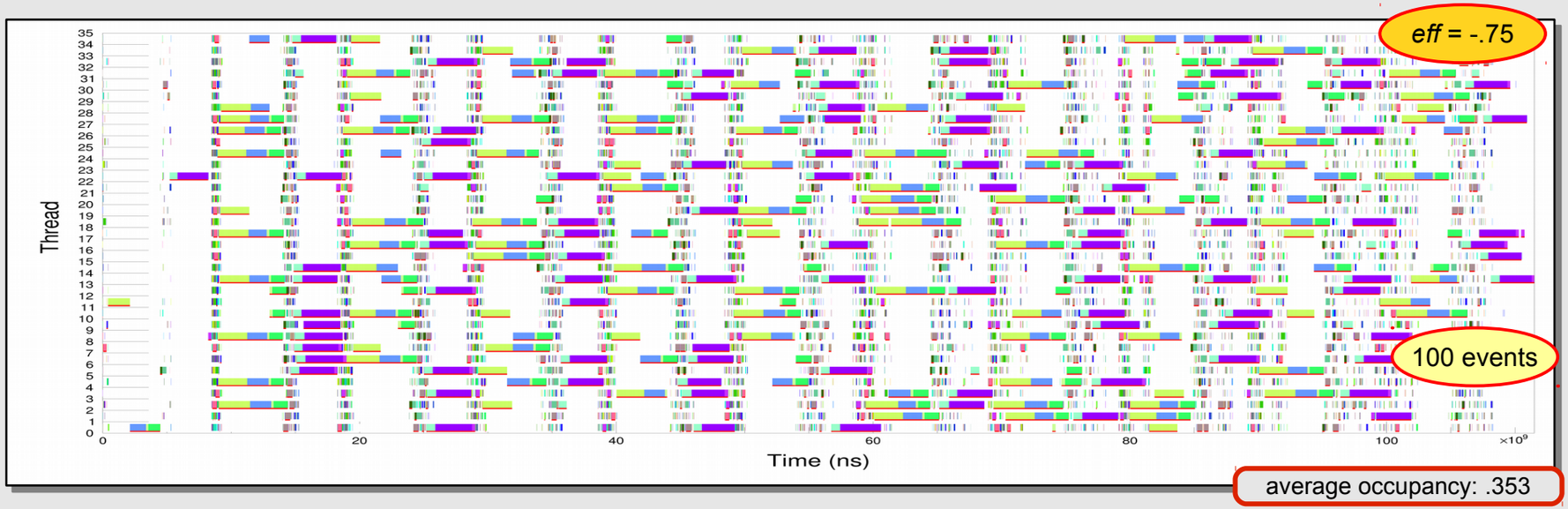
Offloading Algorithms not on Critical Path

- Offloading Algorithms **not** on the critical path, with different accelerator efficiencies

- Total throughput is comparable, but one has significantly higher occupancy than the other

threads: 35
concurrent events: 10
total events: 100
offload frac: 0.9
offload eff: 0.75 -> -0.75



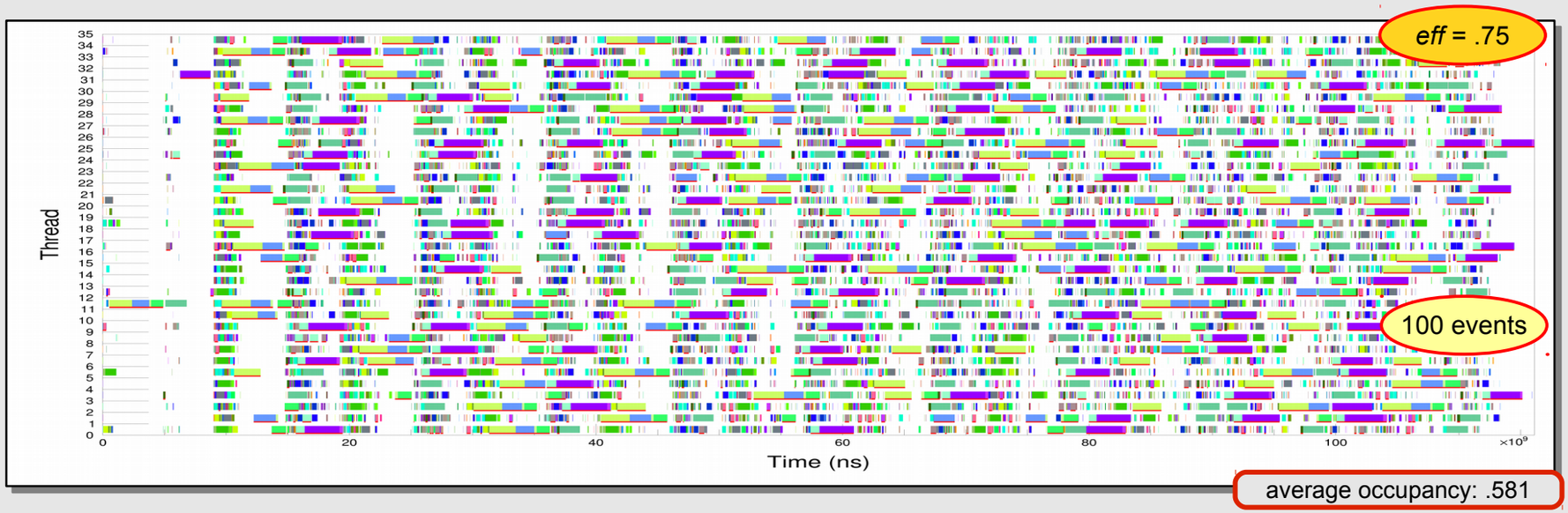


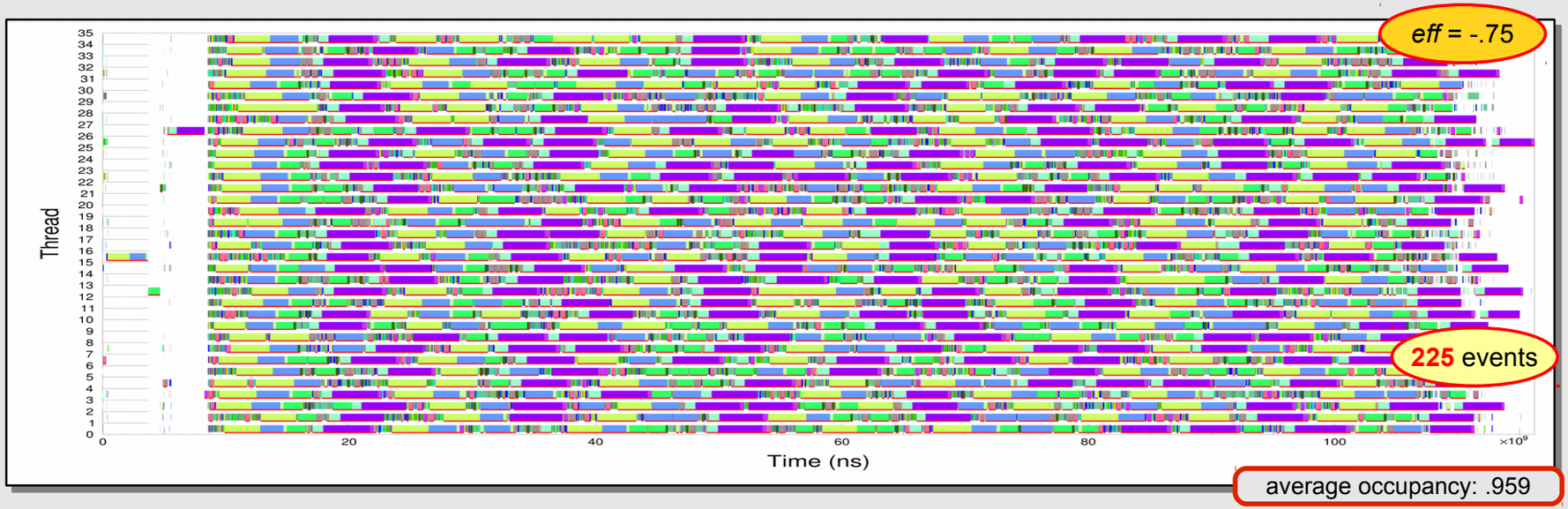
Offloading Algorithms not on Critical Path

- Offloading Algorithms **not** on the critical path, with different accelerator efficiencies

- Total throughput is comparable, but one has significantly higher occupancy than the other
  - can take advantage of low occupancy to schedule additional concurrent events, maximizing throughput

threads: 35
concurrent events: **40**
total events: **225**
offload frac: 0.9
offload eff: 0.75 -> -0.75

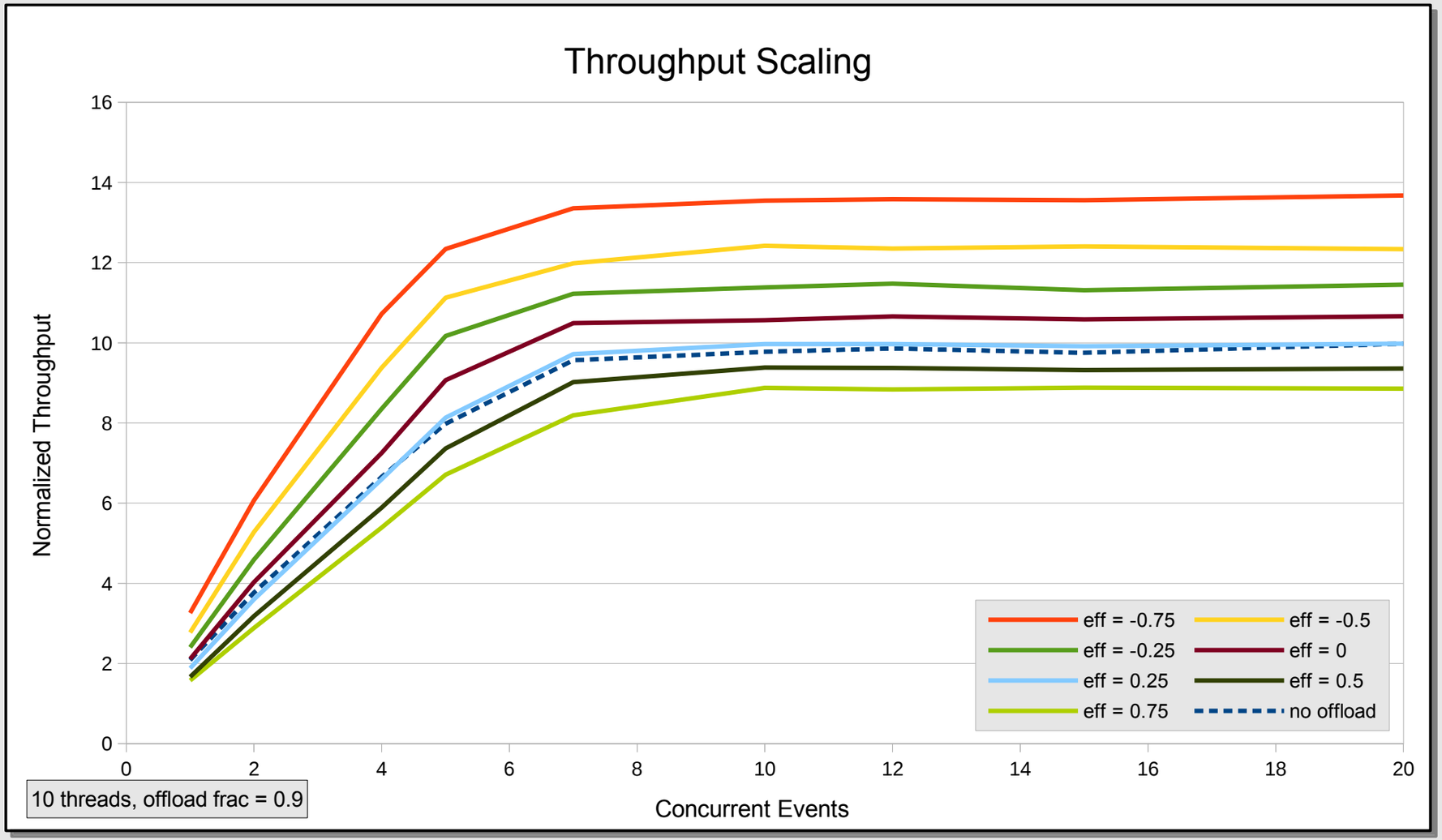




Oversubscription

- Running with only as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are often idle

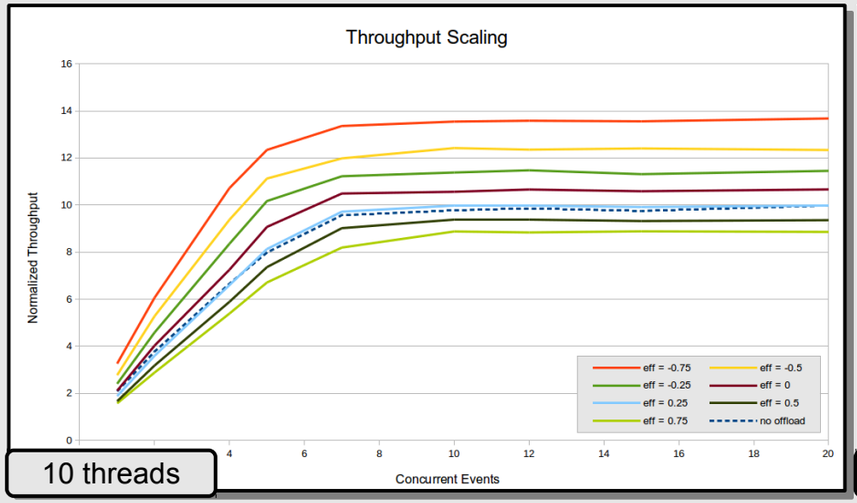
- offload only Algorithms on the critical path, except those that do I/O



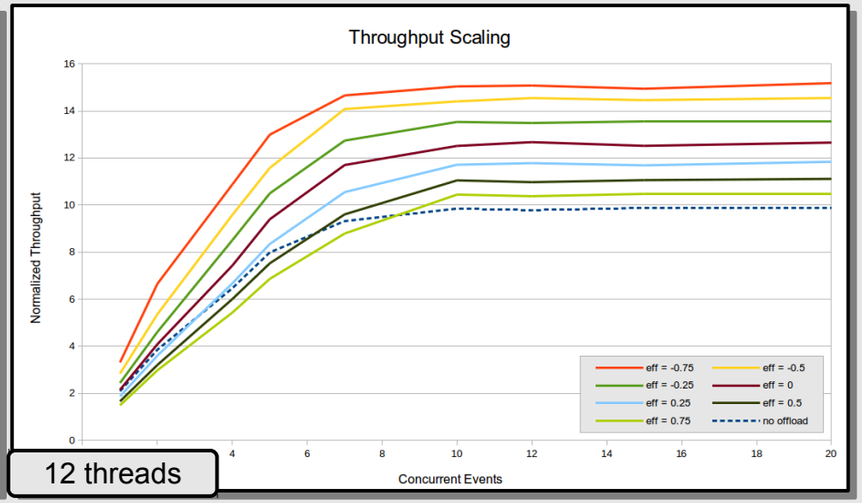
use **taskset** to limit execution to 10 physical cores, occupying all non-HT cores on one physical CPU

Oversubscription : ATLAS

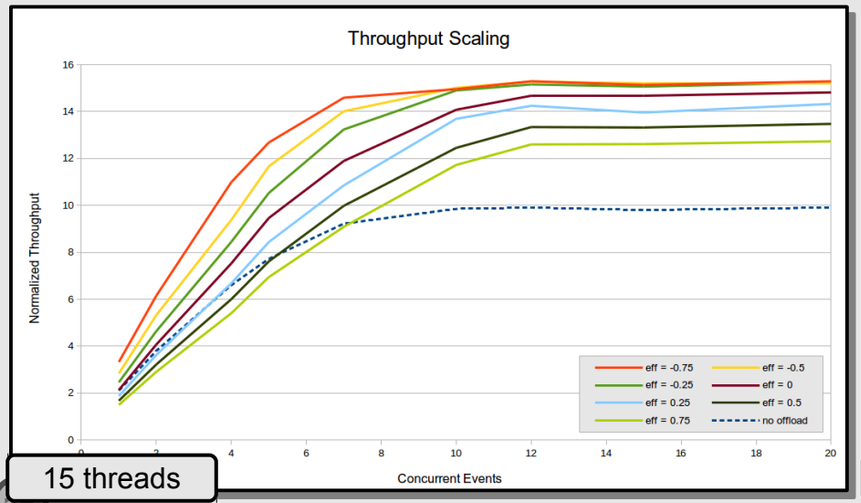
- Running with as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are idle
- We can **oversubscribe** the CPU with more threads to maximize throughput
- This may require increasing the number of concurrent events depending on available concurrency to get maximum throughput



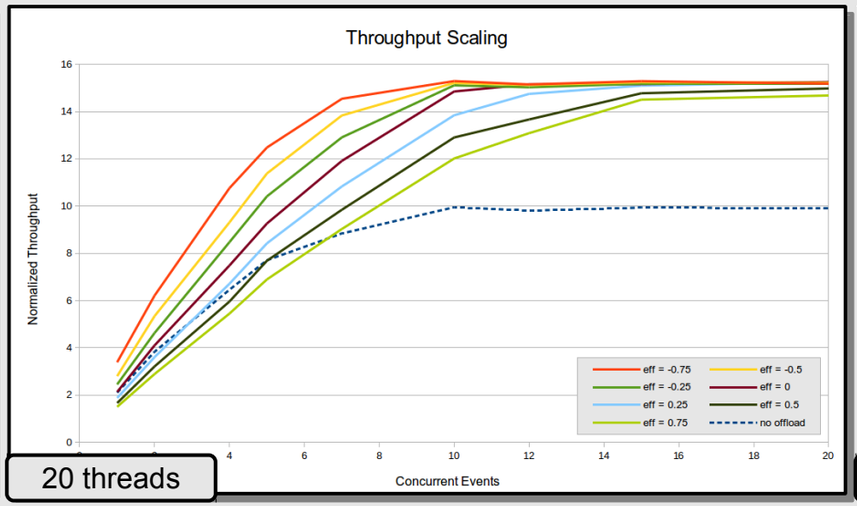
10 threads



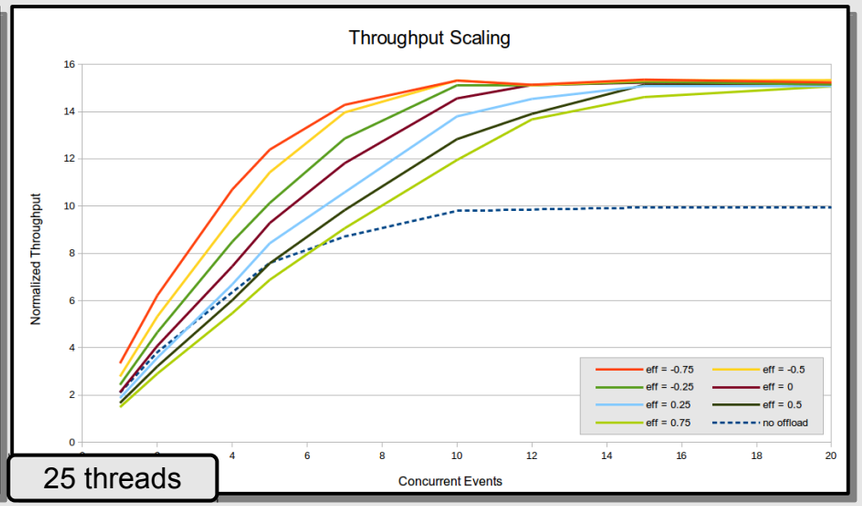
12 threads



15 threads



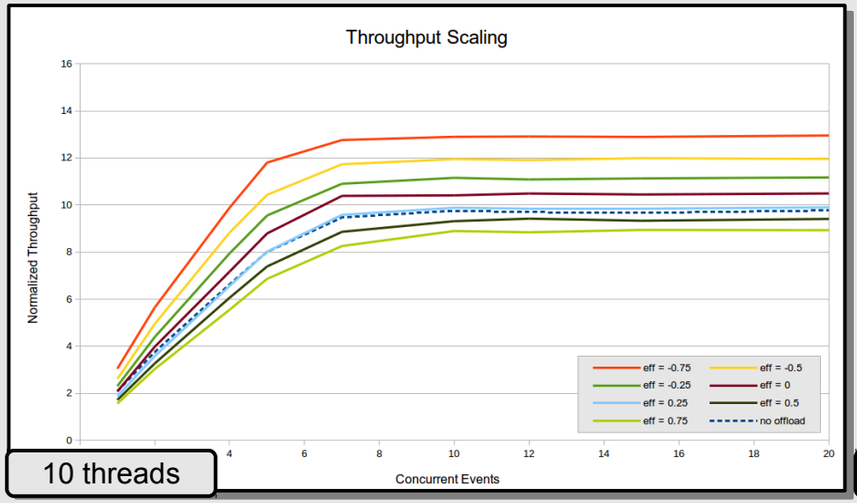
20 threads



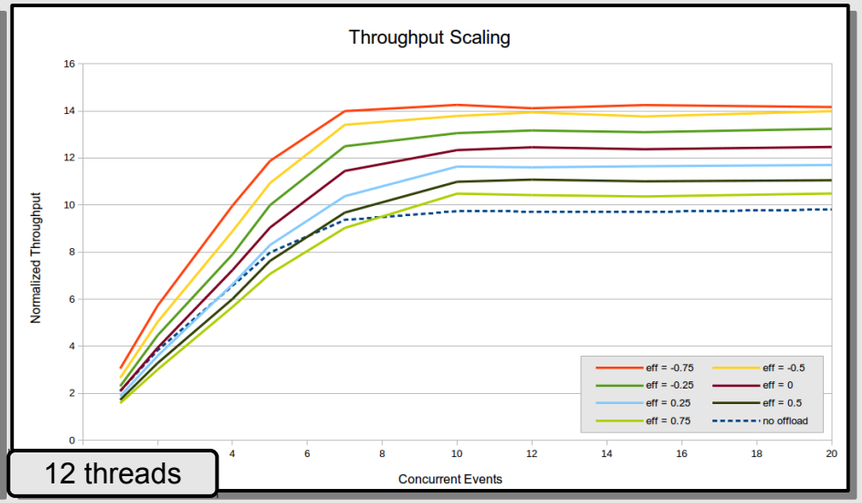
25 threads

Oversubscription : ATLAS

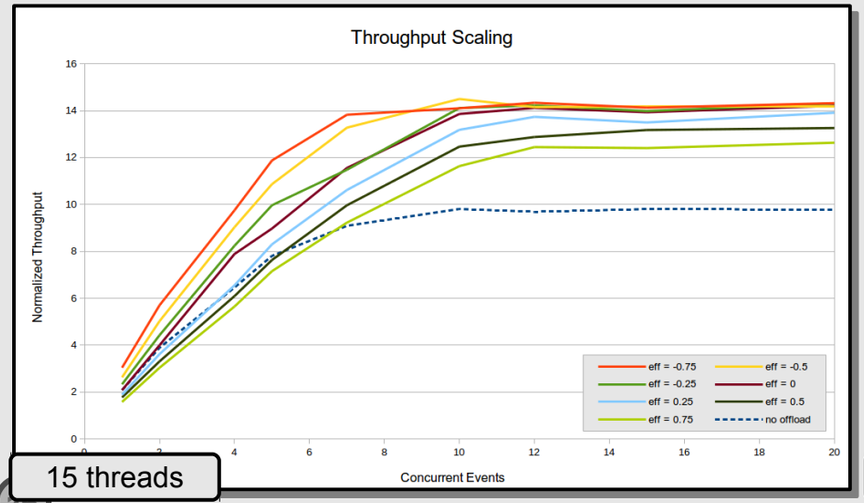
- Running with as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are idle
- We can **oversubscribe** the CPU with more threads to maximize throughput
- ATLAS: offload just 4 slowest Algorithms (3 on critical path)
  - 7% decrease in throughput



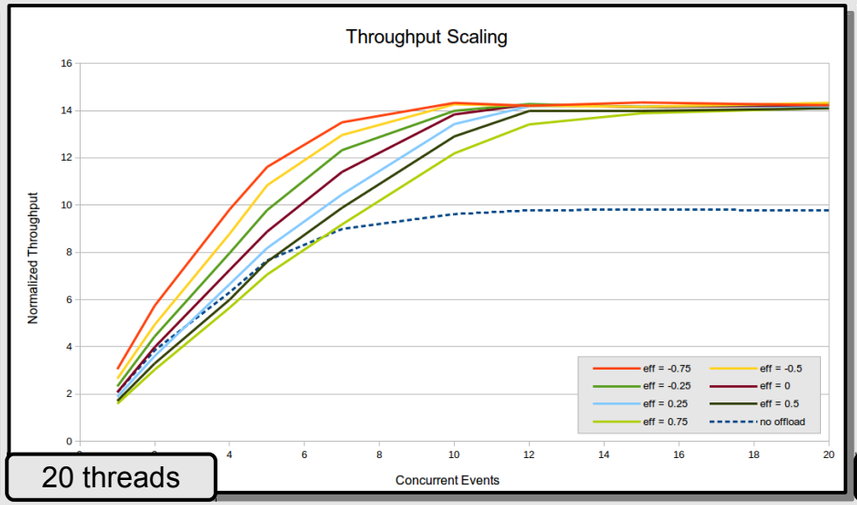
10 threads



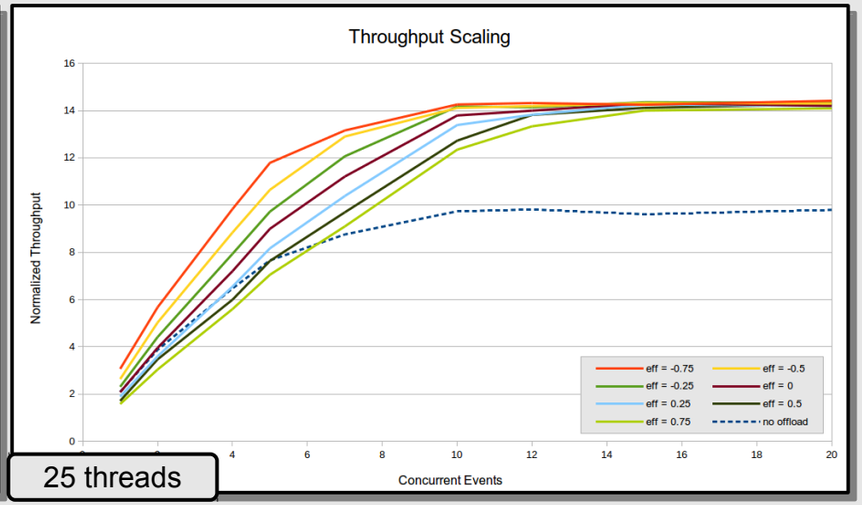
12 threads



15 threads



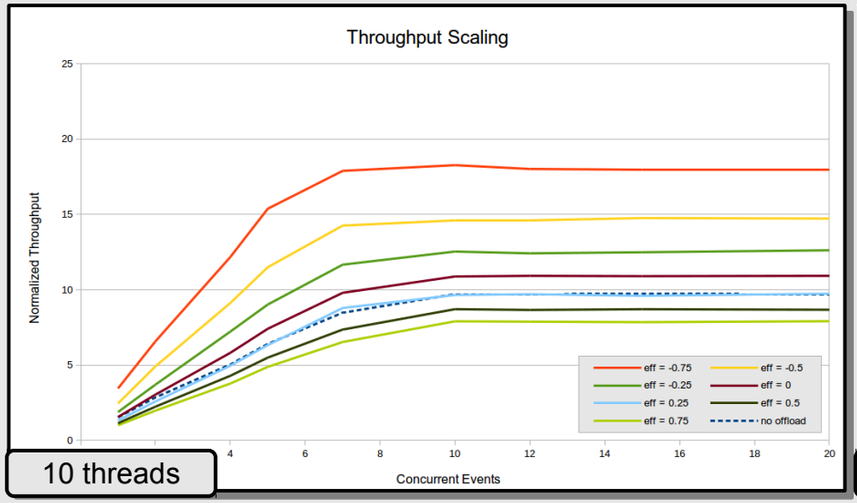
20 threads



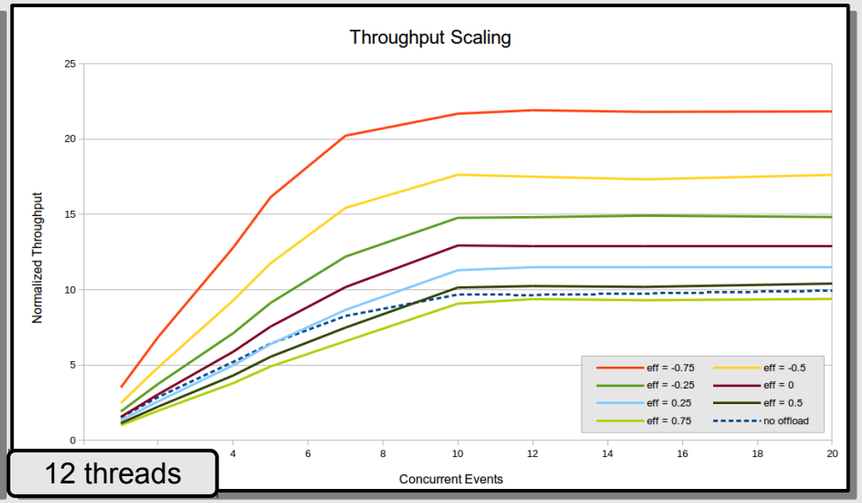
25 threads

Oversubscription : ATLAS High-μ

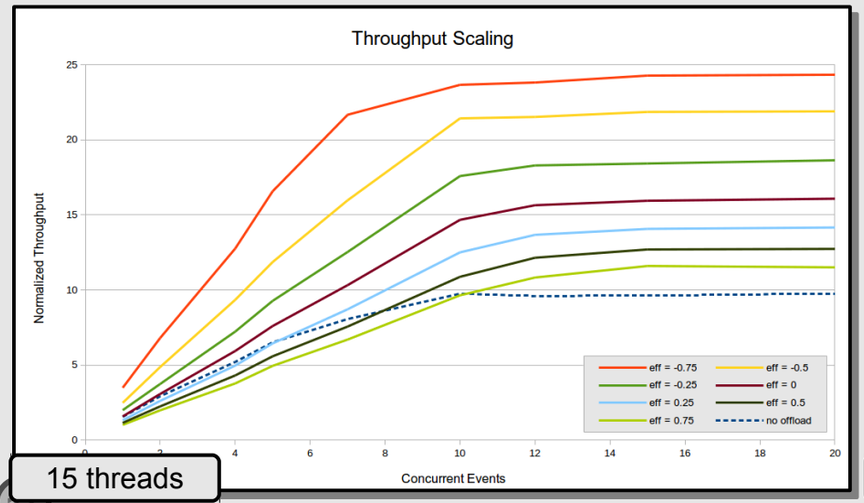
- Running with as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are idle
- We can **oversubscribe** the CPU with more threads to maximize throughput
- ATLAS high-μ input dataset



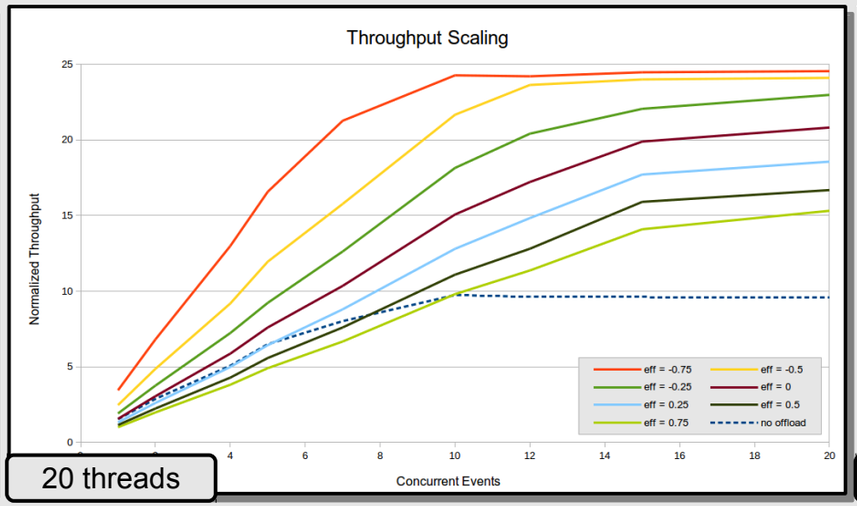
10 threads



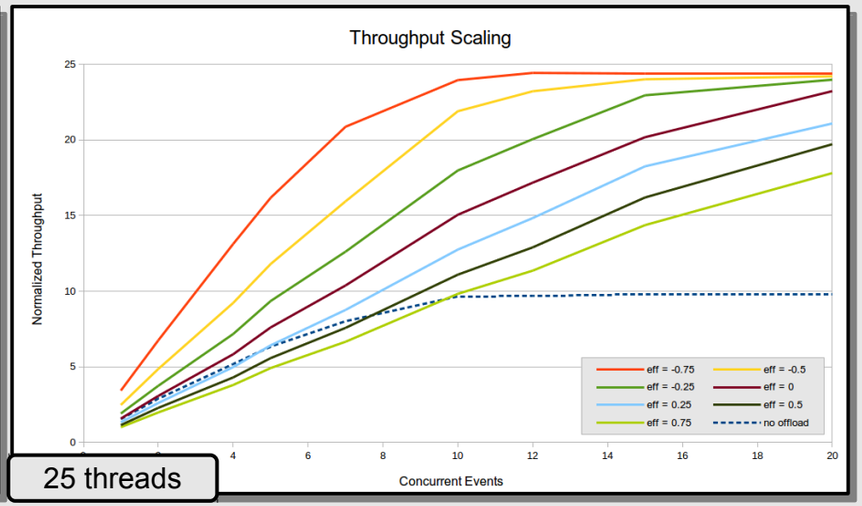
12 threads



15 threads



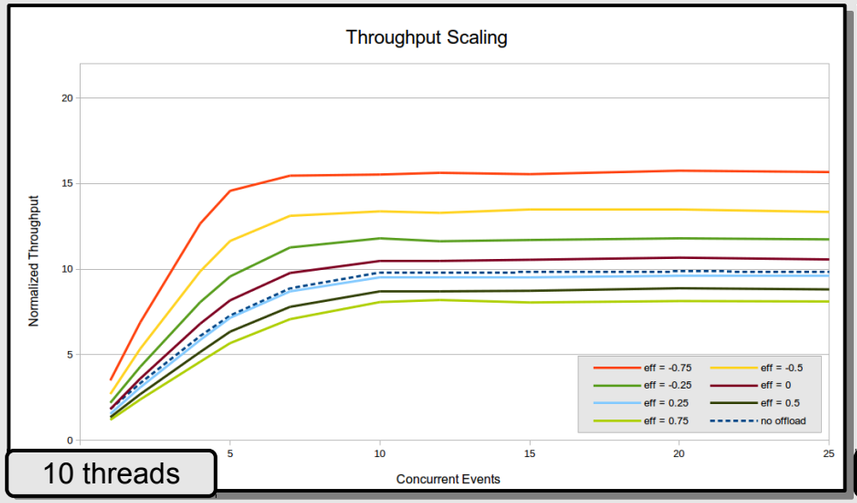
20 threads



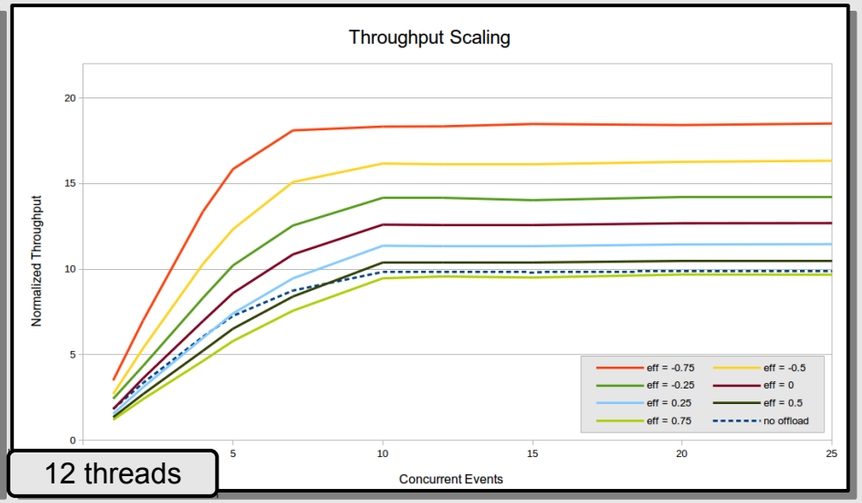
25 threads

Oversubscription : CMS

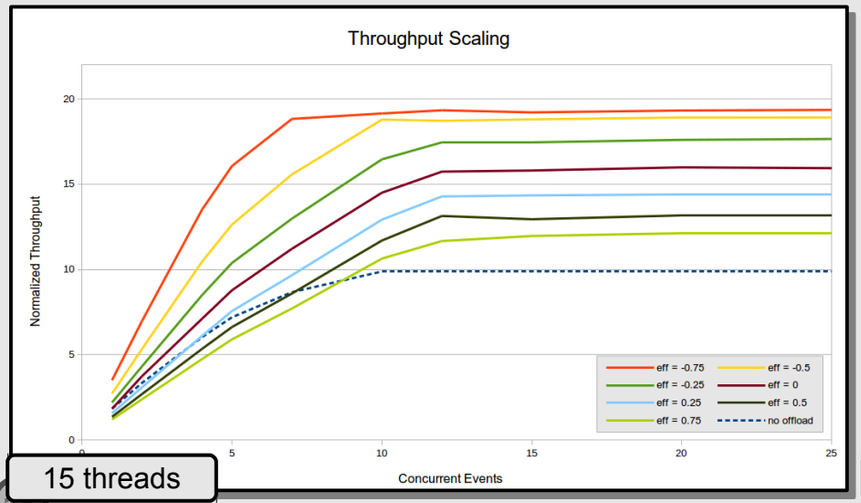
- Running with as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are idle
- We can **oversubscribe** the CPU with more threads to maximize throughput
- CMS scenario



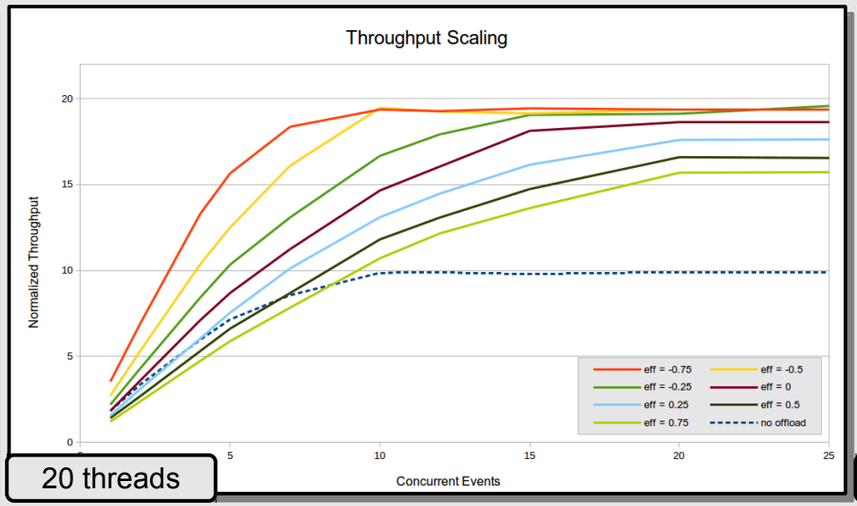
10 threads



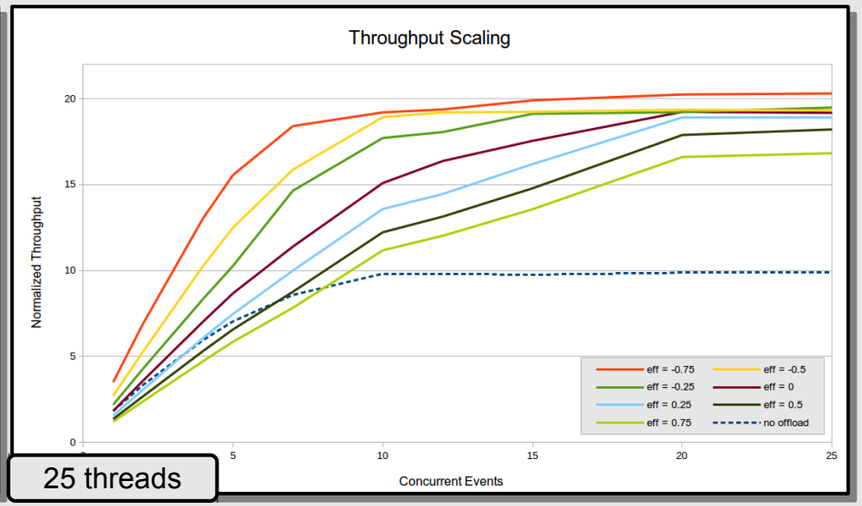
12 threads



15 threads



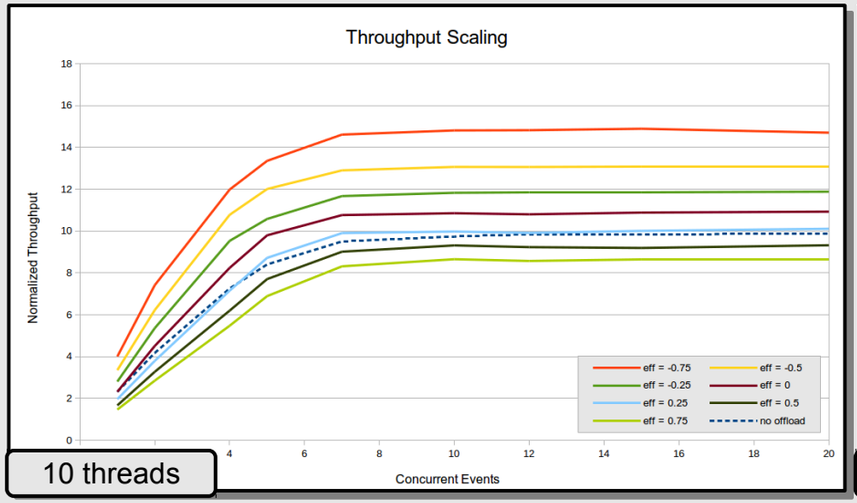
20 threads



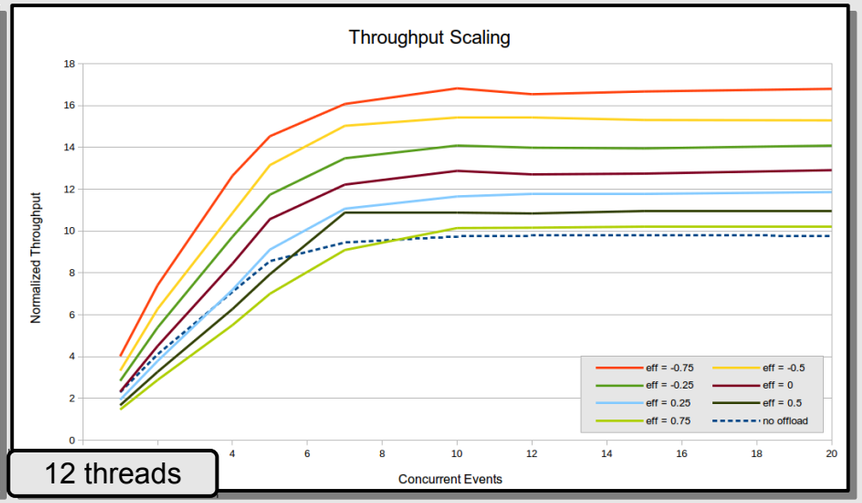
25 threads

Oversubscription : LHCb

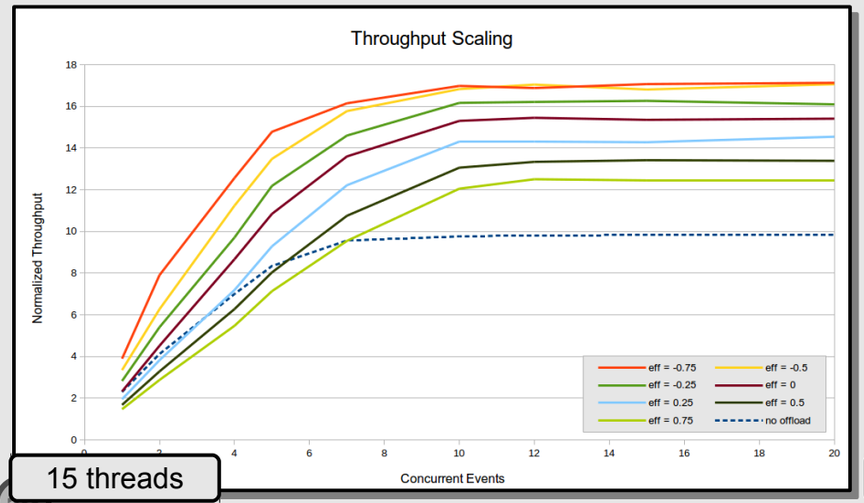
- Running with as many software threads as hardware threads results in less than full occupancy, as the offloaded Algorithms' hardware threads are idle
- We can **oversubscribe** the CPU with more threads to maximize throughput
- LHCb scenario



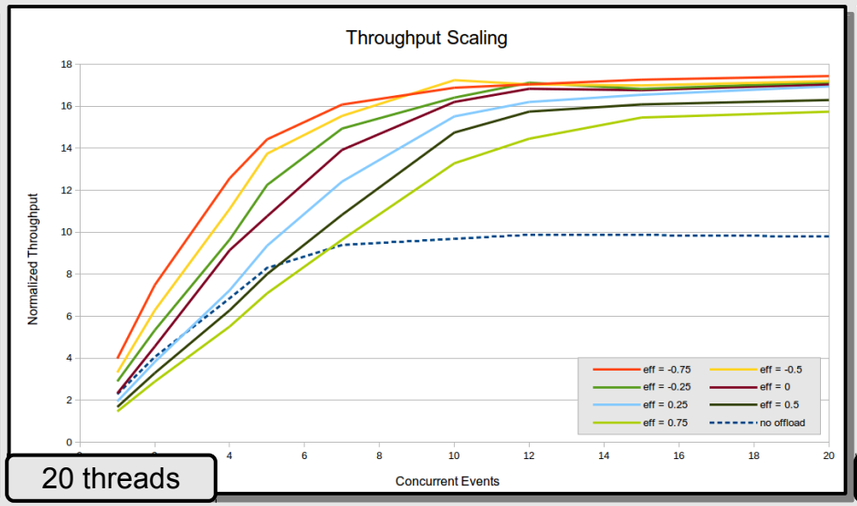
10 threads



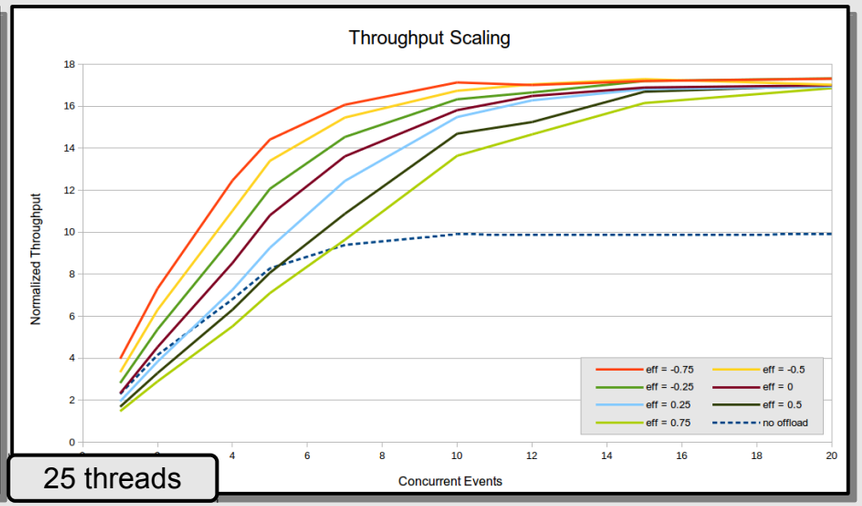
12 threads



15 threads



20 threads



25 threads

Observational Notes

- ATLAS reconstruction with current data has a much higher degree of inherit concurrency than CMS or LHCb.
  - Percentage of single event time spent on critical path (not counting I/O) is 40%
  - When a high-μ dataset is used, the critical path is lengthened to 68%. This has the effect of **decreasing** the available concurrency

Workflow	Total Algorithms	Critical Path Algorithms	Critical Path without I/O
ATLAS	309 : 14.6s	20 8.33s / 57.1%	17 5.78s / 39.6%
ATLAS high-μ	309 : 127.3s	32 95.7s / 75.2%	29 85.7s / 67.3%
CMS	707 : 13.4s	147 8.38s / 62.5%	145 7.32s / 54.6%
LHCb	282 : 491ms	13 259ms / 52.6%	11 234ms / 47.6%

- CMS has the most sequential reconstruction sequence, and the largest number of Algorithms: 55% of the time is on the critical path
- LHCb falls between the two, with 48% of time spent on the critical path
- Longer critical paths lead to less inherent concurrency, requiring more oversubscription and concurrent events in order to maximize throughput

Accelerator Data Transfer Latencies

- It takes time to marshal data, and send it to (and get it back) from an accelerator
- Depending on the Algorithm, this might be significant
- Does this added latency matter?

- Has a similar effect on throughput as decreasing the efficiency of the offloaded Algorithm
  - at some point, it begins to matter
  - effect is very dependent on the runtime of the Algorithm on the accelerator, and the amount of data transmitted

- The effect (less than optimal CPU occupancy) can be managed by increasing the number of concurrent events
  - some downsides due to increased memory usage

- In general, as long as the CPU is not spending time converting/transmitting data (*ie,* data is already in a form that the accelerator can easily use), this is not likely to be a problem

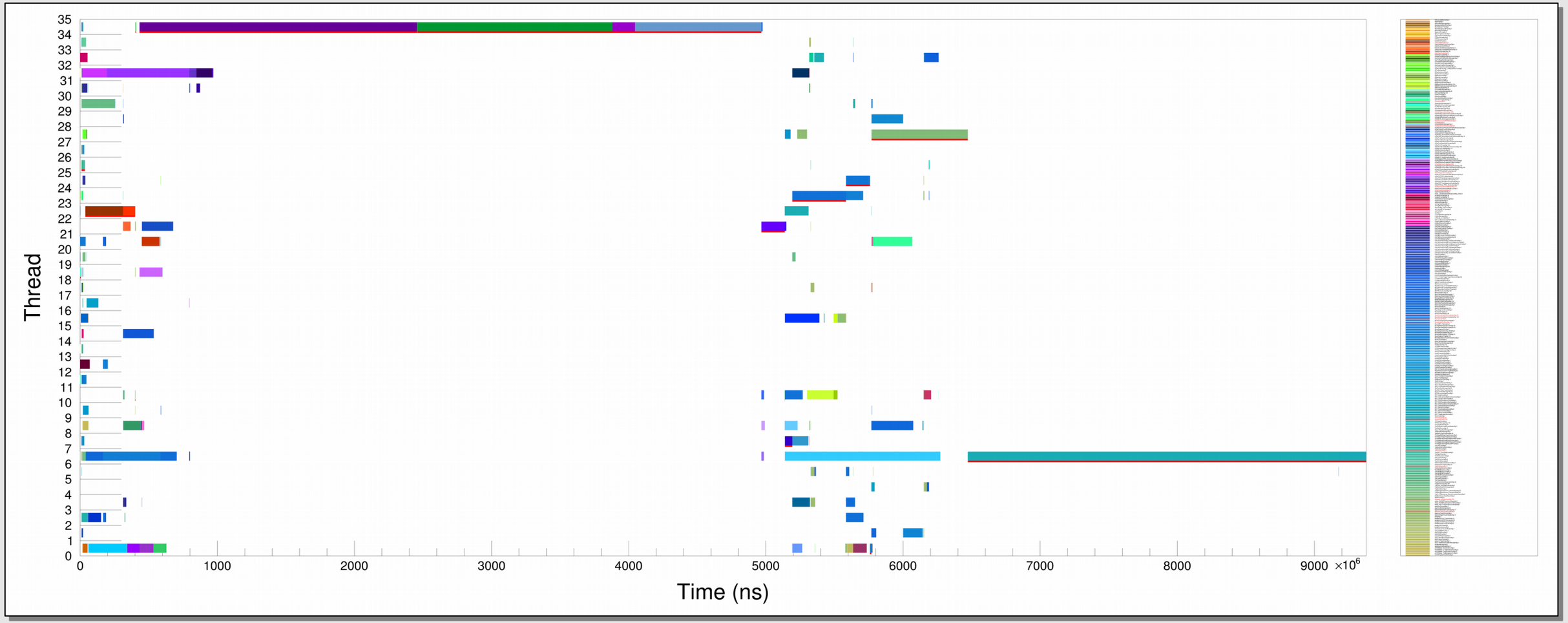
Conclusions

- Understanding the critical path of Algorithm execution is very important when determining where to concentrate efforts in offloading Algorithms
- While simulated throughput studies show that offloading Algorithms on the critical path can be much more advantageous than others
  - latency hiding means that anything you offload is essentially a win, whether or not it's on the critical path
  - offloading Algorithms not on the critical path require increasing the number of concurrent events to maximize throughput
  - rewriting these Algorithms for the accelerator is an exercise left for the implementer....
- Algorithms don't need to run exceptionally efficiently on the accelerator (*ie.* faster than on the CPU)
  - inefficient accelerator usage can be offset by increasing number of concurrent events
- Oversubscription of CPU hardware threads is essential to maximizing overall throughput
  - threads that offload Algorithms are basically sleeping until the accelerator returns
  - in our scenarios the cost of context switching in negligible enough to not affect performance
  - blocking nature of linux kernel for I/O enables similar performance benefits for I/O Algorithms

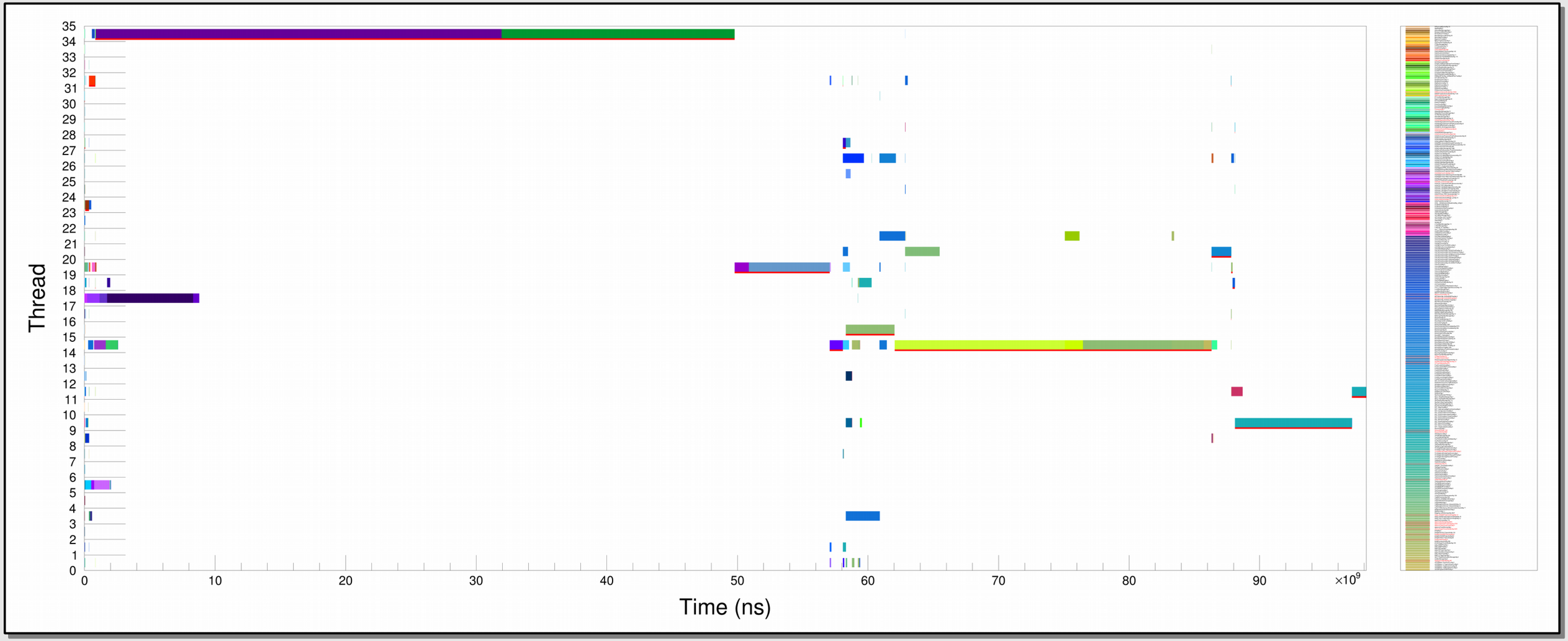
fin

ATLAS Timeline

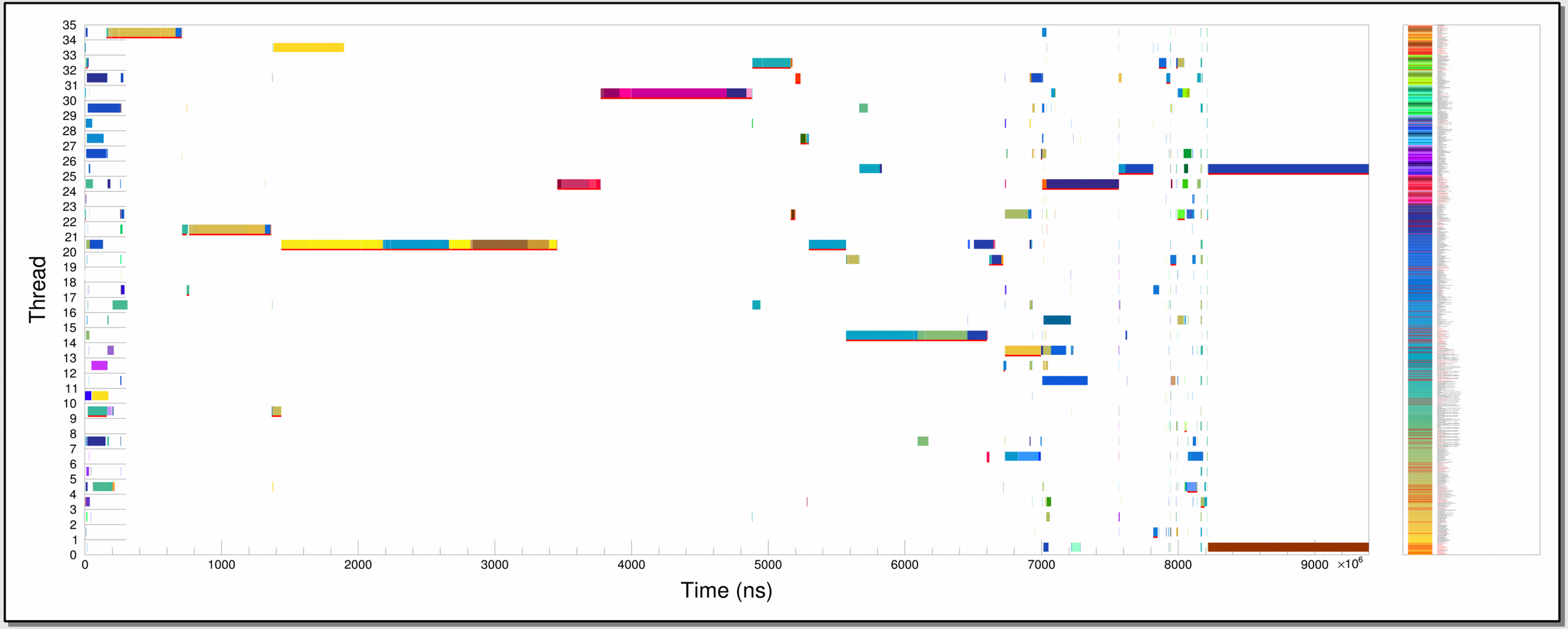
- ATLAS timeline chart for 1 event w/ 35 threads, no offloading
  - critical path Algorithms in red



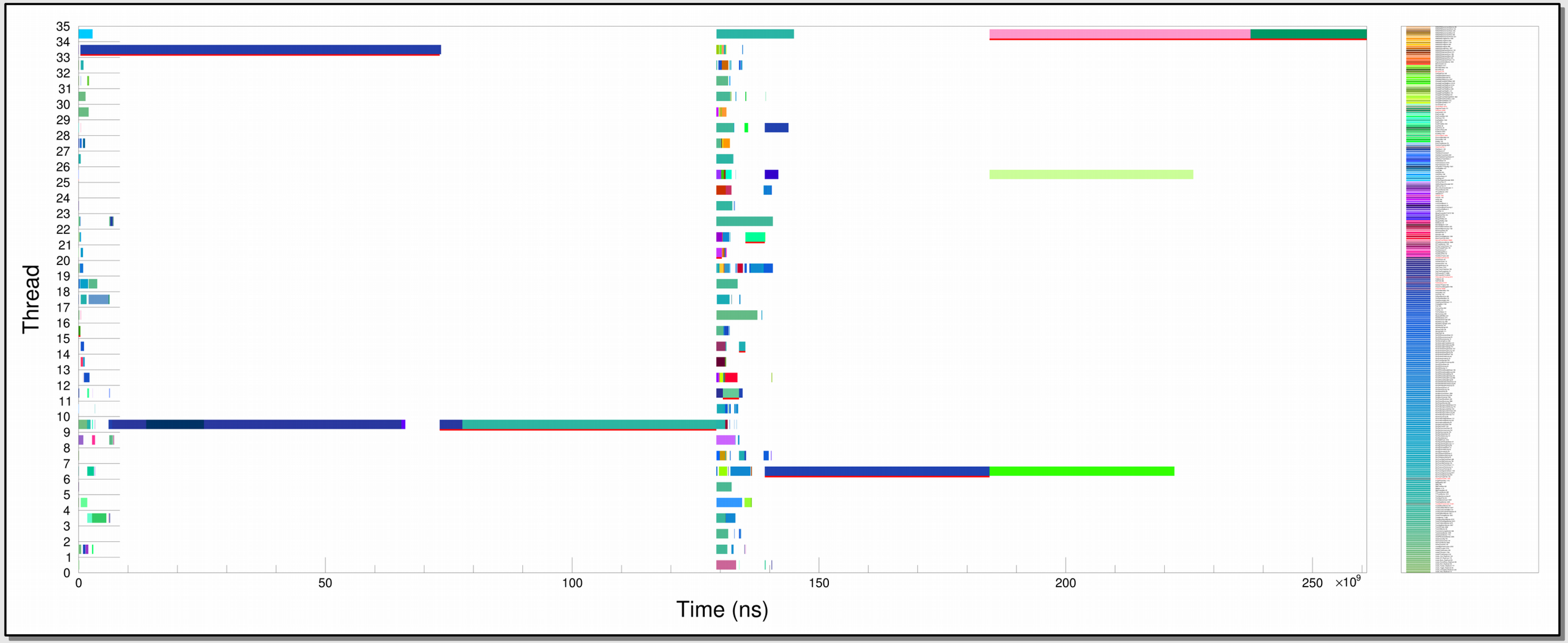
ATLAS High-μ Timeline



CMS Timeline

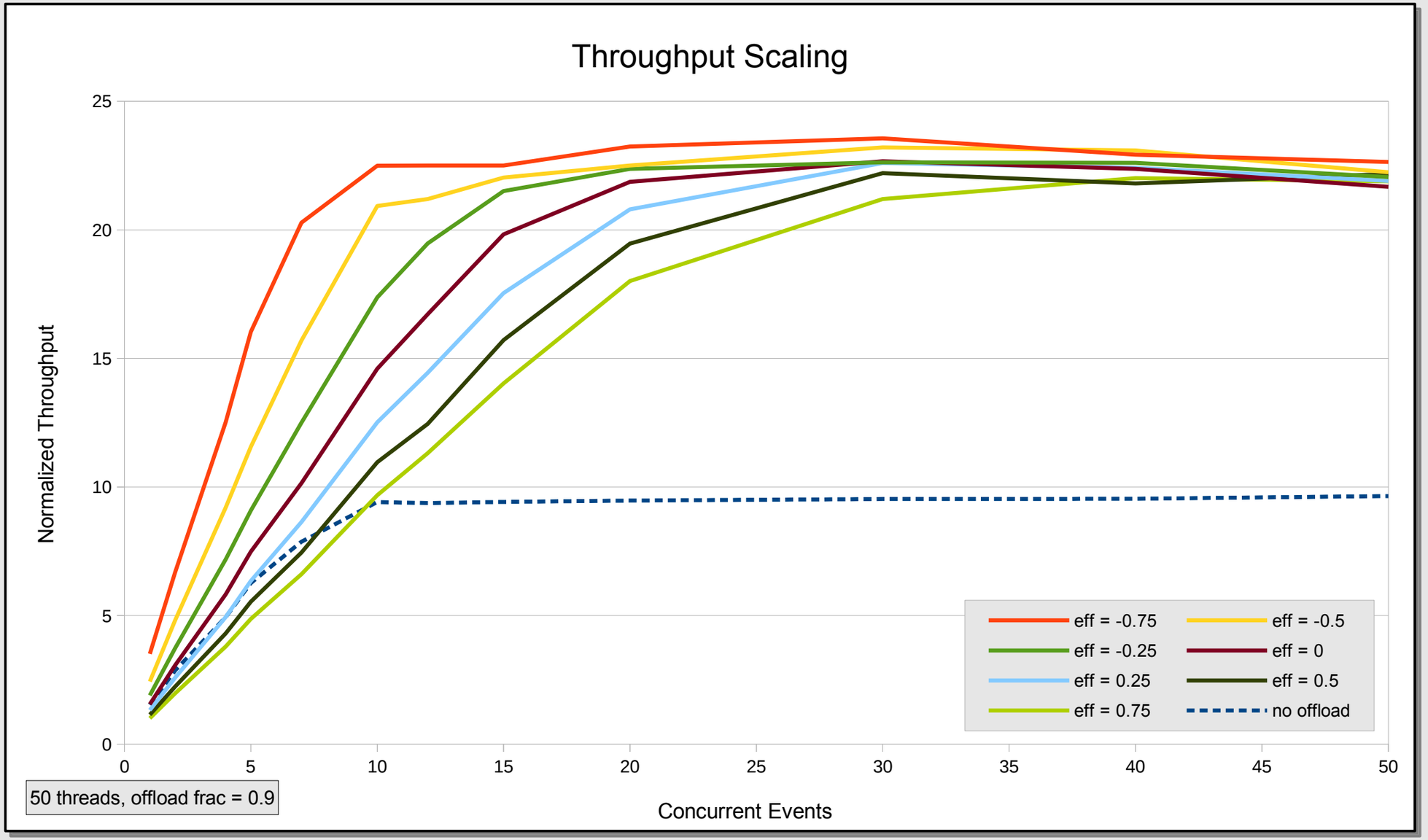


LHCb Timeline



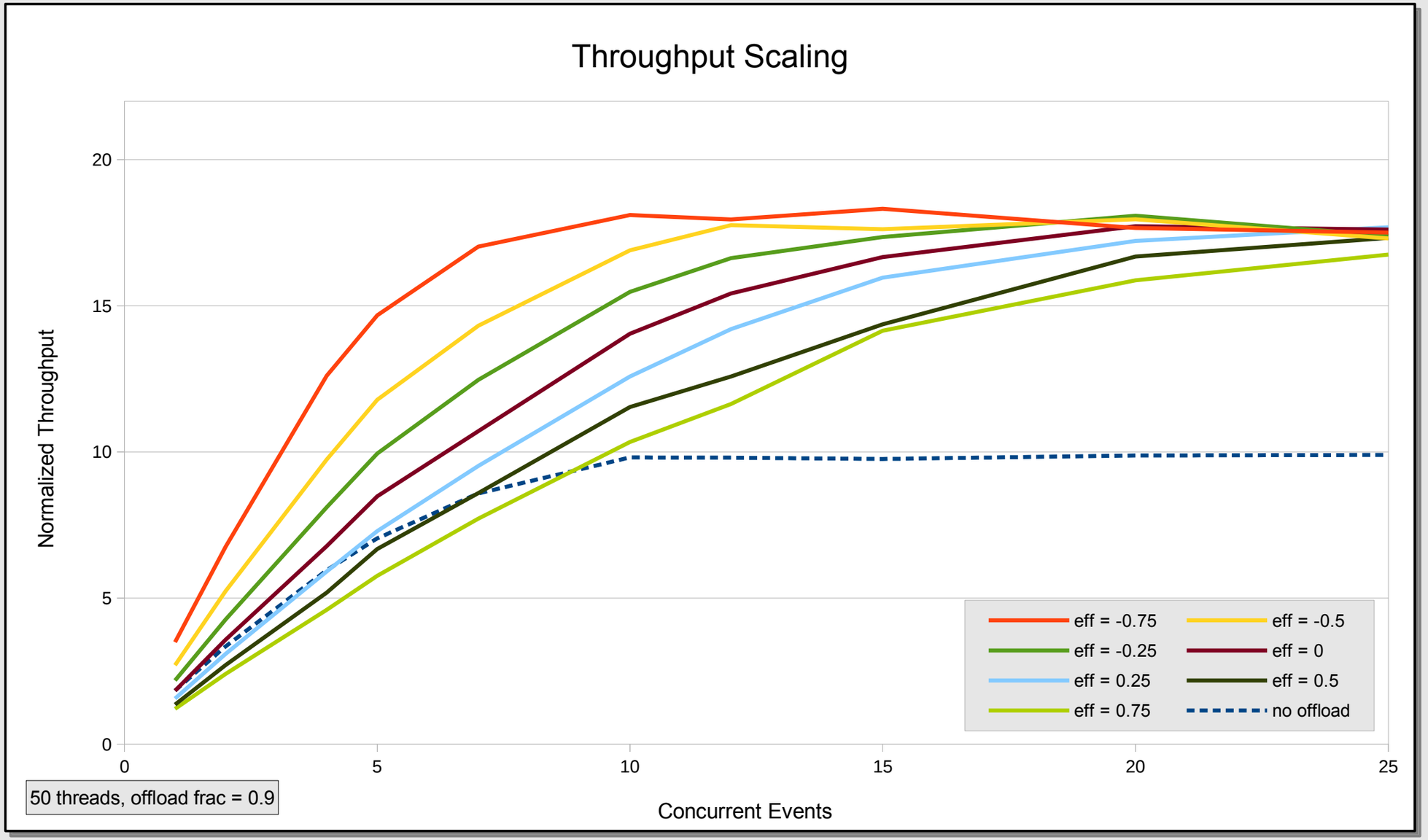
ATLAS High-μ : Massive Oversubscription

- ATLAS high-μ scenario with 50 threads



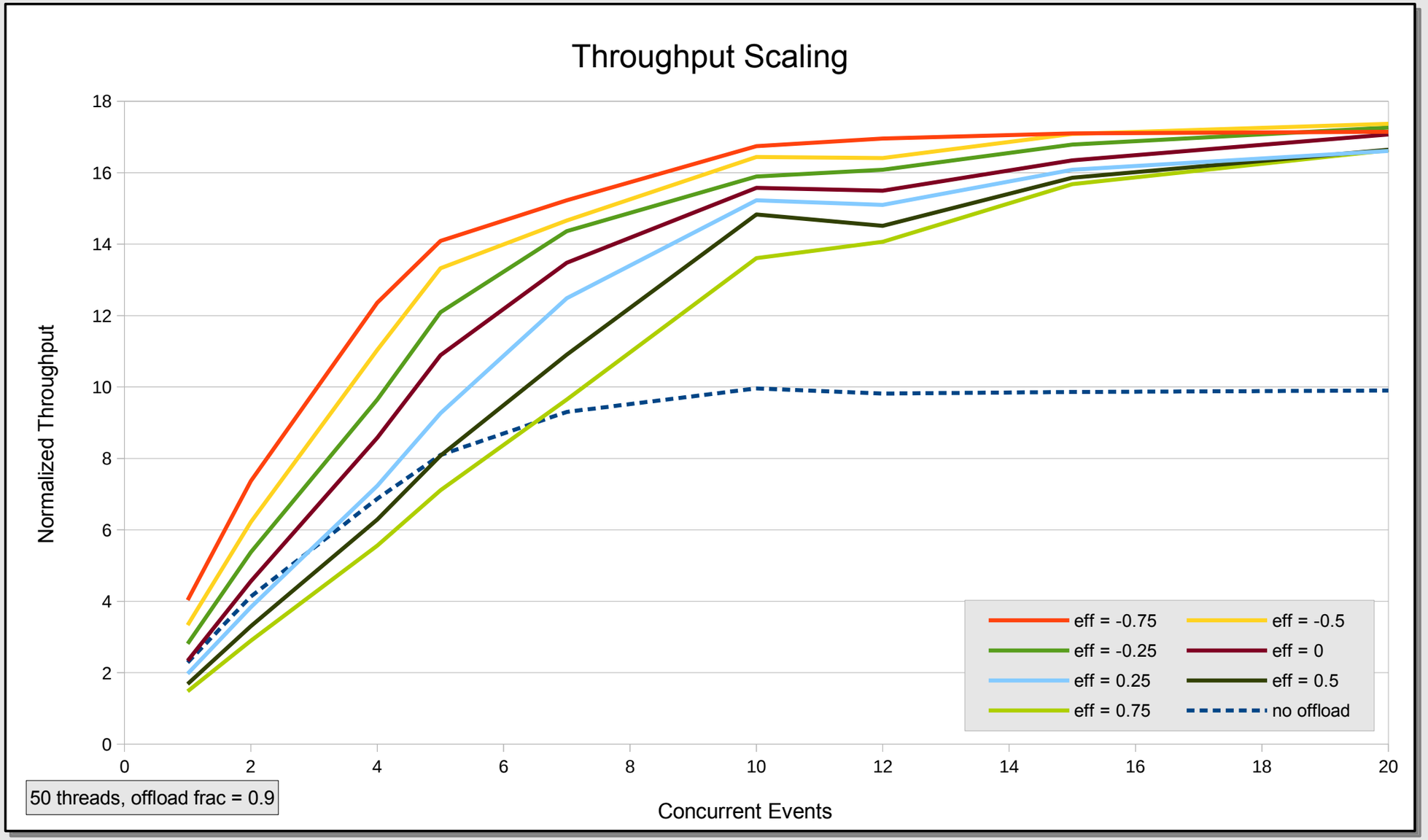
CMS : Massive Oversubscription

- CMS scenario with 50 threads



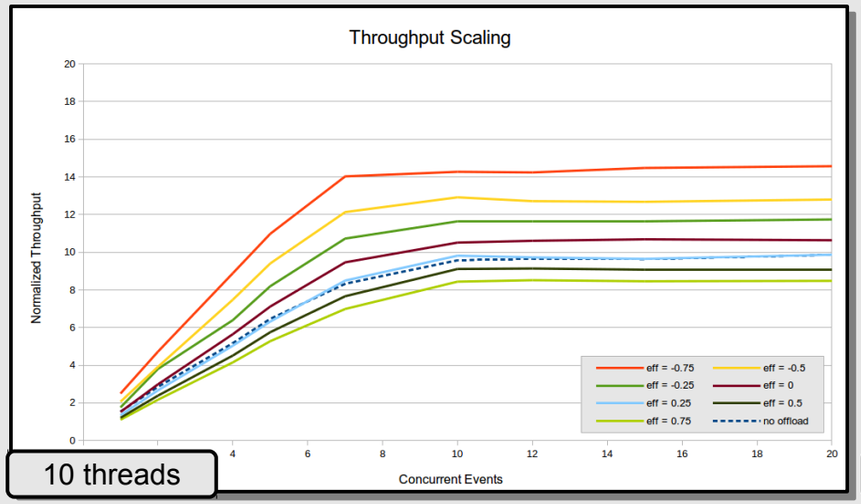
LHCb : Massive Oversubscription

- LHCb Scenario with 50 threads

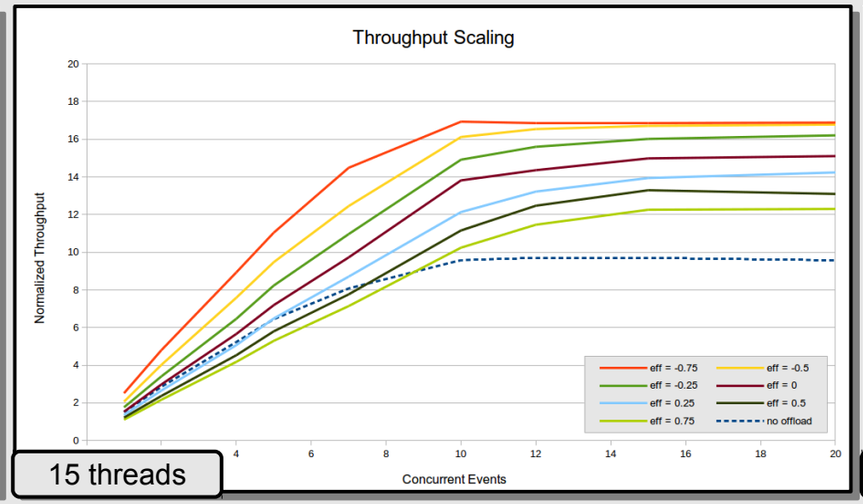


Oversubscription : ATLAS High-μ, 3 Slowest Algs

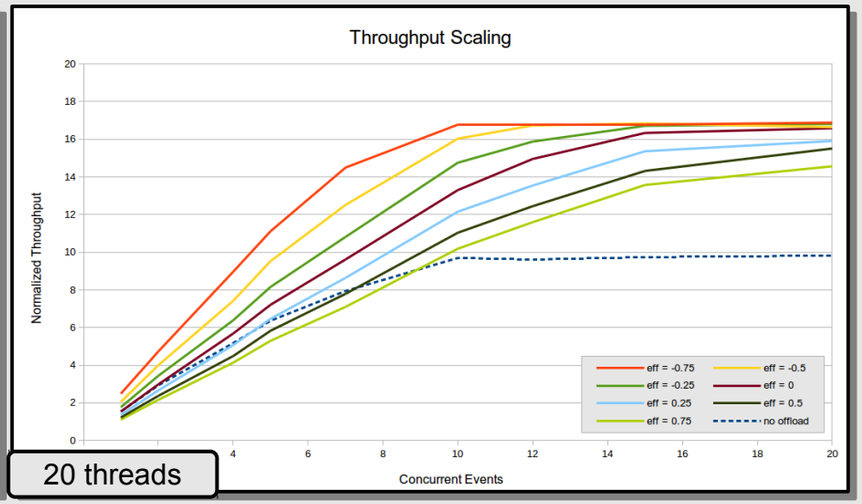
- ATLAS high-μ input dataset, 3 slowest Algorithms (all on critical path)



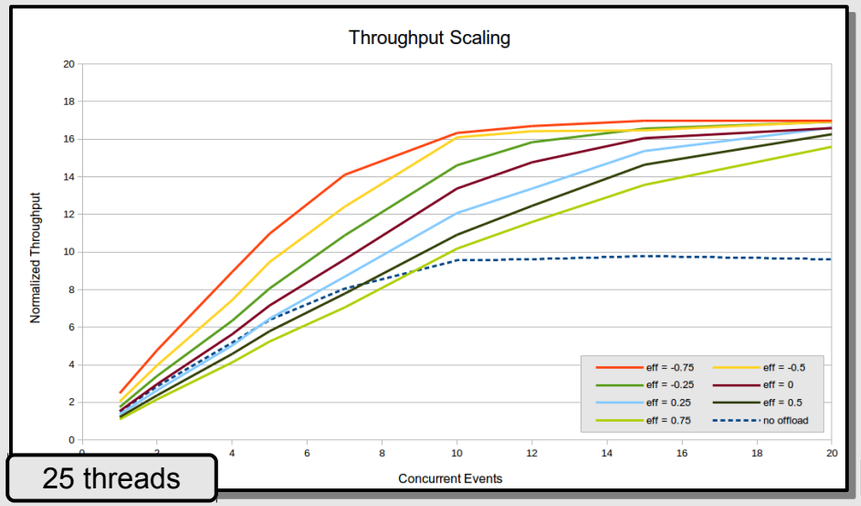
10 threads



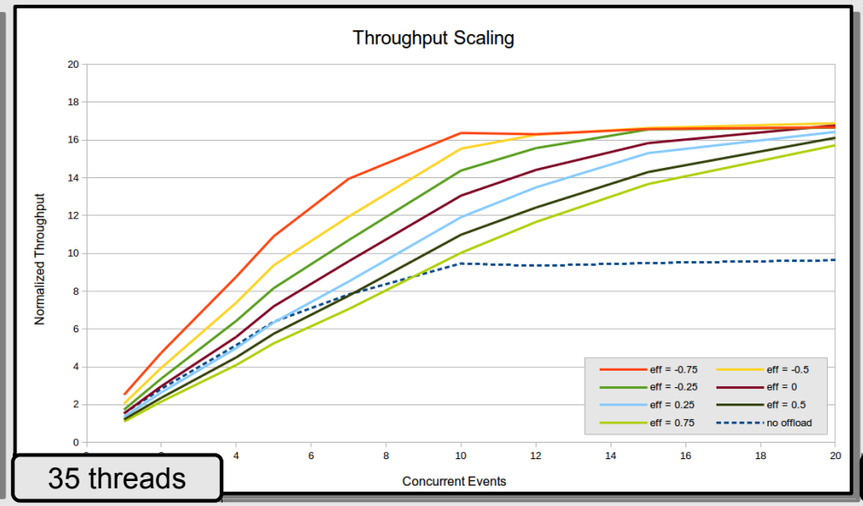
15 threads



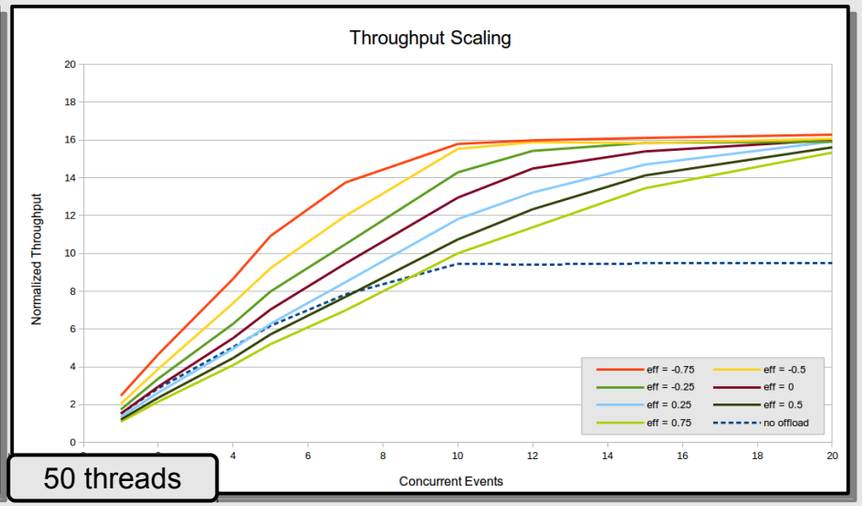
20 threads



25 threads



35 threads

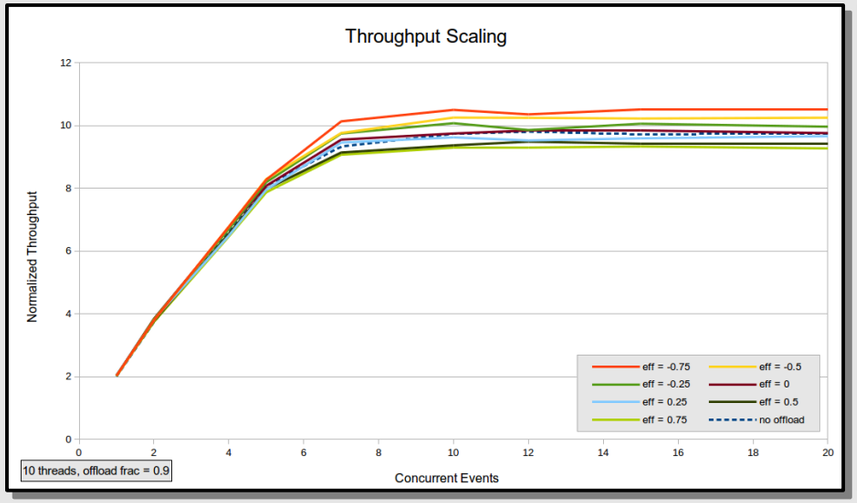


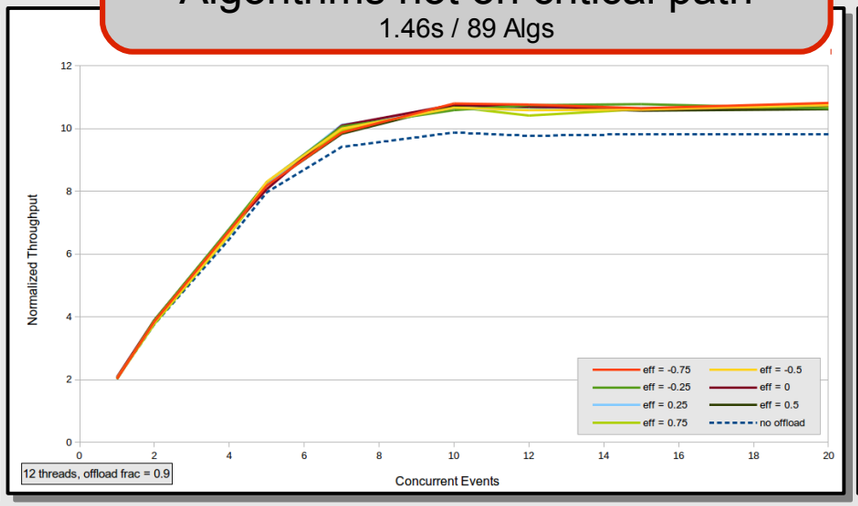
50 threads

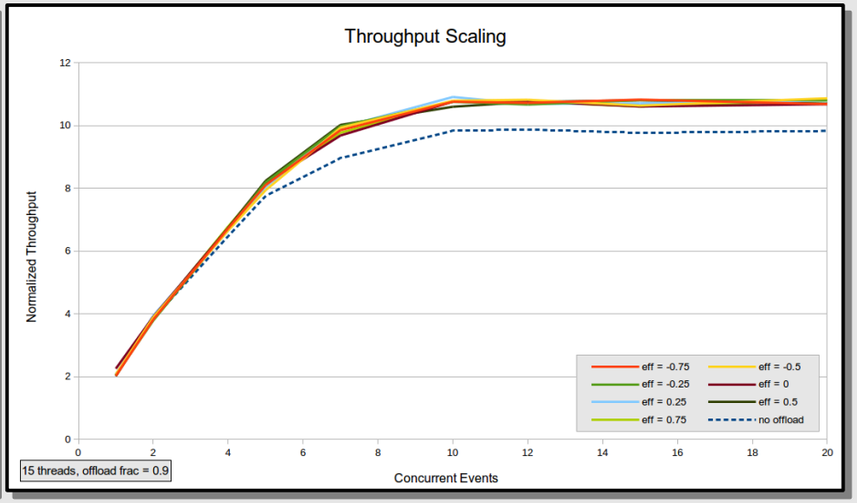
ATLAS: Offload 10% Of Event Processing Time

- ATLAS: randomly select Algorithms on/off critical path, such that total time is equal to ~10% of one event's

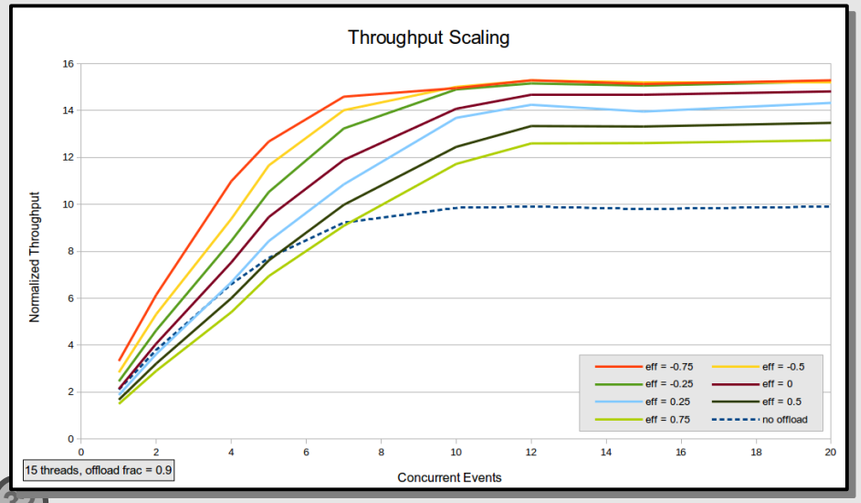
Algorithms not on critical path
1.46s / 89 Algs

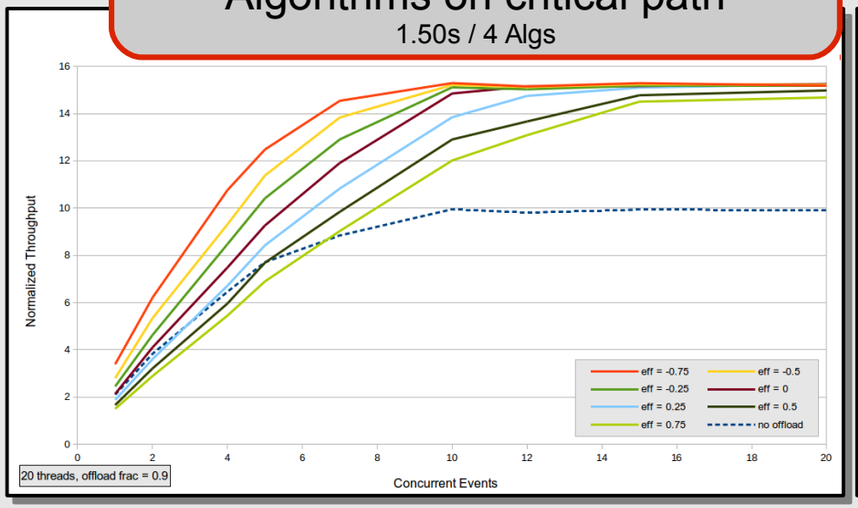


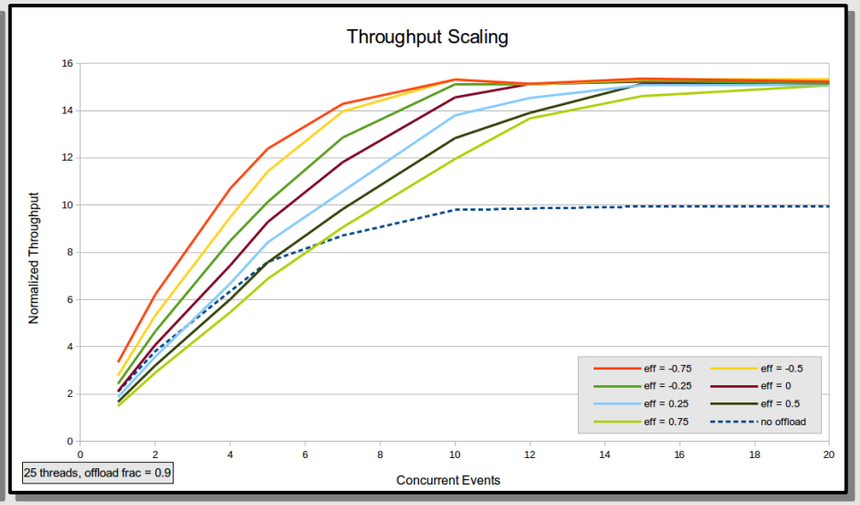




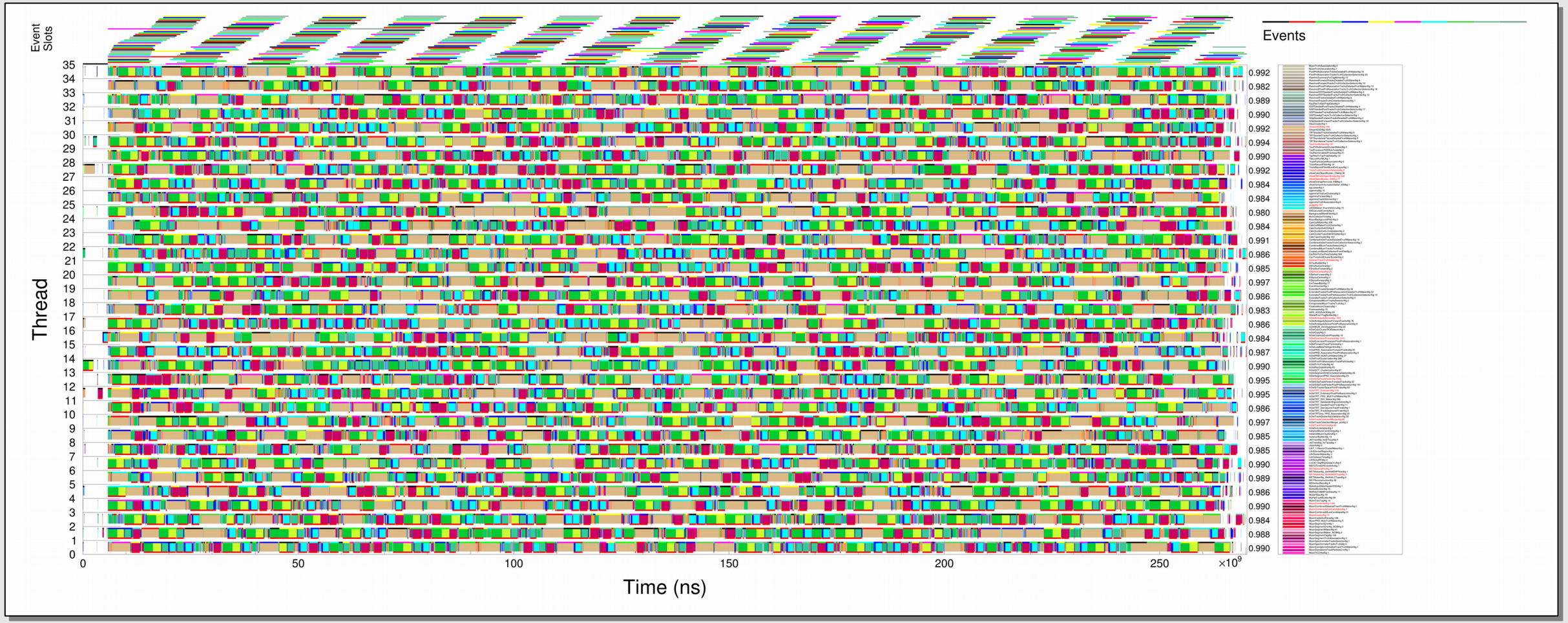
Algorithms on critical path
1.50s / 4 Algs



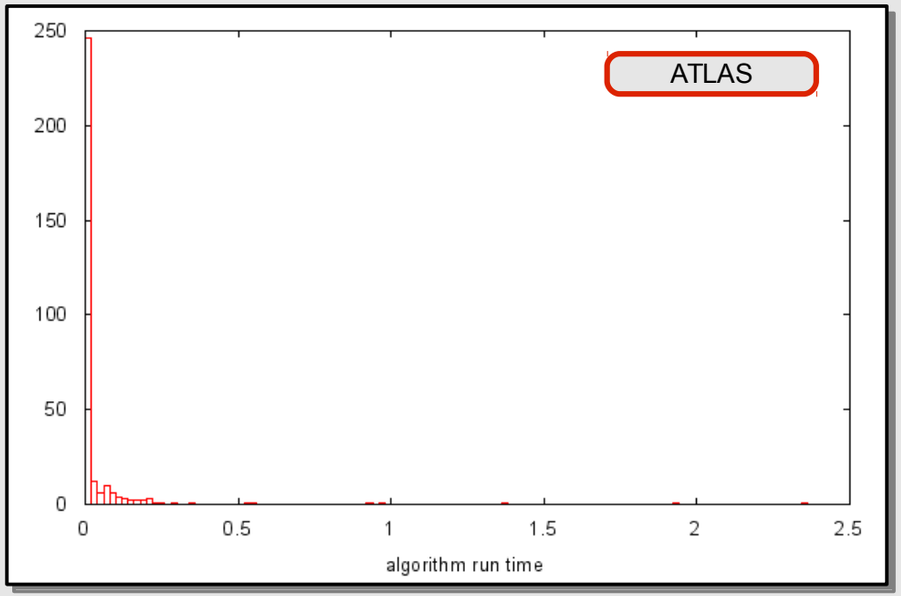


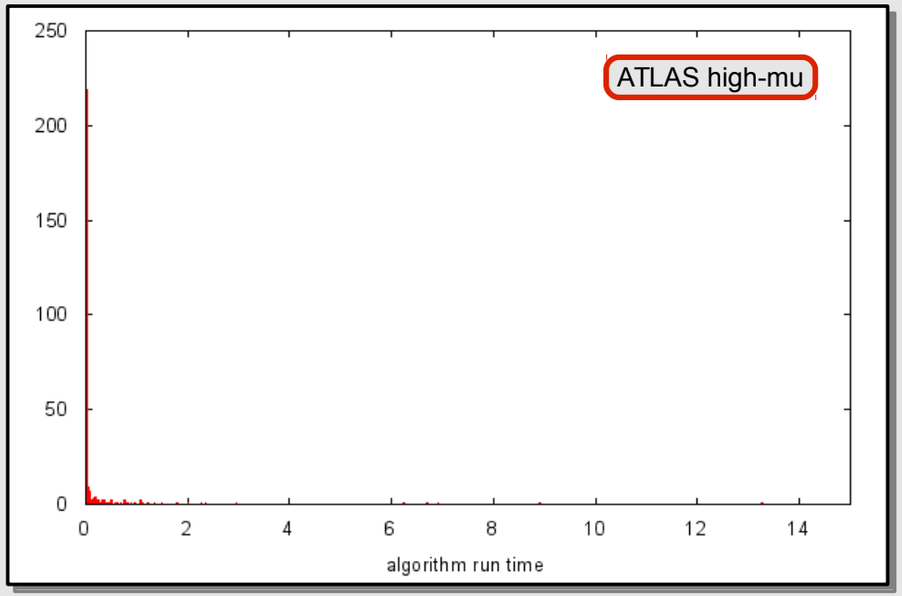


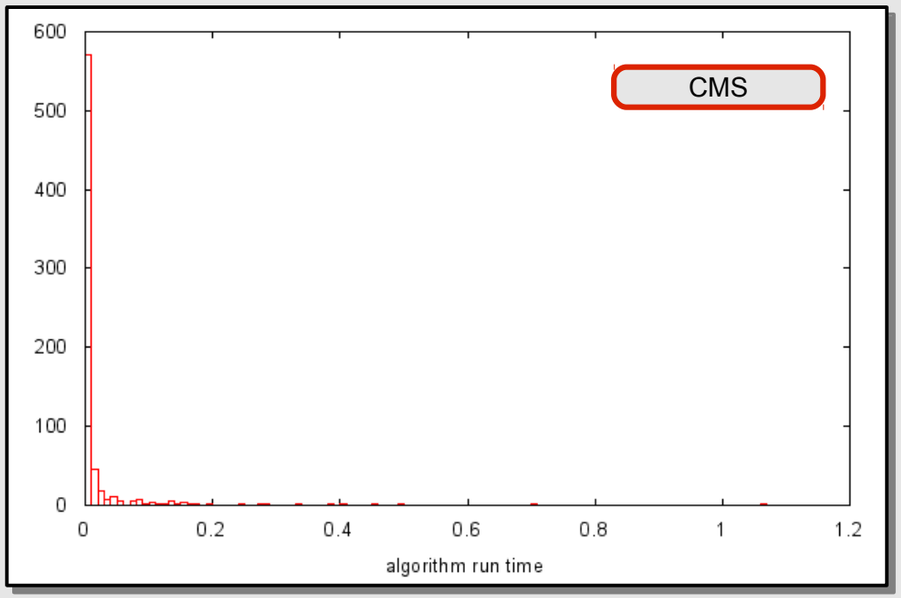
- ATLAS timeline chart for 35 concurrent evts w/ 35 threads, no offloading, 500 events, showing occupancy of each thread

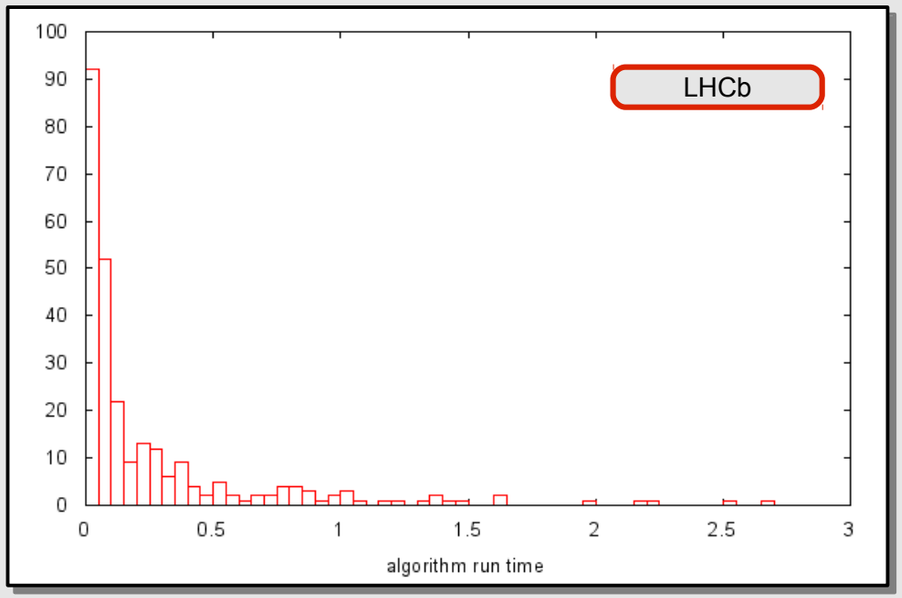


Time Distribution of Algorithms









ATLAS: Algorithms on the Critical Path

Algorithm	time (s)
InDetSiSpTrackFinderAlg	1.920
InDetAmbiguitySolverAlg	1.365
InDetExtensionProcessorAlg	0.924
btagging_antikt4emtopoAlg	0.542
MuonCombinedInDetCandidateAlg	0.291
CaloCellMakerAlg	0.217
InDetTRT_ExtensionAlg	0.163
InDetTrackCollectionMergerAlg	0.129
MuonInsideOutRecoAlg	0.112
CaloTopoClusterAlg	0.067
InDetTrackParticlesAlg	0.026
TileRChMakerAlg	0.010
MuonCreatorAlg	0.006
TileDQstatusAlg	0.005
InDetCaloClusterROISelectorAlg	0.002
egammaTopoClusterCopierAlg	0.001
InDetCopyAlg	0.001

high μ

Algorithm	time (s)
InDetSiSpTrackFinderAlg	29.416
InDetAmbiguitySolverAlg	18.112
EMBremCollectionBuilderAlg	13.262
egammaRecBuilderAlg	6.917
InDetExtensionProcessorAlg	6.233
egammaSelectedTrackCopyAlg	2.945
electronSuperClusterBuilderAlg	2.274
EMVertexBuilderAlg	1.355
InDetTRT_ExtensionAlg	1.079
PFAlgorithmAlg	0.945
InDetTrackCollectionMergerAlg	0.765
topoEgammaBuilderAlg	0.683
PFTrackSelectorAlg	0.612
CaloTopoClusterAlg	0.444
CaloCellMakerAlg	0.225
InDetTrackParticlesAlg	0.175
METAssociationAlg	0.167
jetalgAntiKt4EMPFlowJetsAlg	0.051
InDetCaloClusterROISelectorAlg	0.038
PFONeutralCreatorAlgorithmAlg	0.012
TileRChMakerAlg	0.012
edalg_Kt4EMPFlowEventShapeAlg	0.006
TileDQstatusAlg	0.005
egammaTopoClusterCopierAlg	0.003
jetalgCHSPFlowAlg	0.003
METMakerAlg_AntiKt4EMTopoAlg	0.002
ThinNegativeEnergyCHSNeutralPFOsAlg	0.001
PFLeptonSelectorAlg	0.001
InDetCopyAlg	0.001