Simulating HEP Workflows on Heterogeneous Architectures

Charles Leggett, Illya Shapoval

on behalf of the ATLAS collaboration

iEEE eScience Amsterdam
Oct 31 2018

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 (2021?) → A21
    - ??? - was supposed to be successor to KNL
    - "novel architecture" -> maybe CSA?
  - LLNL: Sierra (2018)
    - 4320 IBM AC922 nodes *w/* 2x Power9 CPU
    - 2x NVIDIA Volta V100 + NVLink / CPU
  - LBL: NERSC-9 (2020)
    - was supposed to be successor to KNL
    - AMD x86 + GPU

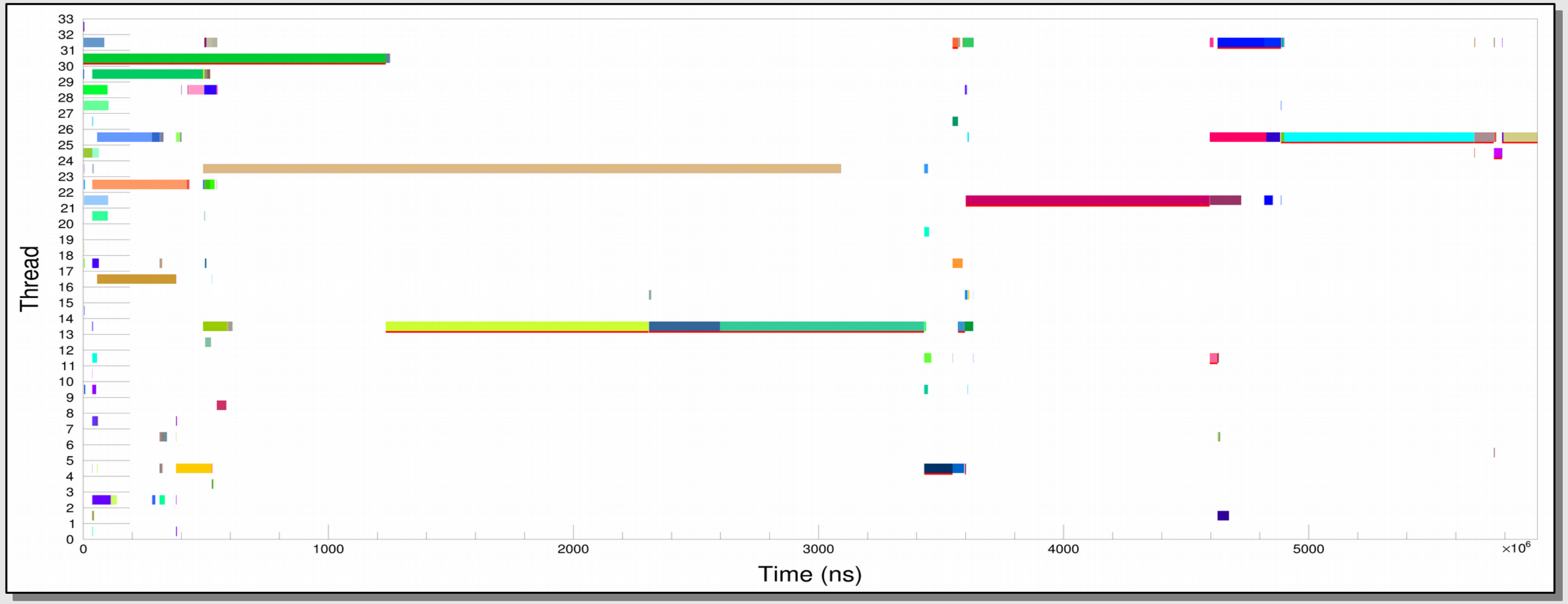
- In order to meet the HL-LHC computing requirements, we need to use all available computing resources, or cut back physics projections
- US funding agencies have indicated that we will not be able to get allocations if our code does not make use of accelerator hardware

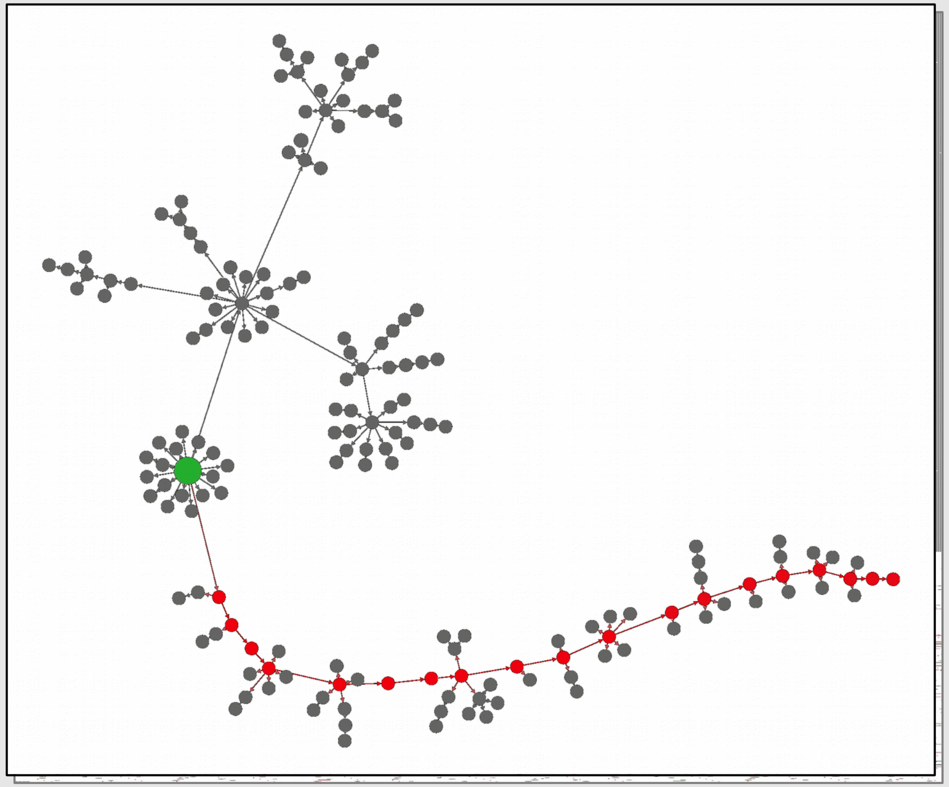
Offloading HEP Software

- 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

- As a test case, we have selected a standard ATLAS reconstruction workflow that comprises 197 Algorithms
  - Algorithm data interdependencies and timings have been extracted from actual data
  - Run using Gaudi Avalanche task scheduler, with artificial CPU Crunchers instead of real algorithms, allowing cloning of all Algorithms





- Analyze 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
  - 19 Algorithms, 5.6s out of 10.2s total event processing time

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

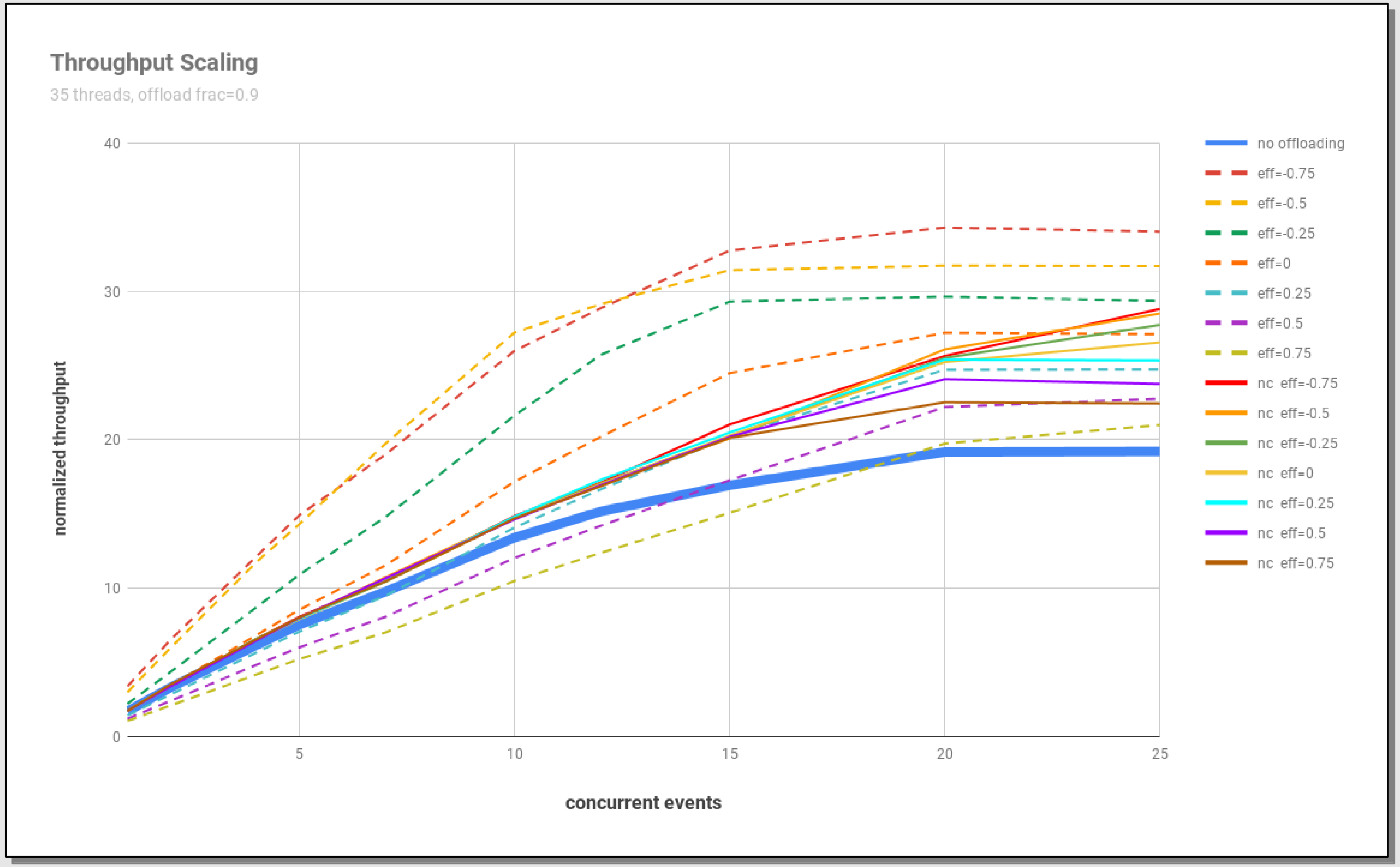
- Model offloading by modifying runtime t_{orig} of the Algorithm with 3 parameters:
  - fraction (*frac*) of Algorithm runtime that can be offloaded
  - efficiency (*eff*) of running offloaded part on accelerator (does it run faster or slower?)
  - extra time (t_{extra}) to transfer data to/from accelerator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

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

- 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

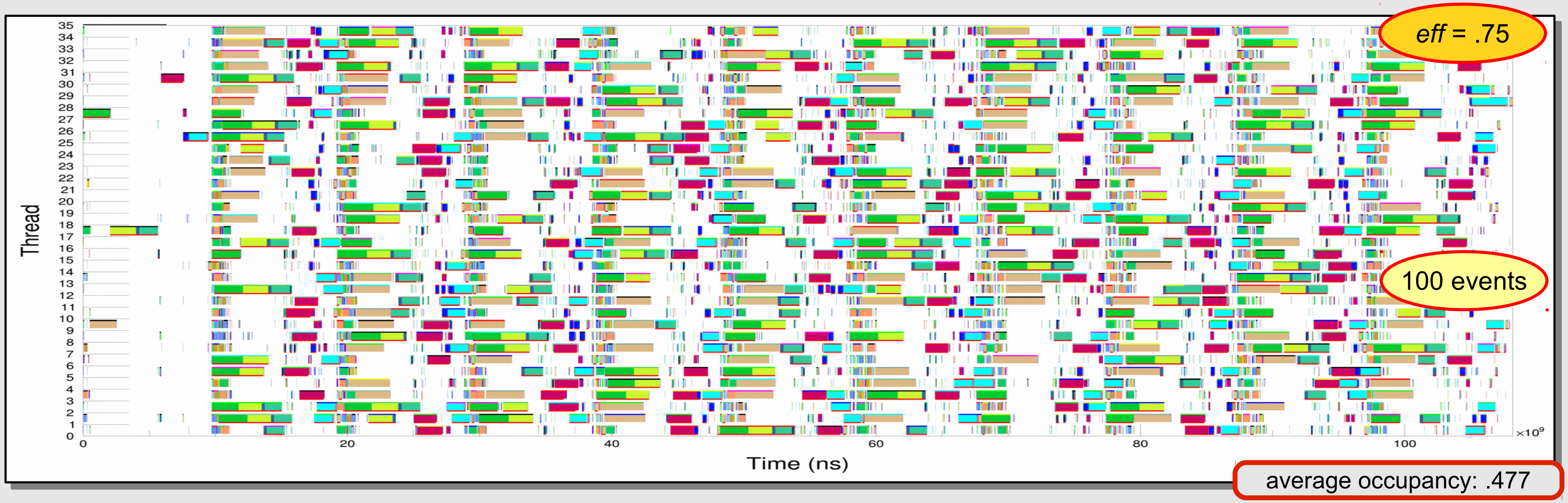
Comparison of Changing Accelerator Efficiency

- 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

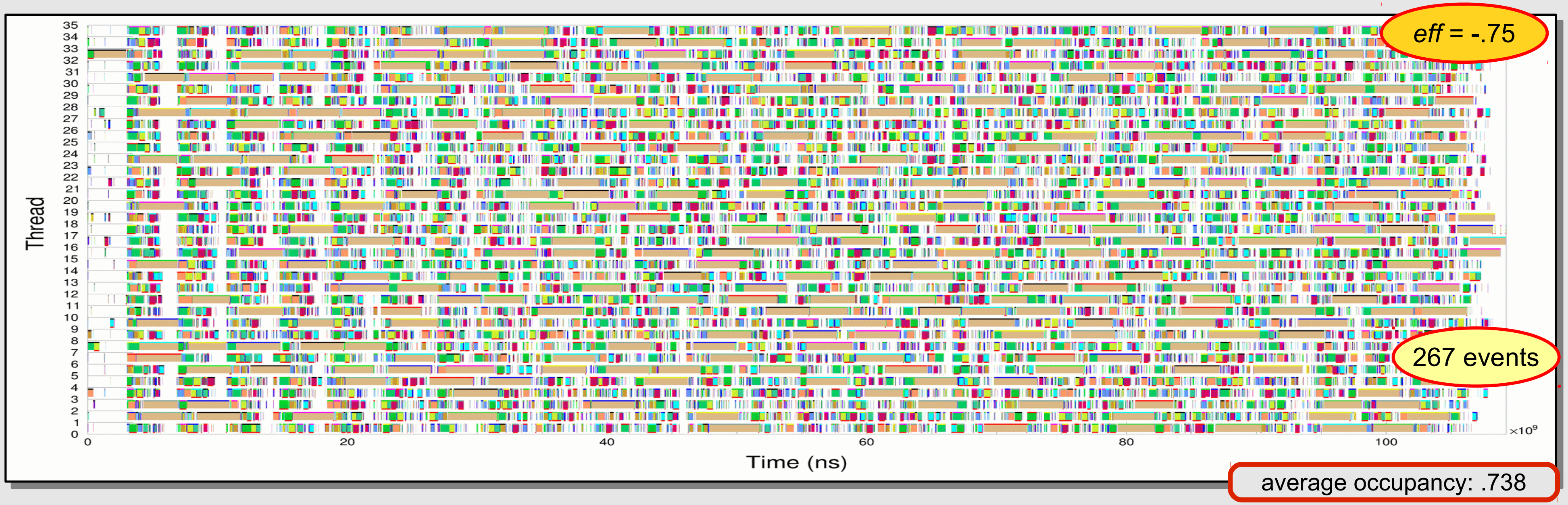
threads: 35
concurrent events: 10
offload frac: 0.9
offload eff: 0.75 -> -0.75



eff = .75

100 events

average occupancy: .477



eff = -.75

267 events

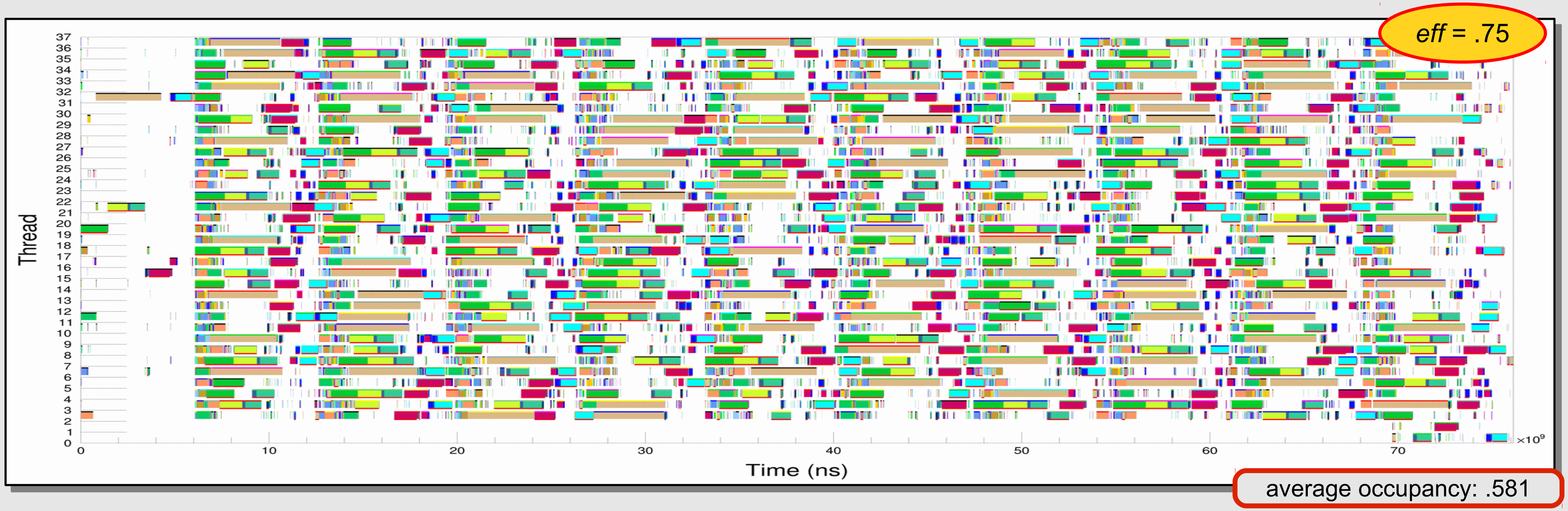
average occupancy: .738

Offloading Algorithms not on Critical Path

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

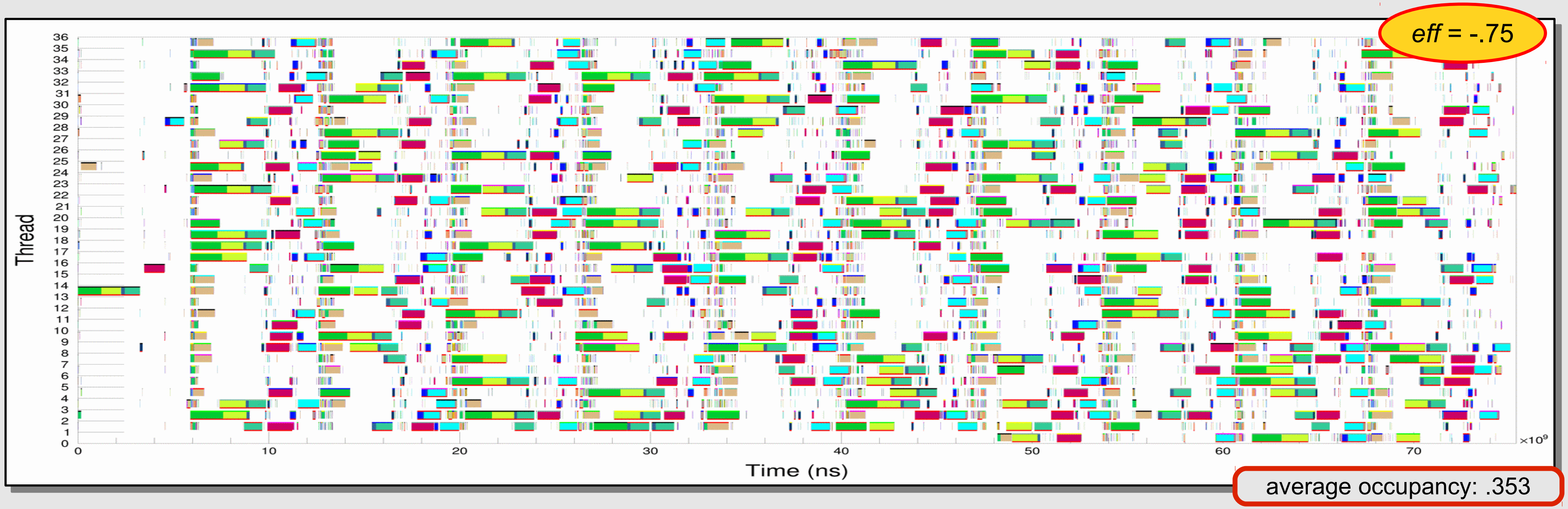
- Total throughput is comparable, but one has significantly higher occupancy than the other
  - must increase concurrency to maximize throughput

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



eff = .75

average occupancy: .581

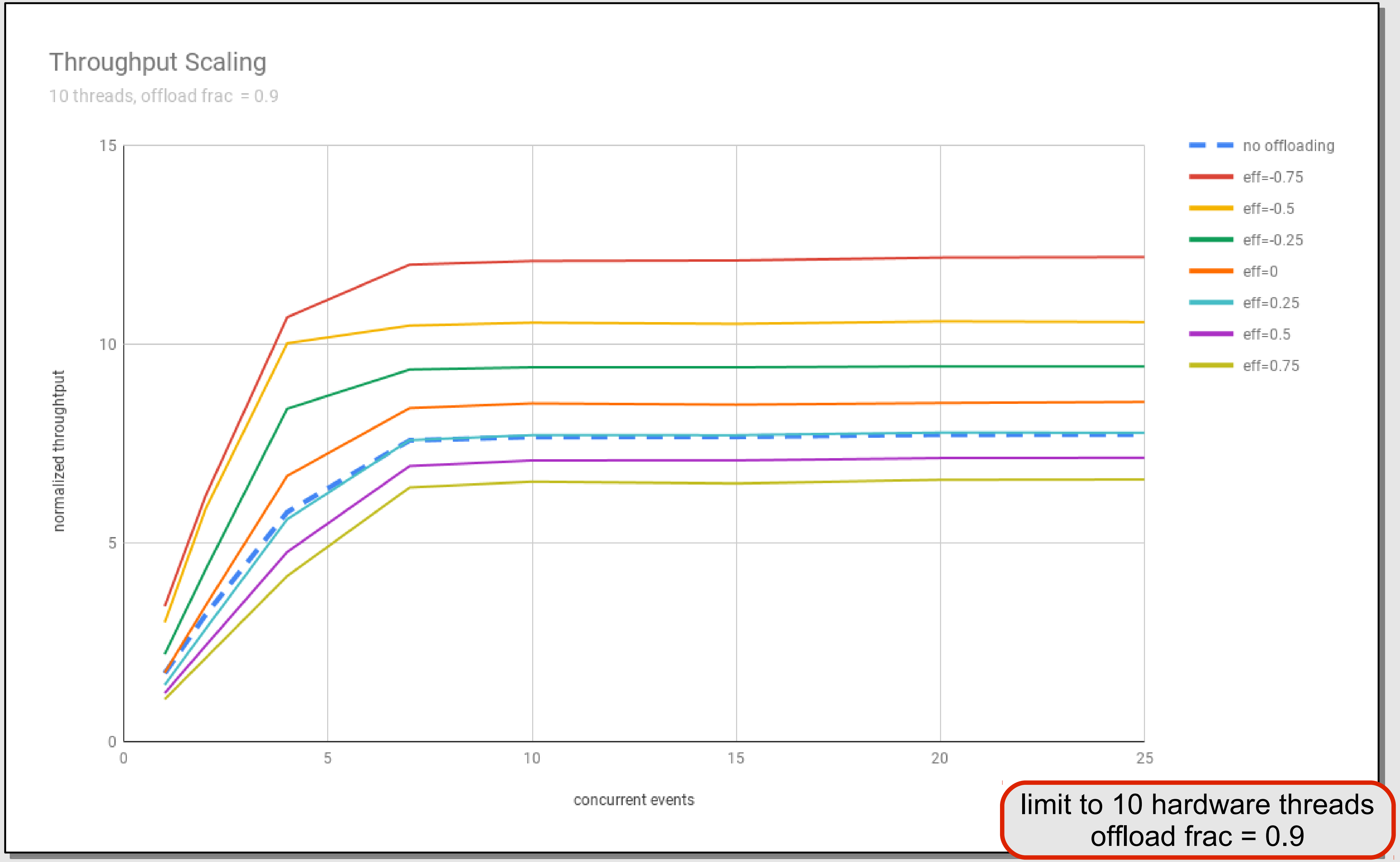


eff = -.75

average occupancy: .353

Oversubscription

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



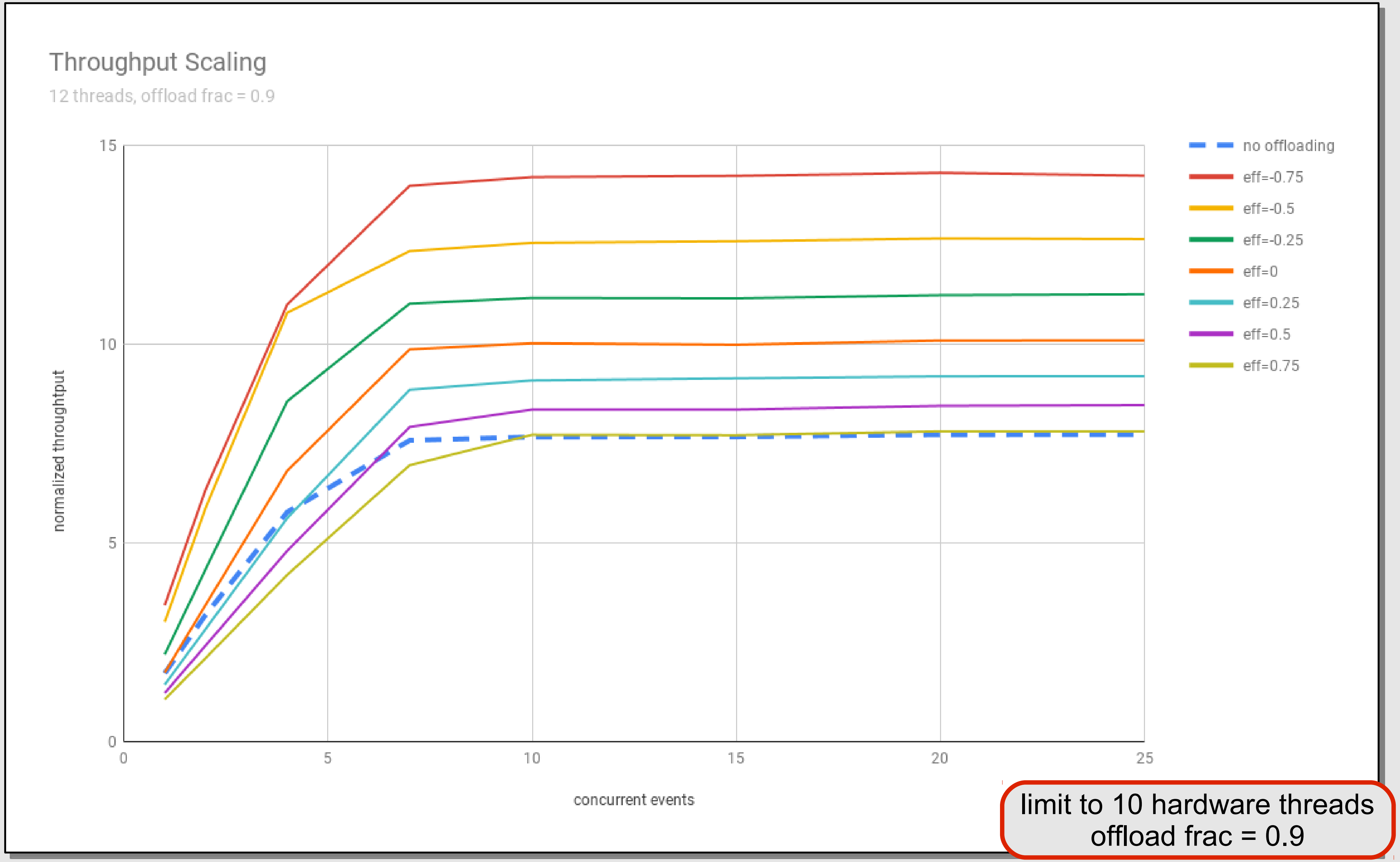
limit to 10 hardware threads
offload frac = 0.9

Oversubscription

- 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



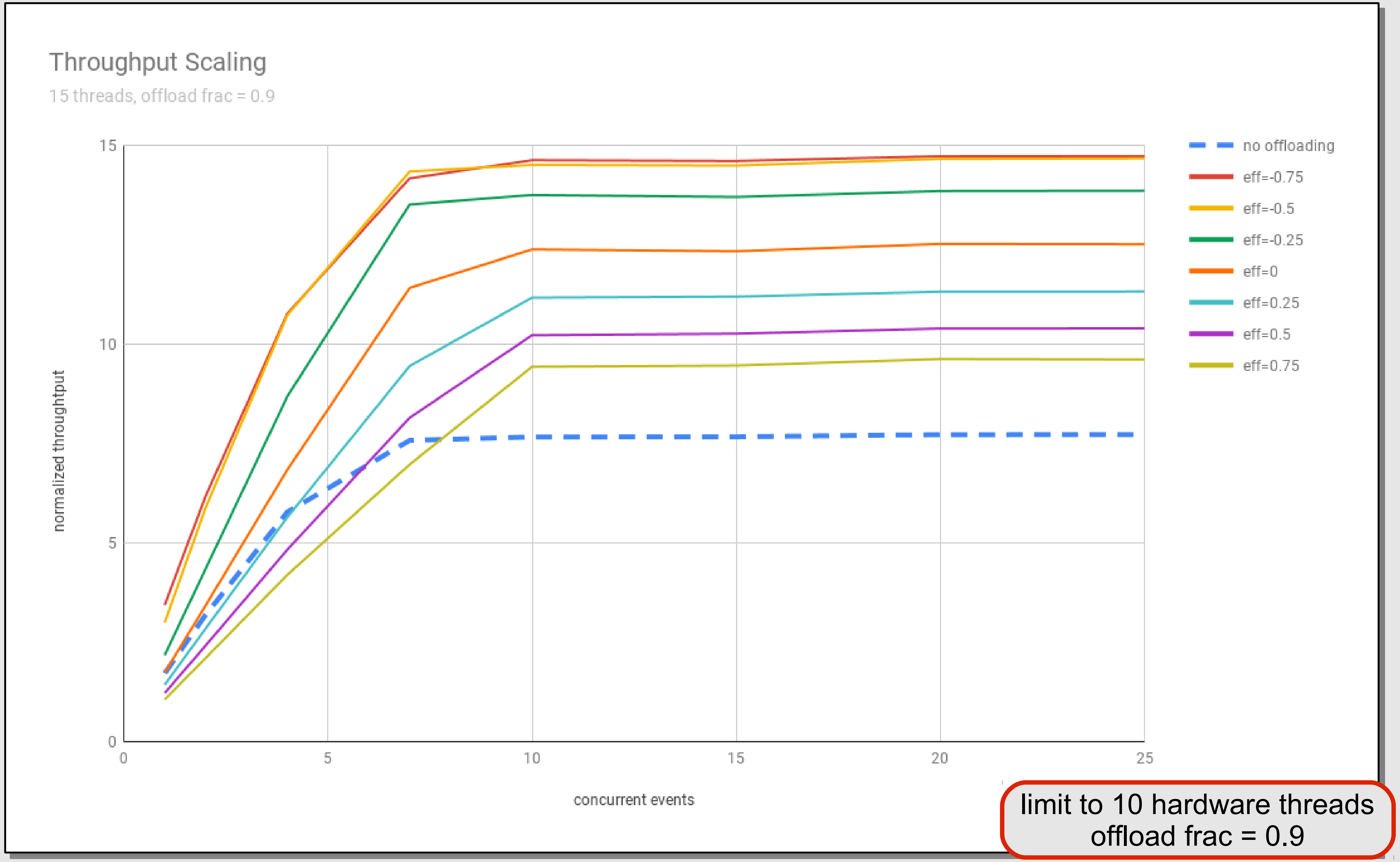
limit to 10 hardware threads
offload frac = 0.9

Oversubscription

- 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



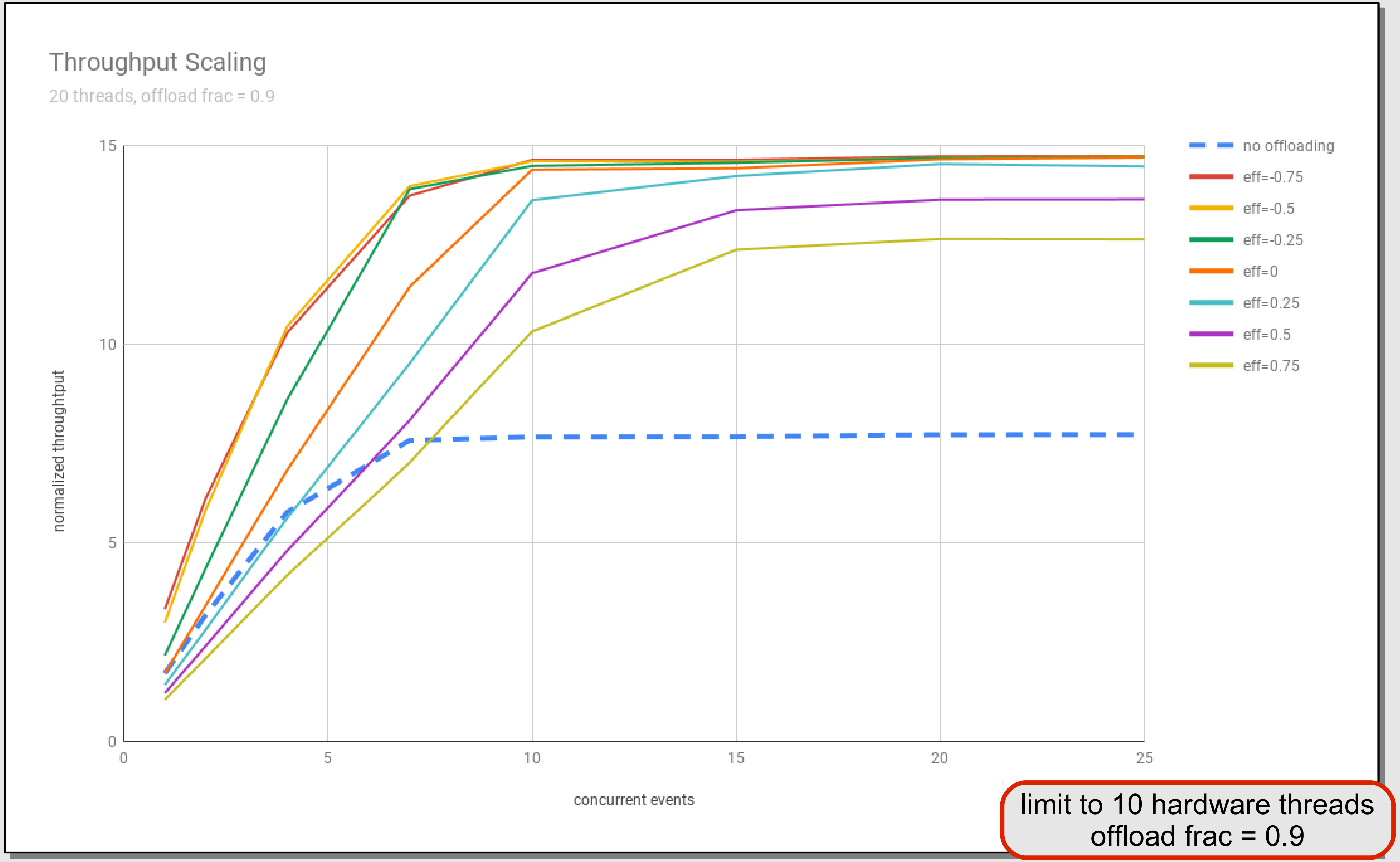
limit to 10 hardware threads
offload frac = 0.9

Oversubscription

- 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



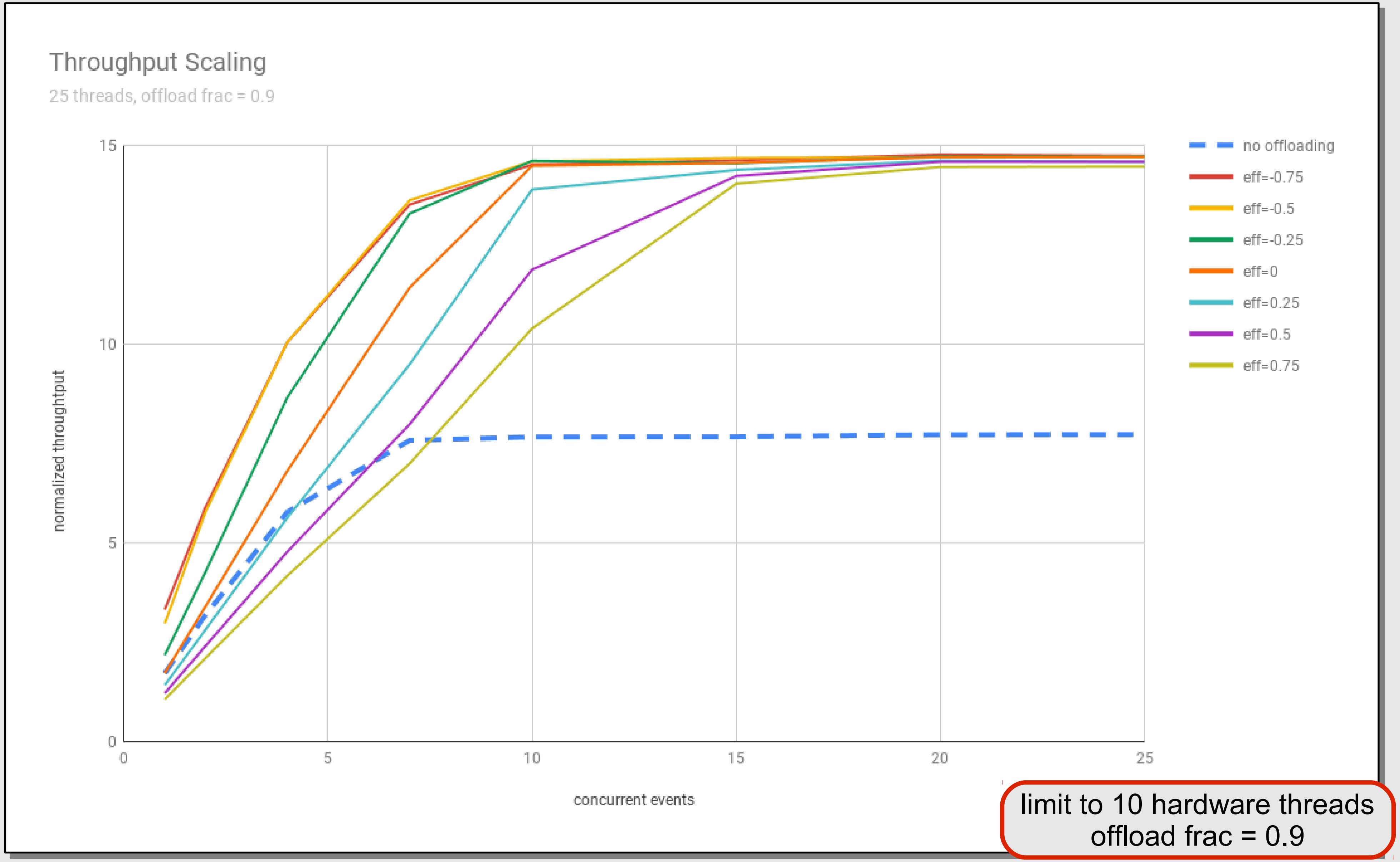
limit to 10 hardware threads
offload frac = 0.9

Oversubscription

- 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



limit to 10 hardware threads
offload frac = 0.9

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

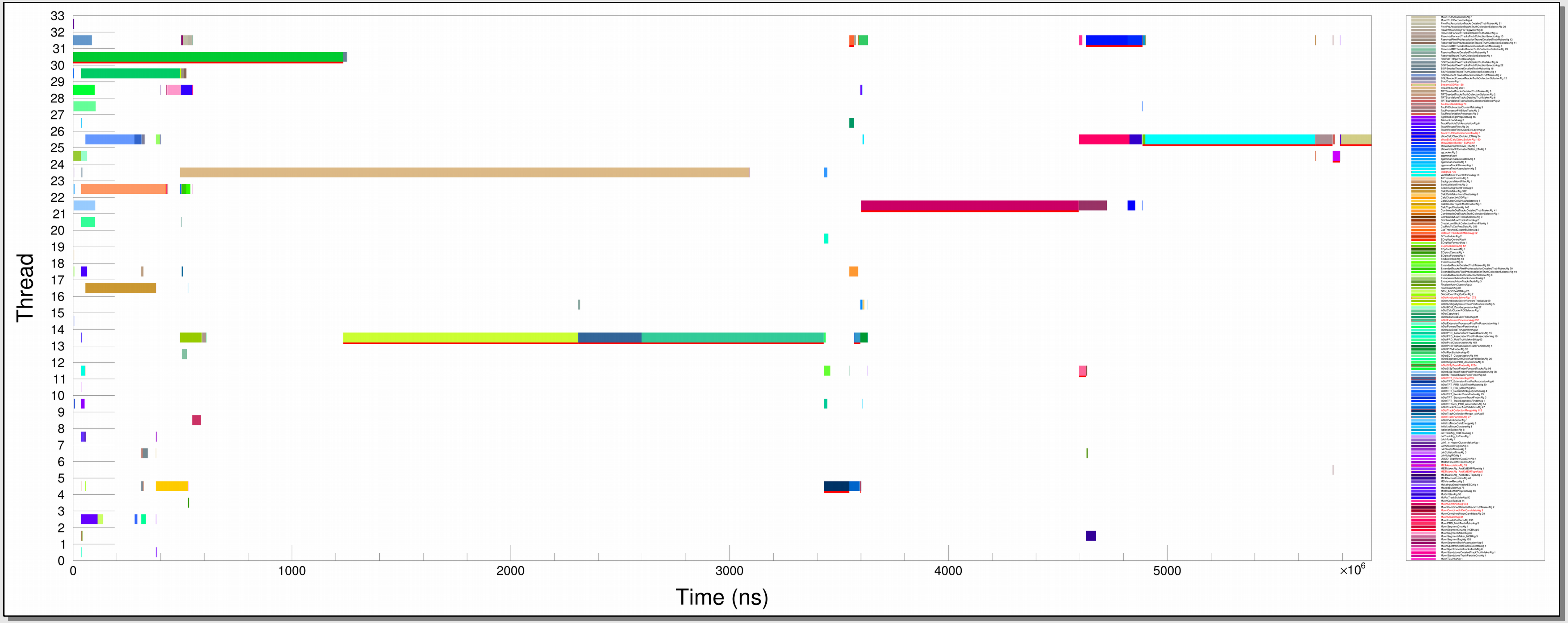
- Scheduling framework modifications to offload Algorithms to accelerators are relatively minimal
  - results are not particular to Gaudi/ATLAS, but applicable to most task based schedulers
- Simulated throughput studies show that offloading Algorithms on the critical path can be much more advantageous than others
  - rewriting these Algorithms for the accelerator is an exercise left for the implementer....
  - offloading other Algorithms may require increasing the number of concurrent events to maximize throughput
- Algorithms don't need to run exceptionally efficiently (faster than on the CPU) on the accelerator
  - inefficient accelerator usage can be offset by increasing number of concurrent events
- Oversubscription of hardware threads on the CPU is essential to maximizing overall throughput
  - threads that offload Algorithms are basically sleeping until the accelerator returns
  - in our scenario the cost of context switching in negligible enough to not affect performance

Extra Slides

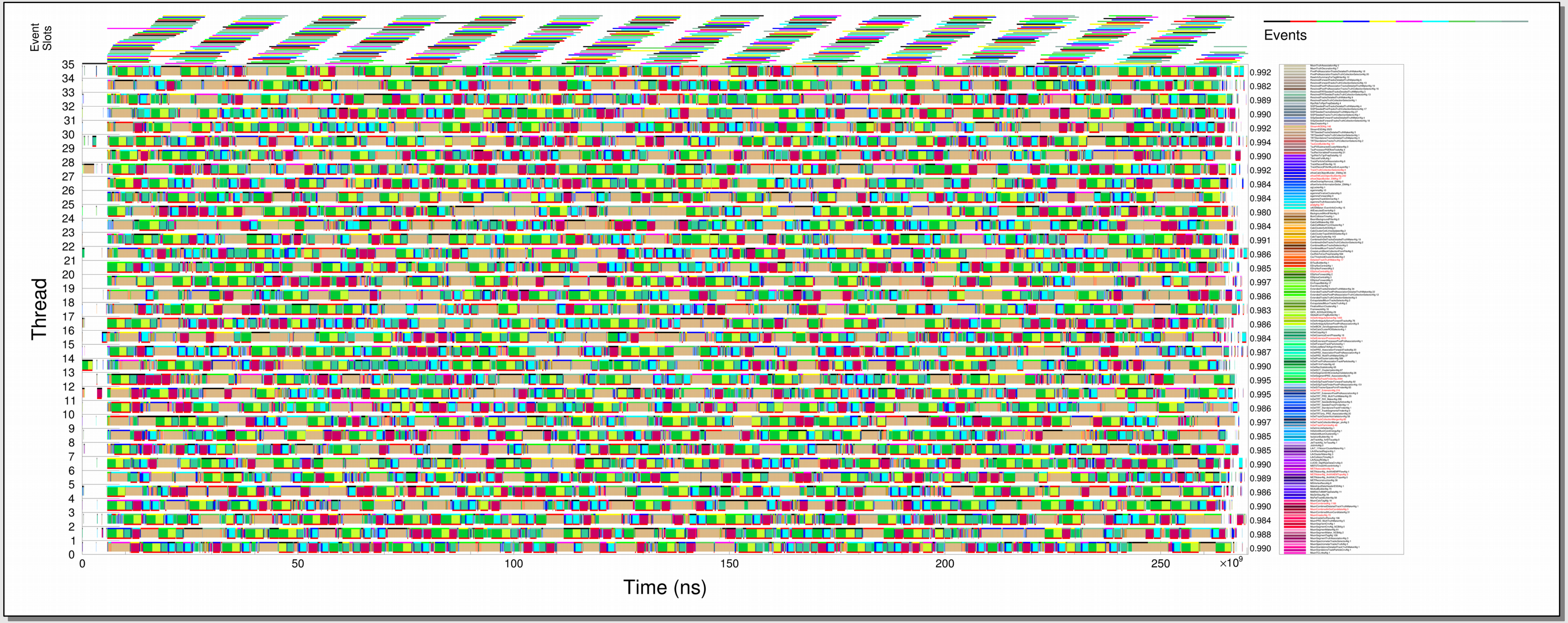
Algorithms on the Critical Path

```
DetailedTrackTruthMakerAlg
EDpfIsoCentralAlg
eflowEMCaloObjectBuilderAlg
eflowObjectBuilder_EMAlg
InDetAmbiguitySolverAlg
InDetExtensionProcessorAlg
InDetSiSpTrackFinderAlg
InDetTrackCollectionMergerAlg
InDetTrackParticlesAlg
InDetTRT_ExtensionAlg
jetalgAlg
METAssociationAlg
METMakerAlg_AntiKt4EMTopoAlg
MuonCombinedAlg
MuonCombinedInDetCandidateAlg
MuonCreatorAlg
StreamAODAlg
TauCoreBuilderAlg
TrackTruthCollectionSelectorAlg
```

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



- timeline chart for 35 concurrent evts w/ 35 threads, no offloading, 500 events


Event Slots
Thread
×10^9
Time (ns)
Events