Overview of Accelerators

(not the particle kind)

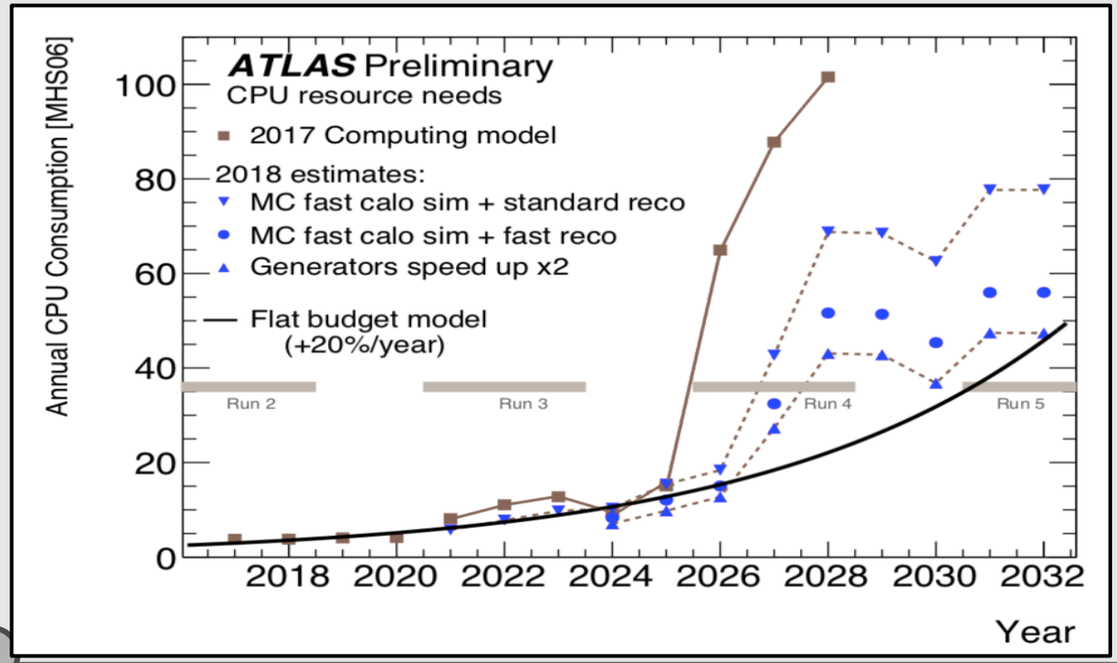
Charles Leggett

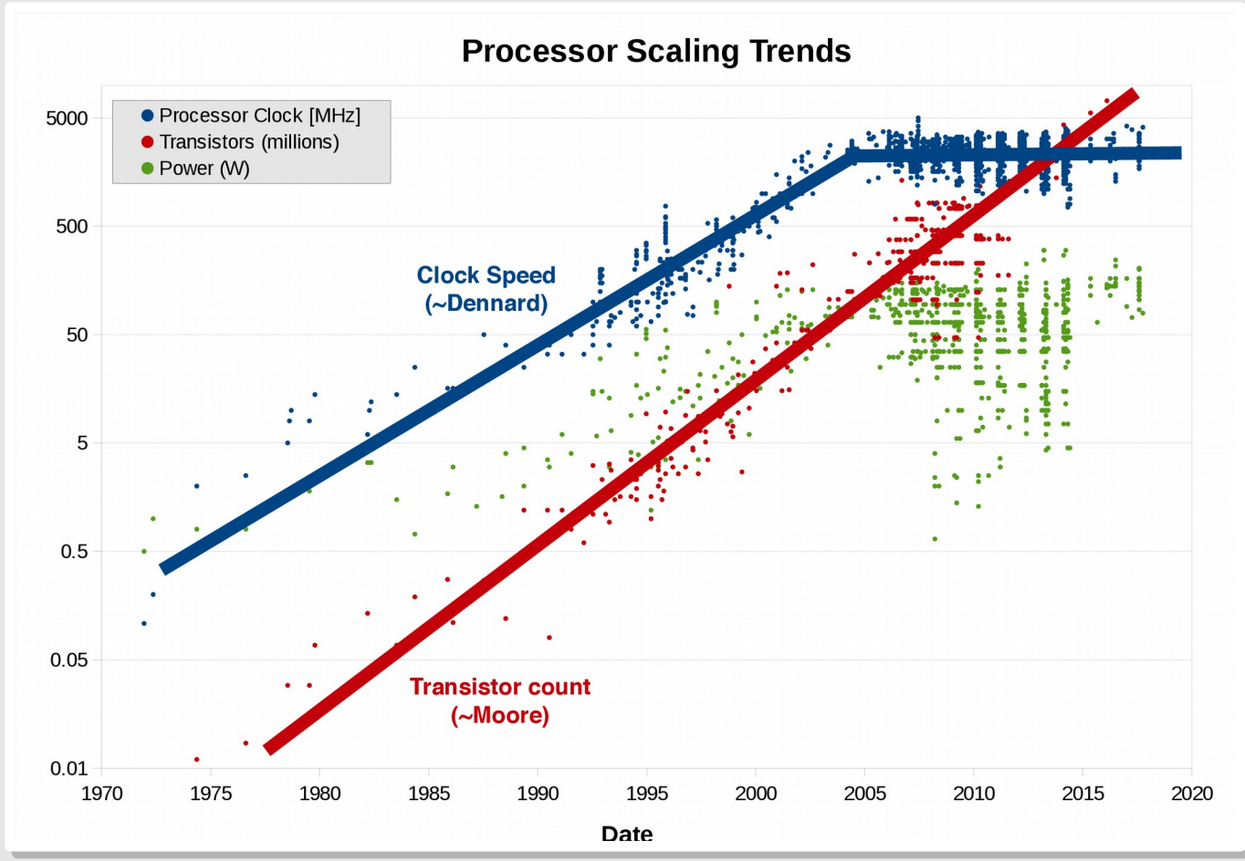
Joint HSF/OSG/WLCG Workshop

Mar 19 2019

The Past Through Tomorrow

- 10 years ago we saw and predicted a major change in computing architecture driven by CPU design limitations
  - smaller, less powerful CPUs
  - less memory per core
- In response, we have heavily invested in multi-threading to better make use available resources





- Unfortunately, we will need **much** more computing power in the not so distant future than we have budgeted for
  - though it doesn't look as bad as it did last year...
- Where can we find it? HPC?

Next Generation HPC/HTC 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 Intel Kaby/Cascade Lake Xeon
    - "single precision GPU subsystem"
  - Argonne: **Aurora** (2021) → **A21**
    - 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**

Next Generation HPC/HTC 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 Intel Kaby/Cascade Lake Xeon
    - "single precision GPU subsystem"
  - Argonne: **Aurora** (2021) → **A21**
    - exascale
    - ~~"novel architecture" -> maybe CSA?~~
    - not novel at all: next generation Xeon + X^e GPU (single die?) + Optane persistent memory
  - 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**

CPU vs GPU

- Why are GPUs not well suited for general computing?
- CPUs are suited to serial processing, can do lots of different things, handle branches, branch mis-predictions efficiently, multiple levels of caches
- GPUs are have much simpler computational internals, no/poor branch handling, small cache, but are very "wide", with thousands of simple cores operating in parallel
  - originally designed to take many independent vertices, turn them into polygons, remove overlaps, shade them, and display them on a screen.

NVidia V100

- 6 Graphics Processor Cluster
- 42 Texture Processor Cluster
- 84 Streaming Multiprocessor
  - 4x 8 FP64
  - 4x 16 FP32
  - 4x 16 INT32
  - 2 Tensor Core

- 7.8 TFLOP FP64
- 15.7 TFLOP FP32
- 125 TFLOP Tensor matrix mult
- 300 W

CPU vs GPU

- Why are GPUs not well suited for general computing?
- CPUs are suited to serial processing, can do lots of different things, handle branches, branch mis-predictions efficiently, multiple levels of caches
- GPUs are have much simpler computational internals, no/poor branch handling, small cache, but are very "wide", with thousands of simple cores operating in parallel
  - originally designed to take many independent vertices, turn them into polygons, remove overlaps, shade them, and display them on a screen. And now ray trace.

NVidia V100

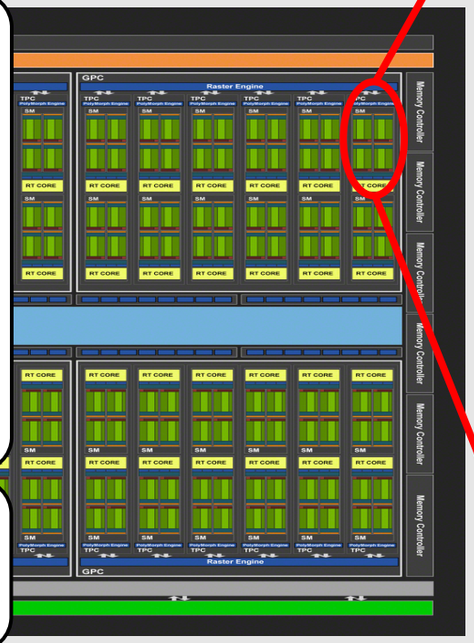
- 6 Graphics Processor Cluster
- 42 Texture Processor Cluster
- 84 Streaming Multiprocessor
  - 32 FP64
  - 64 FP32
  - 64 INT32
  - 2 Tensor Cores

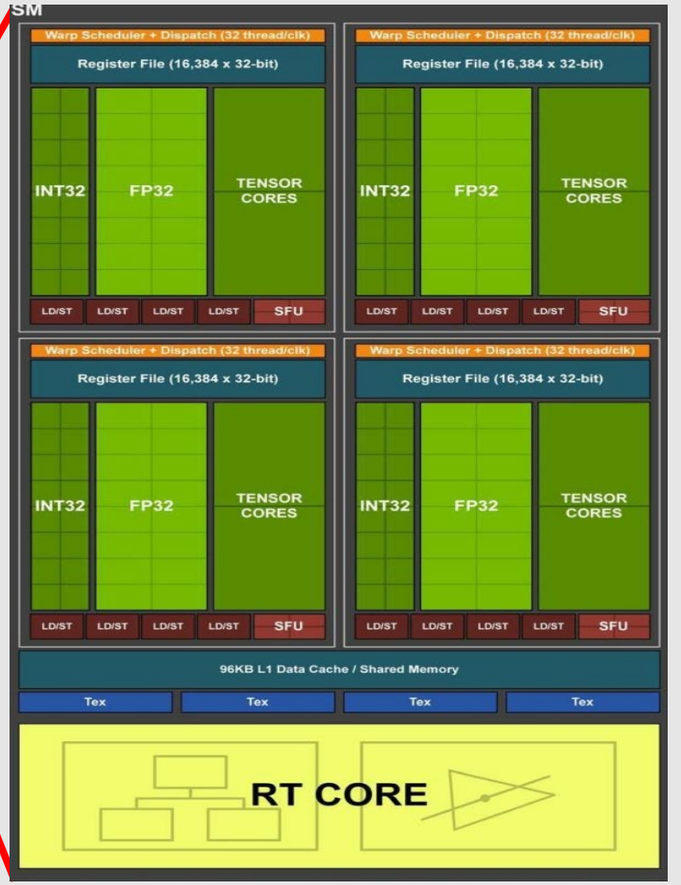
- 7.8 TFLOP FP64
- 15.7 TFLOP FP32
- 125 TFLOP Tensor matrix mult
- 300 W

NVidia Titan RTX

- 6 Graphics Processor Cluster
- 36 Texture Processor Cluster
- 72 Streaming Multiprocessor
  - **2 FP64**
  - 64 FP32
  - 64 INT32
  - 8 Tensor Cores
  - 1 RT Core

- 0.51 TFLOP FP64
- 16.3 TFLOP FP32
- 425 TFLOP Tensor matrix mult
- 280 W





CPU vs GPU

- Why are GPUs not well suited for general computing?
- CPUs are suited to serial processing, can do lots of different things, handle branches, branch mis-predictions efficiently, multiple levels of caches
- GPUs are have much simpler computational internals, no/poor branch handling, small cache, but are very "wide", with thousands of simple cores operating in parallel
  - originally designed to take m… turn them into polygons, rem… and display them on a scree…

RT Core

- Straight line extrapolation to intersect with triangle
- Not great for us
- Maybe future generation will be able to handle curved lines?

NVidia V100

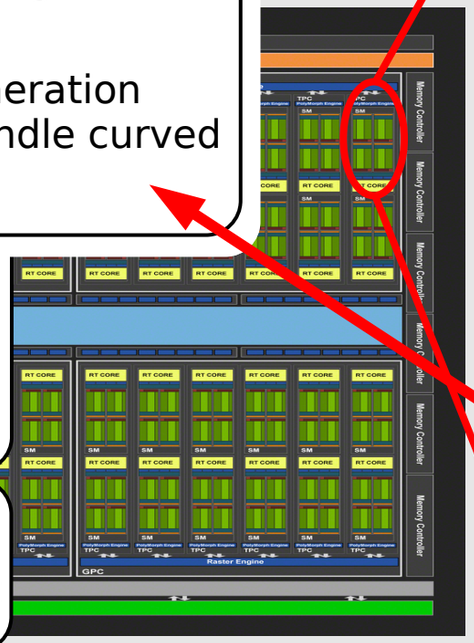
- 6 Graphics Processor Cluster
- 42 Texture Processor Cluster
- 84 Streaming Multiprocessor
  - 32 FP64
  - 64 FP32
  - 64 INT32
  - 2 Tensor Cores

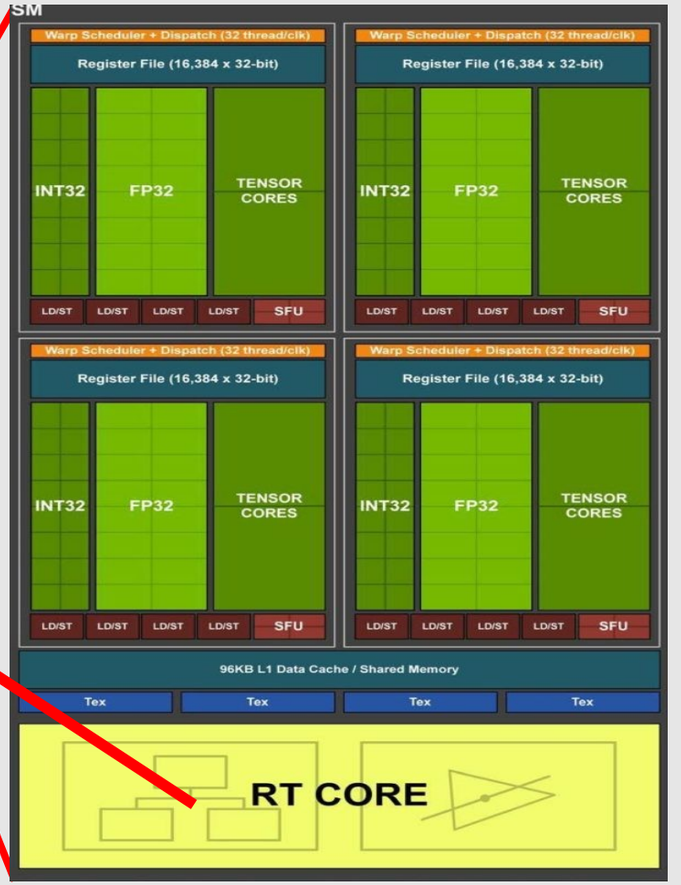
- 7.8 TFLOP FP64
- 15.7 TFLOP FP32
- 125 TFLOP Tensor matrix mult
- 300 W

NVi…

- 6 Grapl…
- 36 Text…
- 72 Stre…
  - **2 FP…**
  - 64 FP32
  - 64 INT32
  - 8 Tensor Cores
  - 1 RT Core

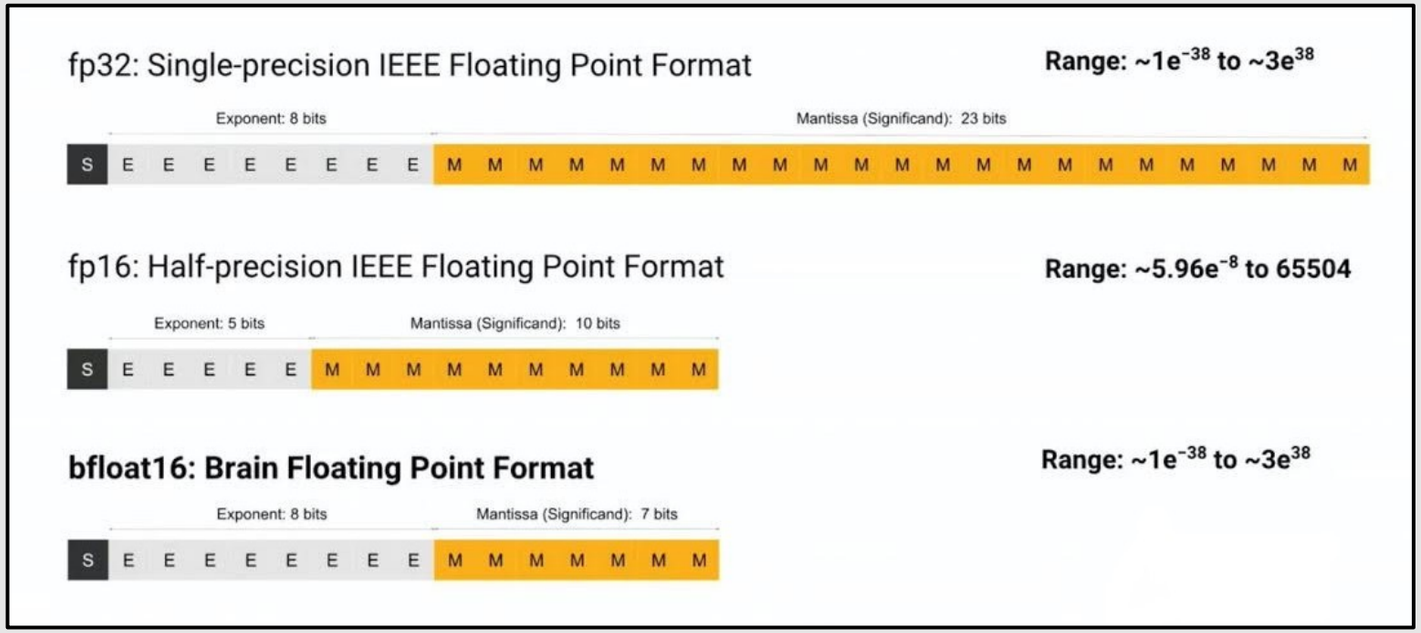
- 0.51 TFLOP FP64
- 16.3 TFLOP FP32
- 425 TFLOP Tensor matrix mult
- 280 W





Precision

- A lot of silicon on accelerators is devoted to single or reduced precision ops
  - FP32, FP16, INT32, Tensor Core
- Google introduced a new format **bfloat16** for its gen-3 TPUs



- supported at the hardware level in Intel's 2020 Cooper Lake architecture
- emulated in microarchitecture in 2019 Cascade Lake
- 2x greater throughput than FP32

- We need to figure out in which circumstances we don't need to do everything with double precision, and take advantage of the significantly faster processing speed for low and mixed precision ops

GPUs in HEP

- There have been many attempts to make use of GPUs for HEP
- In general, GPUs are not really practical to use as just another regular compute engine
  - excel at some tasks, fail horribly at others
  - code/kernels need to be rewritten to take advantage of hardware / memory layout
  - work best with SIMD style processing
- Some types of HEP code are well suited for GPUs
  - some ML based pattern recognition (*eg* tracking)
  - G4 EM and neutral physics
  - Calorimeter seeding / clustering
  - GeoModel: convert to Pixar **U**niversal **S**creen **D**escription + Hydra
- Some things don't
  - anything branchy, sorts, etc
  - lots of code, little data
- Just because we have one way of doing something now that's designed for a CPU, doesn't mean we need to do it the same way on a GPU
  - track fitting w/ kalman -> machine learning for pattern reco

Writing Code for GPUs

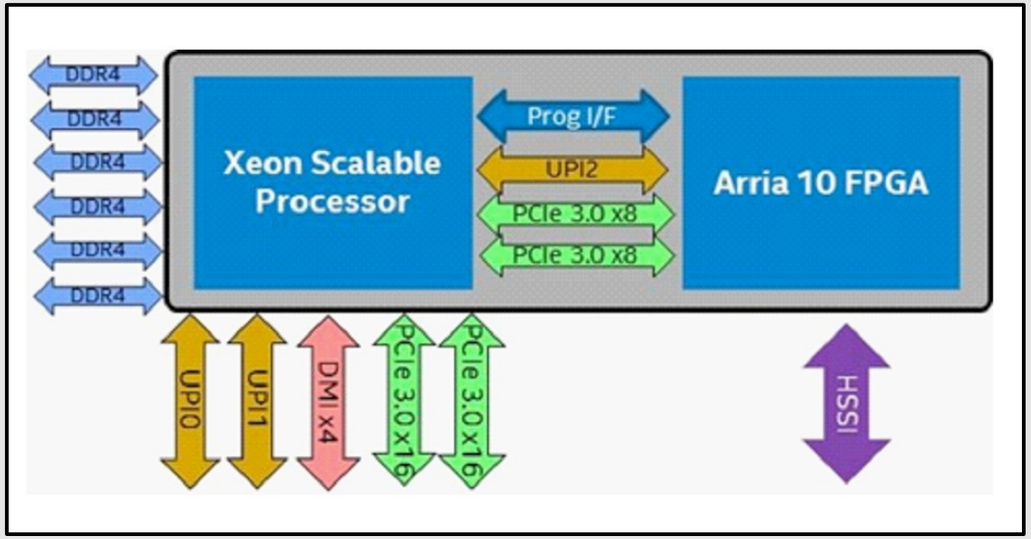
- Two main low level languages for writing code for GPUs
  - **CUDA** (Compute Unified Device Architecture)
    - written specifically for NVidia. Won't run on AMD
  - **OpenCL**
    - open standard, will run (in theory) on any GPU (or CPU)
    - not all versions are supported by all GPUs
      - » 2.2 on nothing (but looks very interesting on paper)
      - » 2.1 only on CPUs
      - » 2.0 on most AMD GPUs
      - » 1.2 on most modern NVidia GPUs
    - same code can run on CPU and GPU

- High level languages / extensions
  - OpenACC
    - extension pragmas like OpenMP
      - » supposed to be merged with OpenMP?
    - few compilers support it
  - Thrust
    - `stl`-like libraries for GPUs
  - SYCL
    - single source CPU/GPU
    - C++ 11-17
  - StarPU, Kokkos

Field Programmable Gate Arrays

- Xilinx is best known FPGA manufacturer (now)

- In 2015 Intel bought out Altera, an FPGA manufacturer
  - In 2018 they announced a Xeon Skylake CPU / Arria-10 FPGA hybrid chip
- 2 UPI channels to link 2 chips together
- On chip memory latency between FPGA and Xeon probably comparable to latency between 2 Xeon NUMA nodes

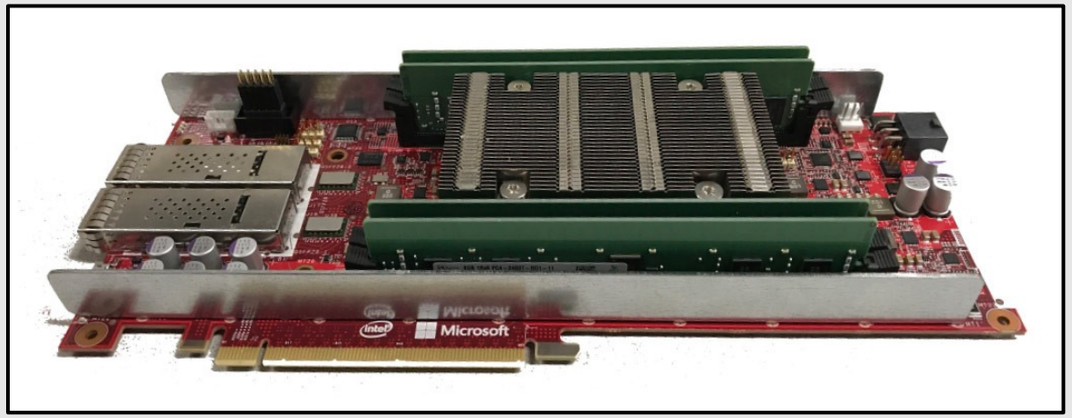


- Programs are written in VERILOG or VHDL
  - you really need to understand the hardware

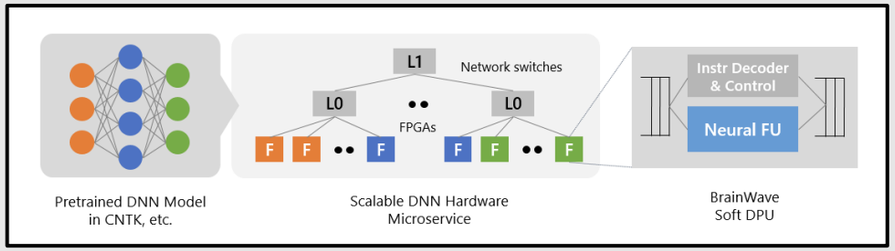
- High Level Synthesis
  - generate standard Register-Transfer Level code for FPGA from more common C/C++ code
  - hardware specific compilers
  - not as fast/efficient as VHDL, but much easier to use

Microsoft Project Brainwave

- Dedicated, cloud based FPGA system for neural network processing
  - provided by Microsoft, deployed to Azure cloud
  - based on Intel Stratix 10



- Only supports a small set of NN models:
  - ResNet 50
  - ResNet 152
  - DenseNet-121
  - VGG-16



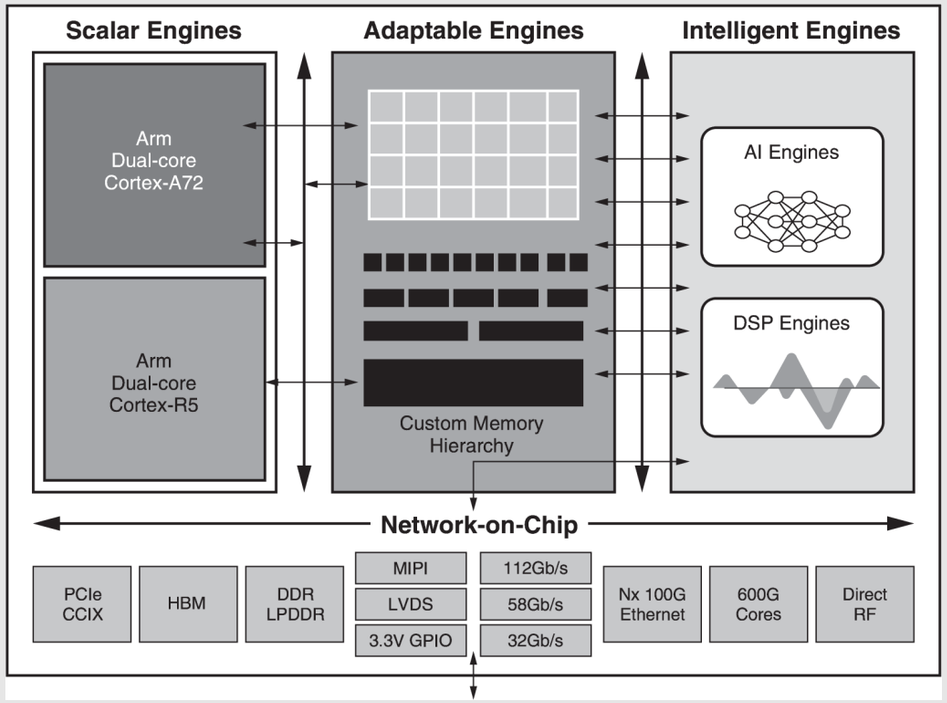
- Used for inference/classification, not training

- You don't need to know how to program a FPGA to use it!

cool talk by J. Ngadiuba at ACAT

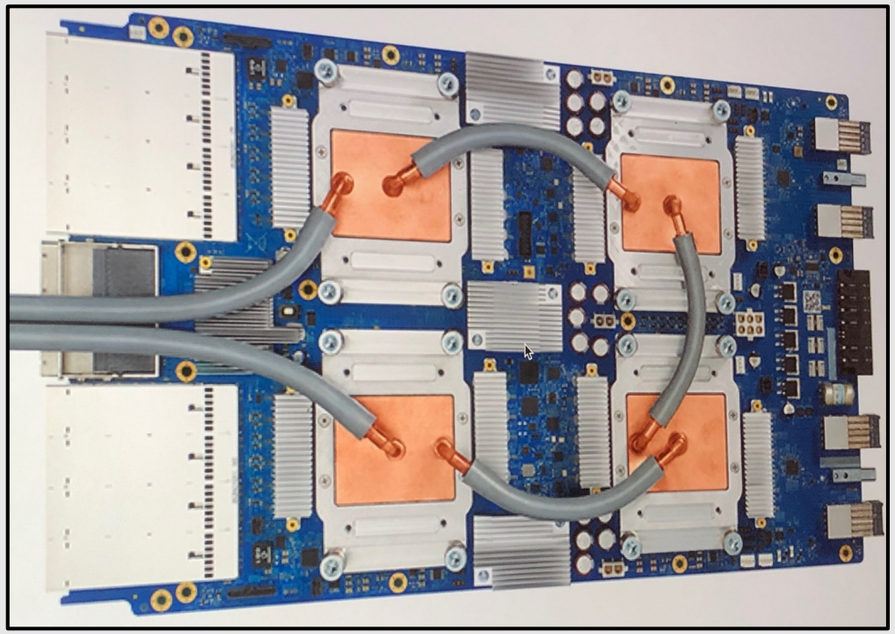
Xilinx Versal ACAP

- Low power, 7nm "Adaptive Compute Accelerator Platform"
  - similar in concept to Intel CSA, though different execution
- Combination of ARM CPUs (Cortex A72, R5), FPGA, Intelligent Engines (AI, DSP, …)
  - different versions can have different Intelligent engines
- Example of next generation domain specific processing
  - different targets for ML, machine vision, automotive, and 5G wireless
  - good indication of where the market is headed with specialized and specializable accelerator devices



Google Tensor Processing Unit

- Custom secret ASIC, specifically designed to run TensorFlow
  - optimized for high volume of low precision matrix manipulation and other operations
  - can now handle floats as well as ints, so suitable for both training and inference

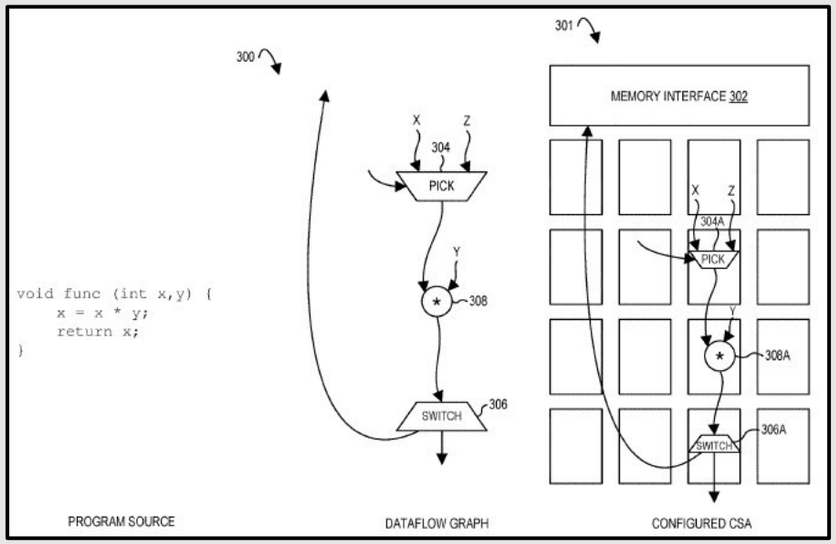


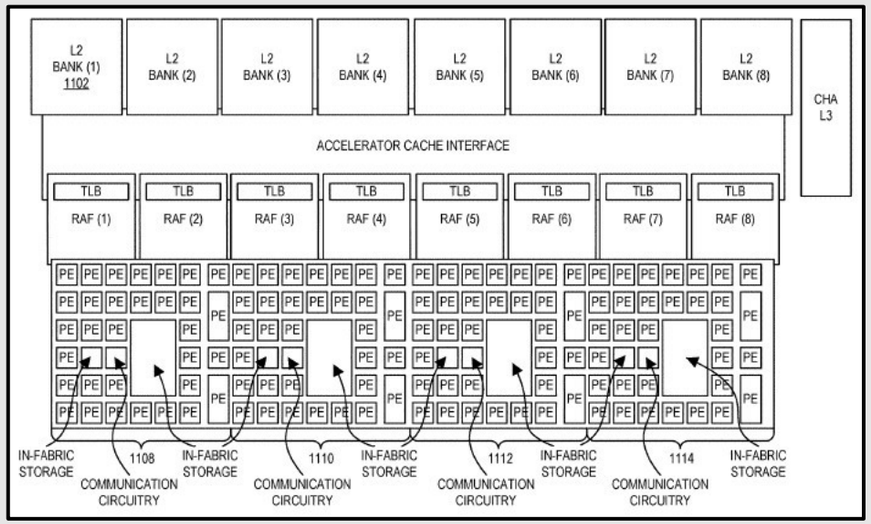
- Google has been using some version of these for years in their search engines

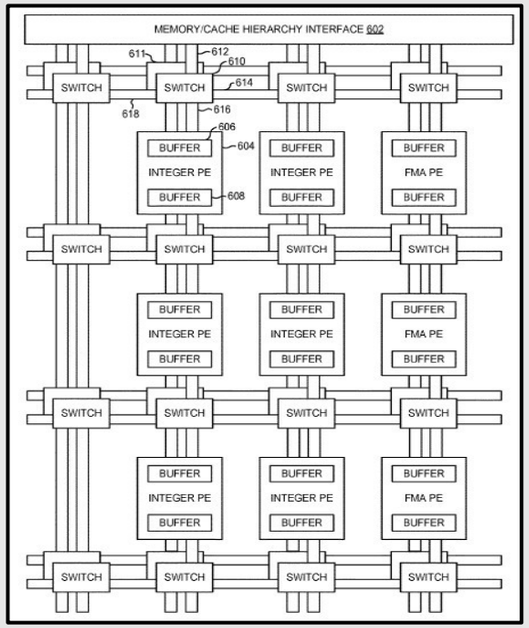
- You can't buy them, but Google has now made them available for use as a cloud service
  - Cloud TPU v2: 180 teraflops, 64 GB High Bandwidth Memory (HBM)
  - Cloud TPU v3: 420 teraflops, 128 GB HBM
  - Cloud TPU v2 Pod Alpha: 11.5 petaflops, 4 TB HBM, 2-D toroidal mesh network

Intel CSA: Configurable Spatial Accelerator

- Dataflow engine, that works directly on graphs that programs create before they are compiled down to the CPUs.
- Dataflow laid down on static hardware etched in silicon
  - not created on the fly like an FPGA, graph mapped to matching elements in the silicon
  - take C, C++, Fortran source, and directly target the CSA







- Very dense compute and memory, very high energy efficiency
  - only the elements needed for a particular dataflow are activated as a program runs
- Array of Processing Elements will be heterogeneous, customized to different flows
- Include a low end Xeon to handle operations that don't map well to CSA
- 16x density of Fused Multiply-Add units compared to Xeon, FLOPS order of magnitude higher

https://www.nextplatform.com/2018/08/30/intels-exascale-dataflow-engine-drops-x86-and-von-neuman

Accelerators and Machine Learning

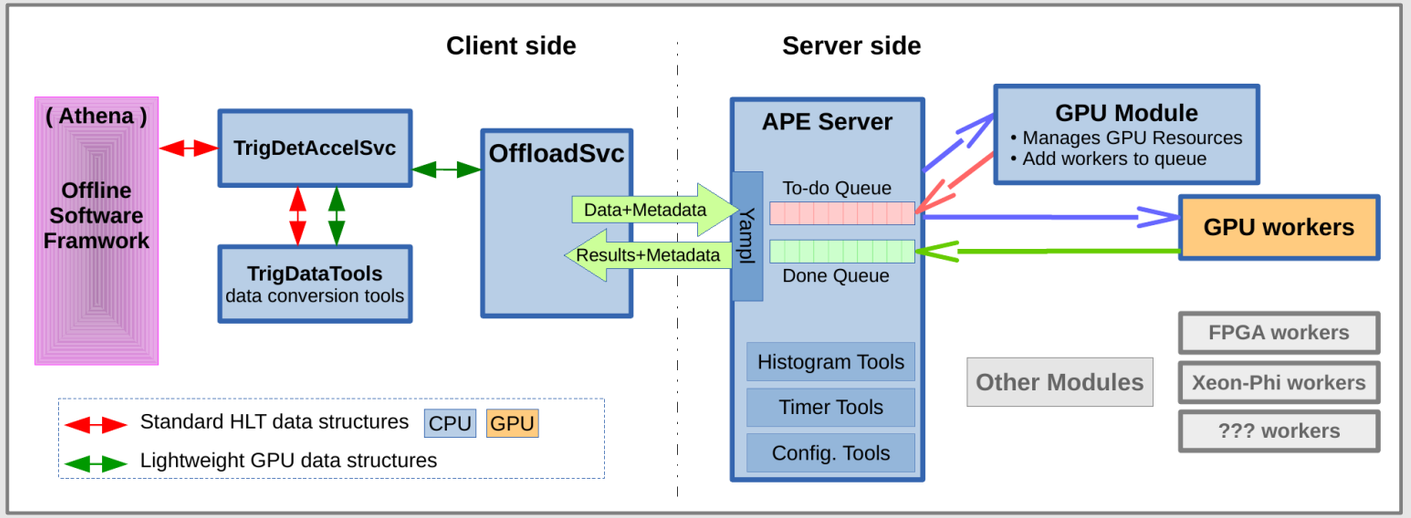
- We are seeing increasing support for accelerator back ends for Machine Learning from many providers
  - GPU: NVidia Tensor Core
    GPU hardware in general
  - ASIC: Google TensorFlow Processor Unit (TPU)
  - FPGA: Xilinx Versal
    Azure / Project Brainwave

- Instead of converting code to run on a specific GPU, we can bypass this difficult step by reformatting the analysis to use Machine Learning
  - increasing number of products that provide C/C++/python ML front ends and do the heavy lifting of writing the CUDA/Verilog/HLS kernels for you to explicitly target various accelerator technologies
  - take advantage of existing libraries for training and inference
  - make use of pre-configured cloud services (like Brainwave)

- However, even ML proponents say one should take results with a grain of salt….

Current GPU Usage in ATLAS

- Investigated for use in Trigger
  - Track seeding and topological cluster formation in Calorimeter
  - 3 stages:
    - convert ATLAS EDM objects -> plain arrays
    - offload to GPU via APE server
    - retrieve data from GPU, convert to ATLAS EDM
  - Significant speedup found in GPU based processing:
    - individual Alg 10x faster
    - total throughput 1.4x
  - **APE** server can be used to offload to GPU, FPGA, other network sinks
    - APE can accumulate data until a large chunk of work can be offloaded



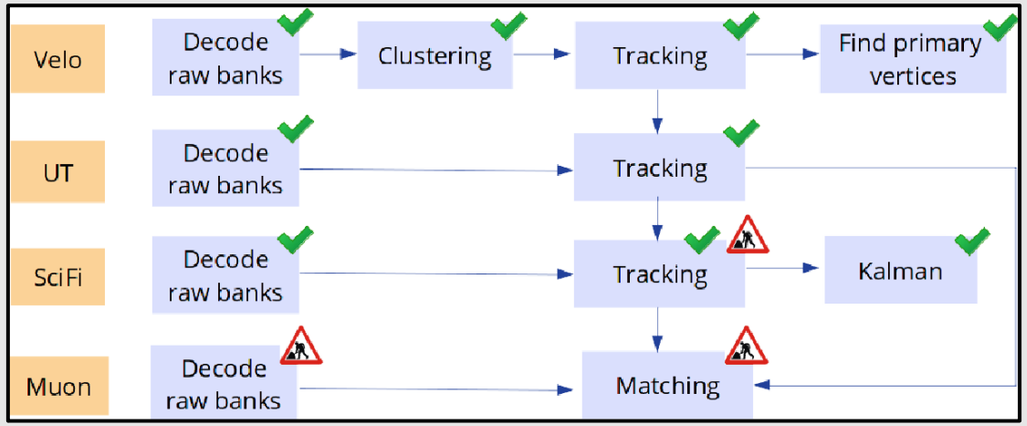
- Hard to saturate GPU (in this limited testbed)
  - required (many) more than 55 clients to saturate a single NVidia GTX1080 (pascal)
    - V100 (Summit/Sierra) is several times more powerful than a GTX1080
- On the framework (Gaudi) side, transparent Alg offloading mechanism already in place.

Accelerator Usage in CMS

- CMSSW already supports transparent offloading of modules to an accelerator. Modules are re-activated when accelerator has finished processing and data is ready
- Investigated full Pixel, HCAL and ECAL online reconstruction on GPU
  - Pixel reconstruction uses ~ 9% of event processing
  - ECAL/HCAL reconstruction use 20%-25% of event processing time
  - lots of inherent parallelism due to independence of channels
  - code to bare metal using CUDA with "naive" implementations
    - optimization improvements in process
  - need to change data format to SoA
- Investigating FPGA usage
  - standalone implementation of FastNNLS in OpenCL
- Also investigating tracking and new detectors (for HL-LHC)
  - track and vertex finding on GPU for the HLT
- See Chris' talk next on CMSSW and Matti's talk on Patatrack

Accelerator Usage in LHCb

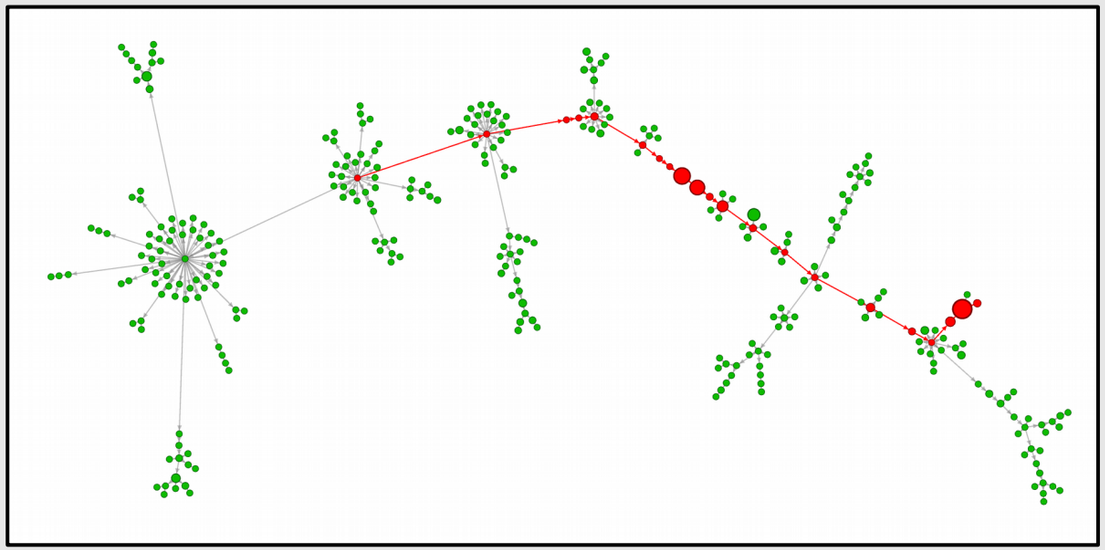
- Run3: LHCb is going to a full software trigger for HLT1 at 30MHz
  - Allen framework R&D project: goal is to see if **entire HLT1 can run on GPU**
    - C++14 and CUDA v10.0
    - lots of work already done
- Mostly clustering and tracking in HLT1
  - keep as many steps on the accelerator as possible
  - convert data from SoA to EDM at end



- LHCb has a very small event size (*cf* ATLAS/CMS)
  - need to gang events to use GPU efficiently
- VELO clustering, and SciFi tracking also done on FPGA
- Not clear how this can be applied to offline reconstruction, but after porting GPU code back to CPU, it ran faster there than original algorithms.
- see Roel's talk

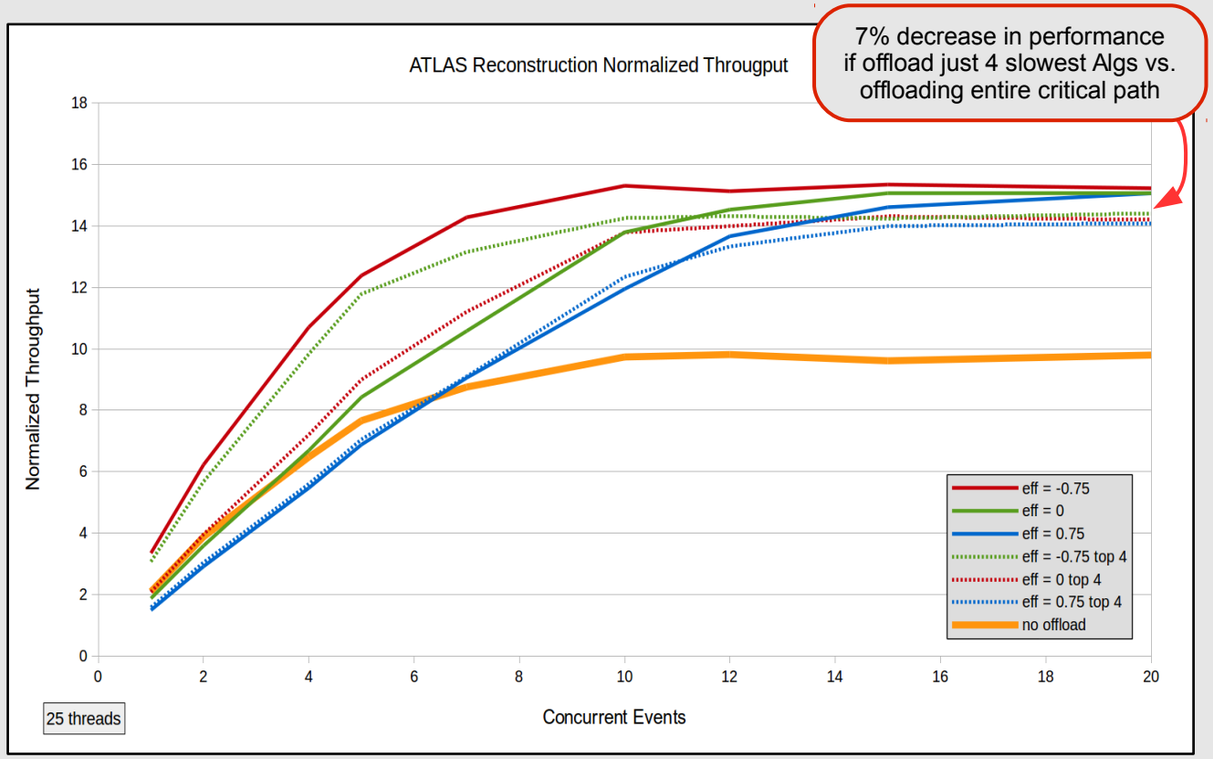
What to Run On An Accelerator (For Offline)?

- Established that writing code for accelerator is hard. So what do we do?
- Can we identify which Algorithms are worth running on an HPC's GPUs?
  - they are easy to port
  - porting them makes a big difference in throughput
- Create a simulation framework for running on a heterogeneous system of CPU + GPU
  - simulate offloading Algorithms to GPU using standard workflows for ATLAS, CMS, LHCb
  - vary parameters such as:
    - which Algs to offload
    - what fraction of Alg's work can be offloaded
    - accelerator efficiency
    - data transfer overhead
- Very important to understand "Critical Path" of Algorithm execution
  - Algorithms that determine event processing time
  - ATLAS: 17/309 Algs, 5.8s/14.6s (40%)
  - CMS: 145/707 Algs, 7.3s/13.4s (55%)
  - LHCb: 11/282 Algs, 234ms/491ms (48%)



Simulating Workflows on Heterogeneous Architectures

- Studies have shown that while there are much greater benefits to offloading Algorithms on the critical path, offloading *something* is still better than offloading nothing.
  - even if an offloaded Algorithm runs *slower* on a GPU than CPU, total event processing throughput is increased
  - maximum throughput may require increasing the number of concurrent events



- It is important to oversubscribe the CPU with more threads than there are hardware threads to keep it busy

- Data transfer overheads will likely not be a significant issue as long as objects are already in a flat structure that is easily digested by the GPU (xAOD are already like that)
  - don't want to have to convert data structures back and forth
  - implications for EDM

see results presented at **ACAT**

Alternate Workflows

- Do we really need to port Algorithms to run on accelerators?

- Instead of converting individual Algorithms, can we identify complete workflows that are genuinely well suited to run on accelerators?
  - event generation
    - madgraph
  - ML network training

- Run entirely different workflows on the CPU and the accelerator
  - some part of CPU devoted to scheduling
  - can we identify other non-ATLAS users, who have GPU dominant workflows that we can piggy back off of?

- Run reconstruction on CPU, bitcoin miner on accelerator
  - use bitcoins to buy a large and ever expanding homogeneous x86 cluster!

Conclusions

- Though we are developing coping strategies, from our (HEP) point of view, the increasing usage of accelerators in HPC/HTC is grim news
  - our code is not well suited for massive parallelism
    - the hardware was designed for high FLOP count, low energy - not with the average user in mind
  - increasing emphasis on mixed/low precision computation
    - that's fine for "deep learning", not so fine for a Higgs analysis
  - it takes time and people to identify and translate kernels for running on accelerators
- In the past, various institutions have offered "help" in getting our code to run on their brand of hardware.
  - it would be crazy to have to do this differently for every new accelerated HTC architecture, especially since they only last ~5 years
  - instead of porting Algorithms one by one, time/money would be better spent finding common, high level solutions that work across ALL platforms
    - *eg* what if we could get OpenCL 2.2 or SYCL support across the board?
    - use ML wherever we can, and take advantage of ML backends for accelerators

What To Do Next

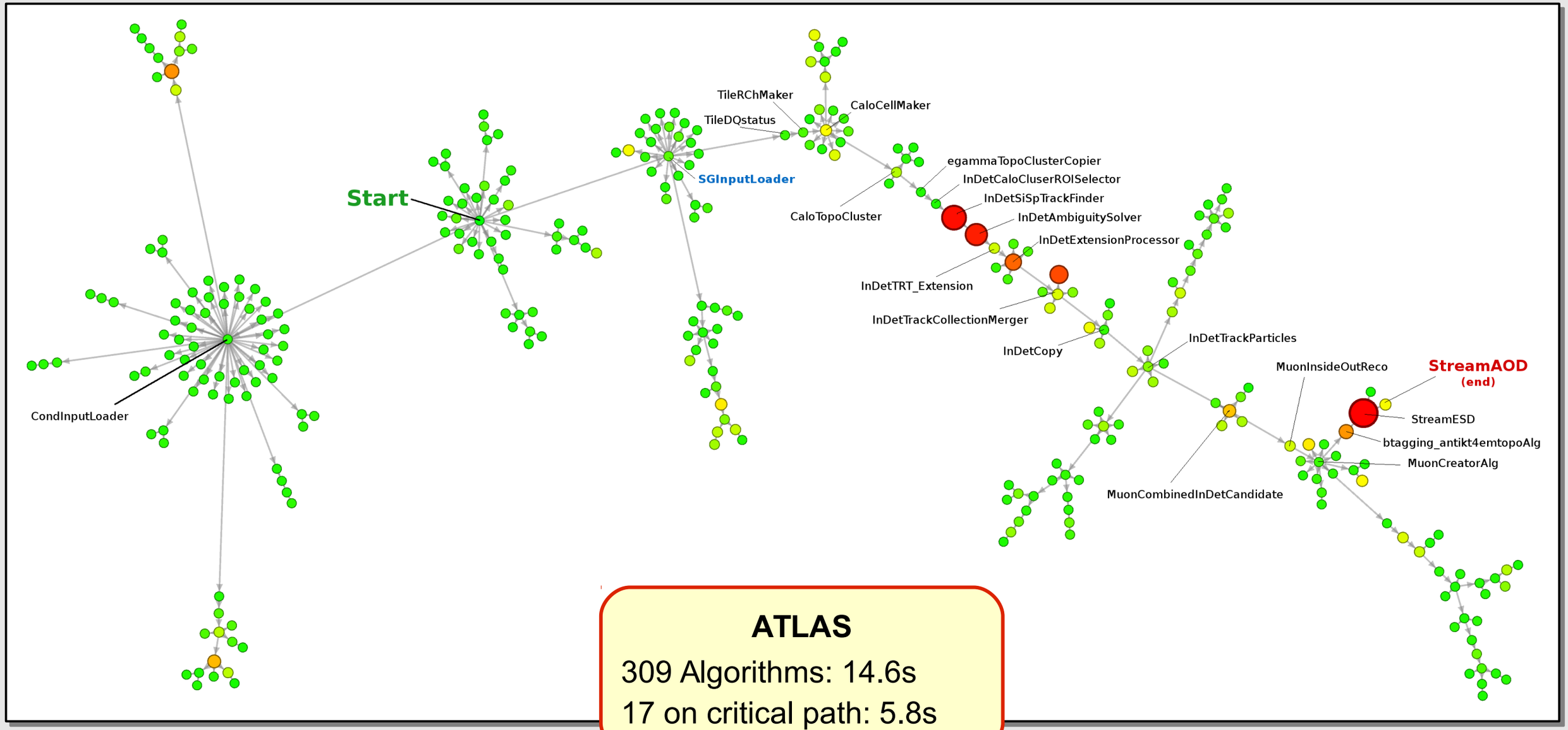
- Short term: port a select few Algorithms to test out the machinery
  - code to bare metal (CUDA) to investigate maximum potential / learn techniques
    - take advantage of existing trigger implementations (LHCb!)
  - incorporate existing ML prototypes into workflows
  - understand EDM implications for flat SoA style structures

- Mid term
  - look for entire workflows that do well on accelerators
  - paradigm shift: investigate more ML implementations of many different Algorithms
    - and find way to trust the results….
  - develop framework/EDM extensions that allow combining of data objects across many events to efficiently saturate accelerator, but still allowing physicists to deal with "events"
  - understand where we can used low/mixed precision computing, and implement it

- Long term
  - invest in developing high level interfaces / languages that will allow use of accelerators without needing to code to bare silicon

fin

ATLAS Precedence Trace

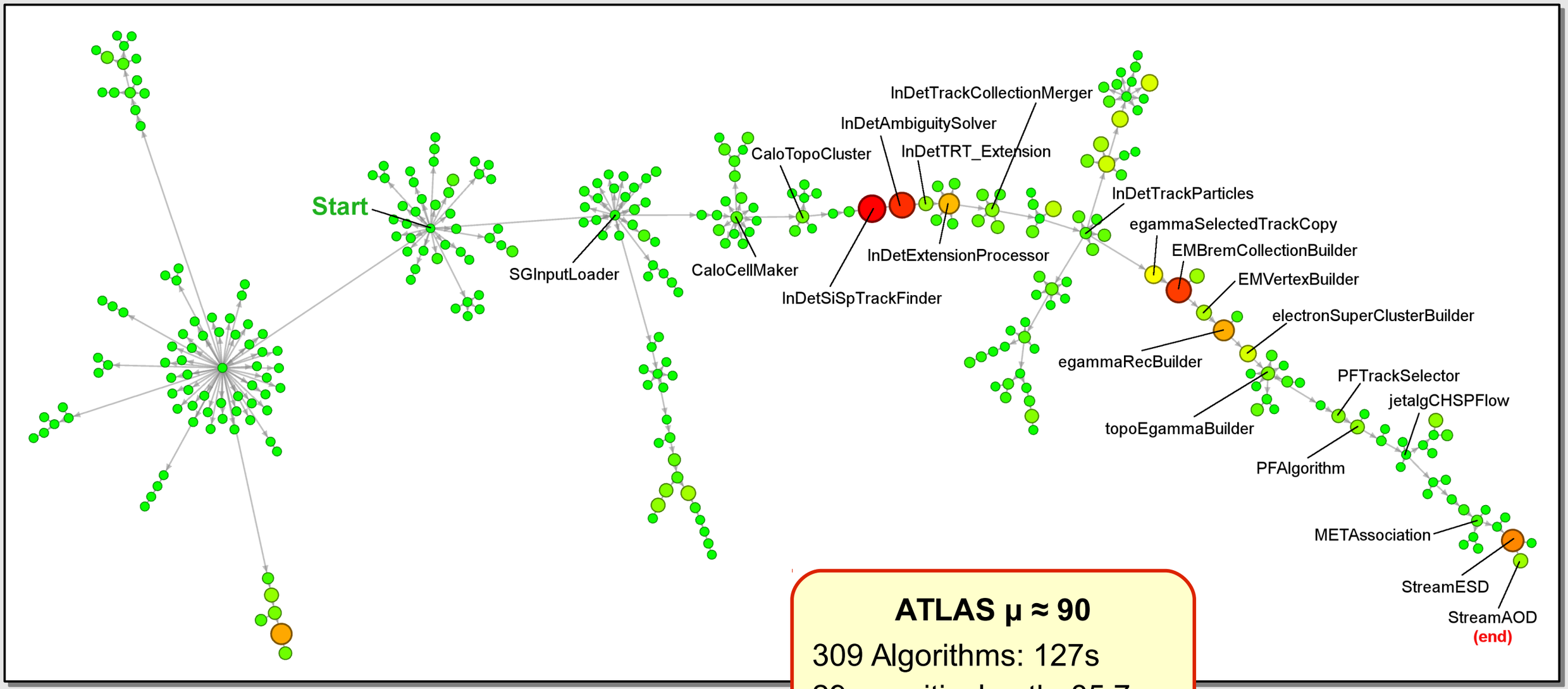


ATLAS

309 Algorithms: 14.6s

17 on critical path: 5.8s

ATLAS High μ Dataset Precedence Trace

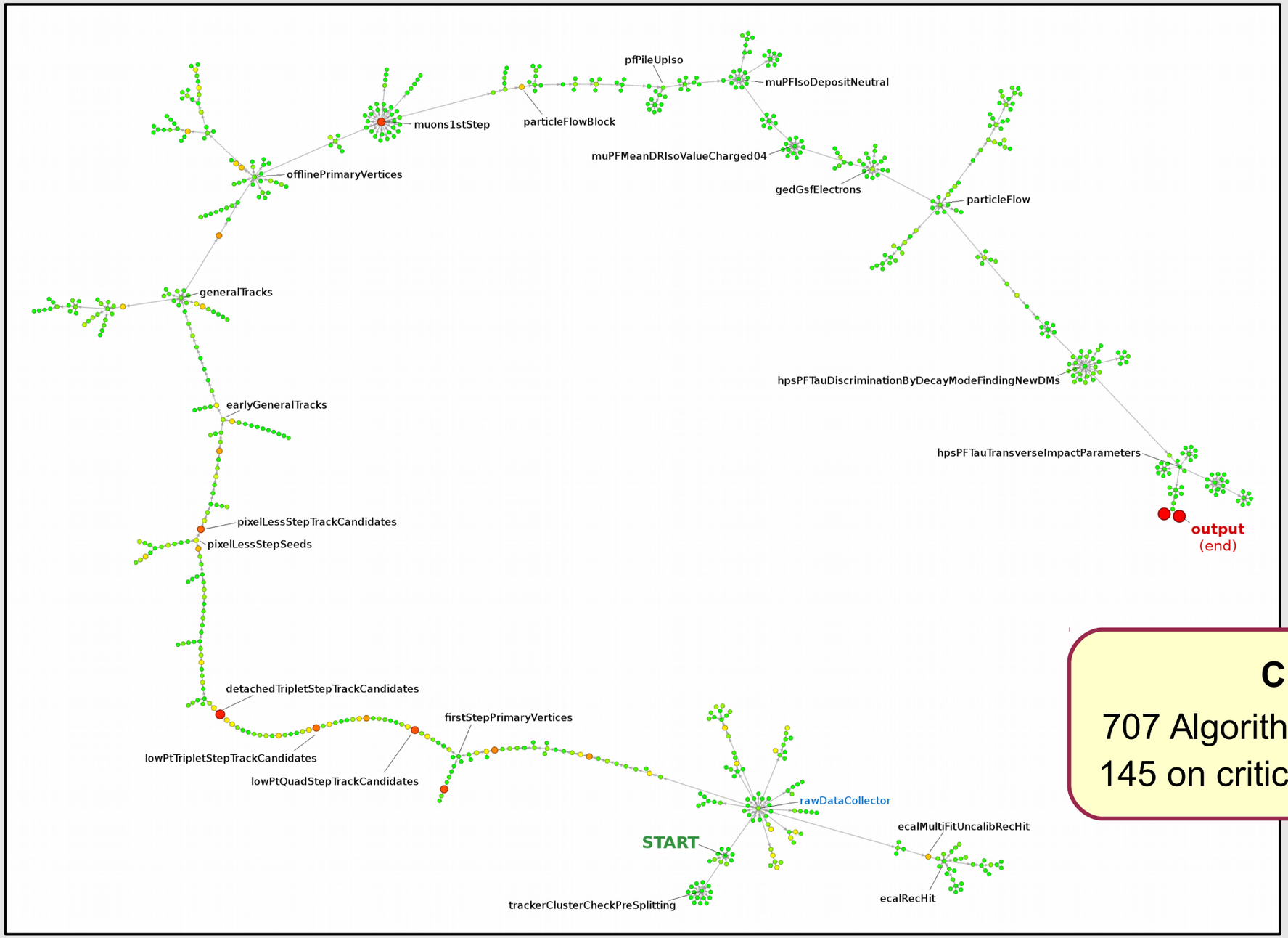


ATLAS μ ≈ 90

309 Algorithms: 127s

29 on critical path: 85.7s

CMS Precedence Trace

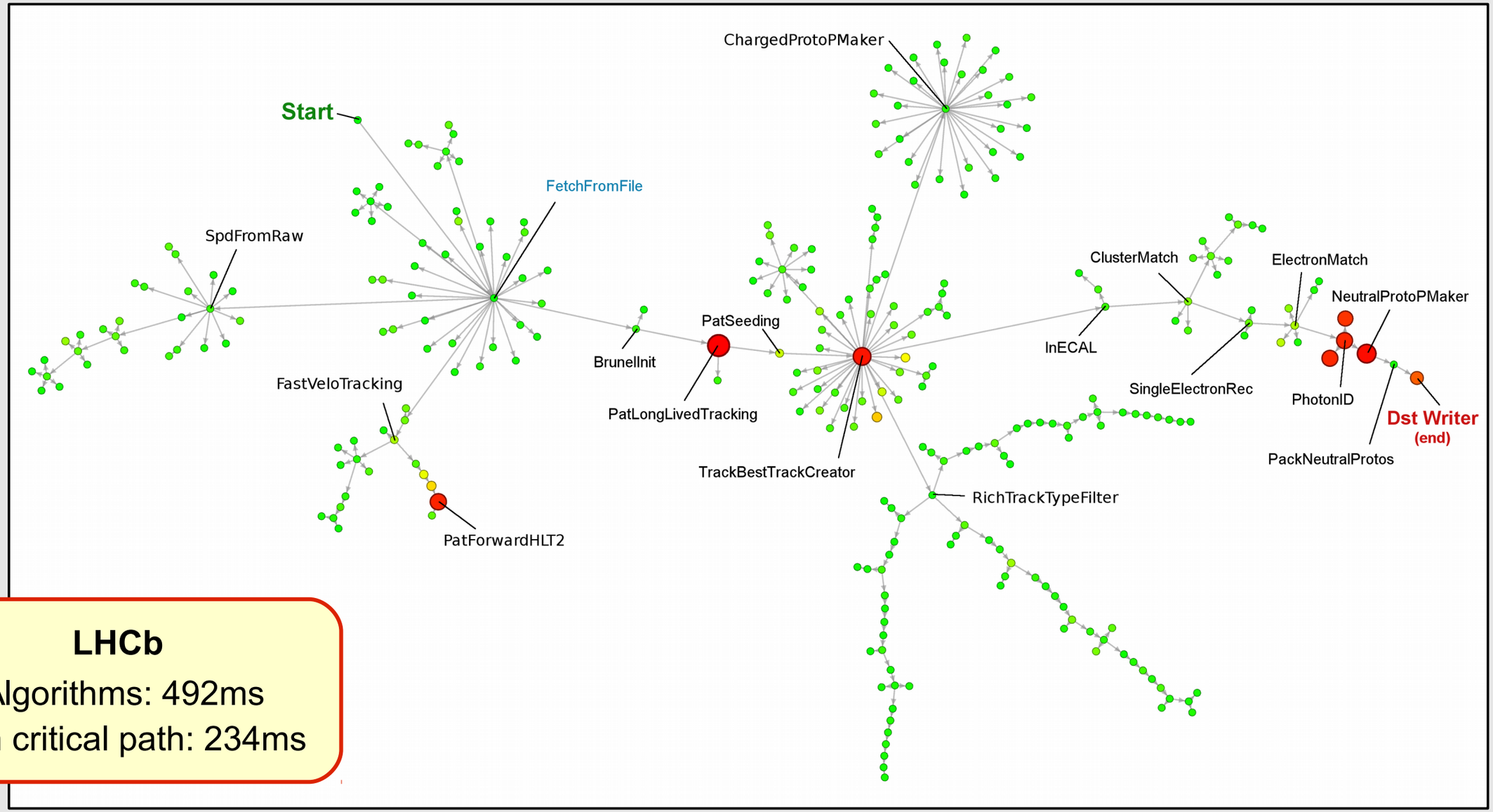


CMS

707 Algorithms: 13.4s

145 on critical path: 7.32s

LHCb Precedence Trace



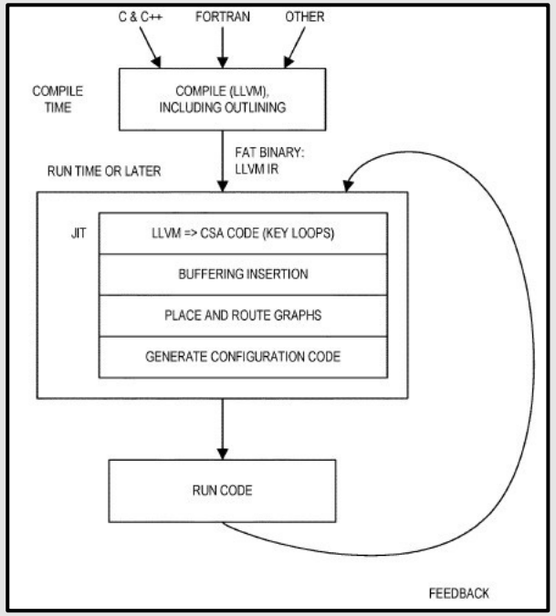
LHCb

282 Algorithms: 492ms

13 on critical path: 234ms

Intel CSA

- The software stack for the CSA as envisioned would be able to take C, C++, and Fortran programs and directly target the CSA
- Prototype has a Clang front end that kicks out LLVM intermediate representation for the regions of code to be accelerated, and then this IR gets compiled down to the CSA itself



- Chip will still have a (low end?) Xeon core to handle operations that don't map well to the CSA
- Parallelism in the code is explicitly grabbed and mapped onto a CSA with a given configuration of processing elements, storage, switching, and such.
  - all kinds of parallelism – instruction, data, pipeline, vector, memory, thread, and task – can all be implemented in some fashion in a CSA device
- Sequential code will require tricky interfaces to memory on the device
  - Intel says the trick is to treat memory access as any other kind of dataflow operation
- 16x density of Fused Multiply-Add units compared to Xeon
- FLOPS performance order of magnitude higher