Experiment Software Frameworks on Heterogeneous Resources

Graeme A Stewart, CERN EP-SFT

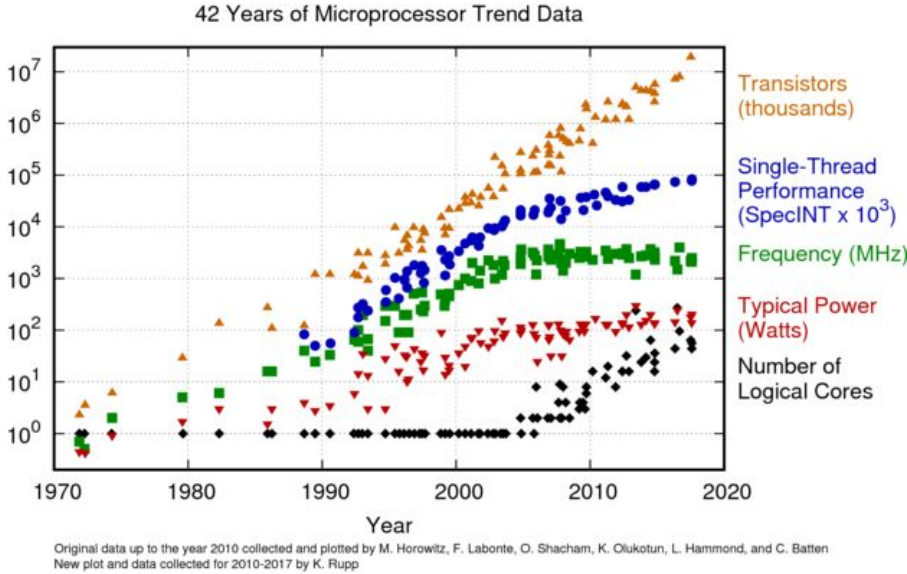
HOW2019 Workshop at JLab, 2019-03-19

Thanks and a Forward Declaration

- Particular thanks to…
  - Charles Leggett, Chris Jones, Mohammad Al-Turany, Giulio Eulisse, Ilya Sharpoval, Dmitry Emeliyanov, Vardan Gyurjyan, Thorsten Kollegger, … and probably others I overlooked
- I will not cover too much here the amount of code the actual codes that we can run on accelerators in HEP
  - For that, please come to this afternoon’s HSF session on Software for Accelerators (in ARC)
  - That session will also expand on quite a few of the topics that I will introduce here

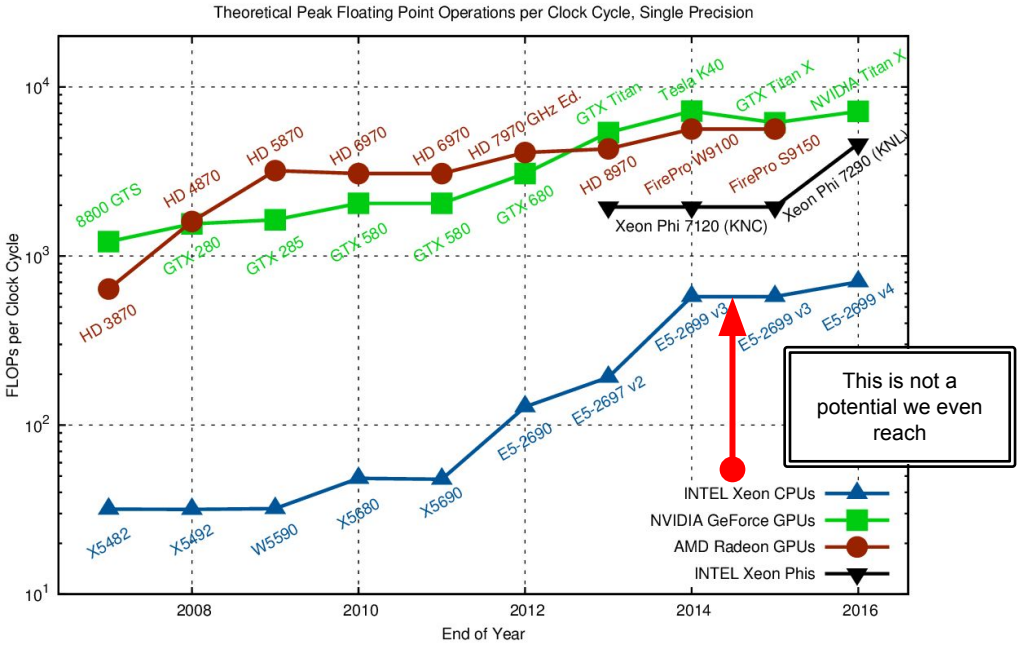
Processor evolution

- As was discussed this morning…
- Moore’s Law slowing significantly
  - Doubling time is lengthening
- Clock speed increases stopped around 2006
  - No longer possible to ramp the clock speed as process size shrinks (Dennard scaling failed)
- So we are basically stuck at ~3GHz clocks from the underlying Wm^{-2} limit
  - This is the *Power Wall*
  - Limits the capabilities of serial processing
  - Push to parallelism and concurrency



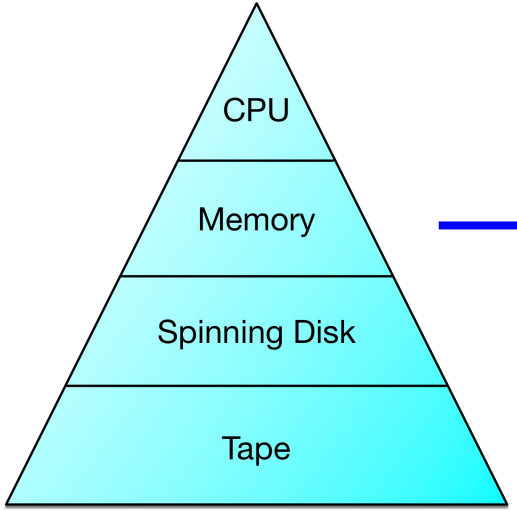
Compute Accelerators

- Most of the CPU die goes to things other than doing maths
  - Even CPU vector registers are hard for us to exploit
- Accelerators have a different model
  - Many cores, high floating point throughput, but lose a lot of ‘ease of use’
- We have to adapt to maintain our ability to use processors effectively

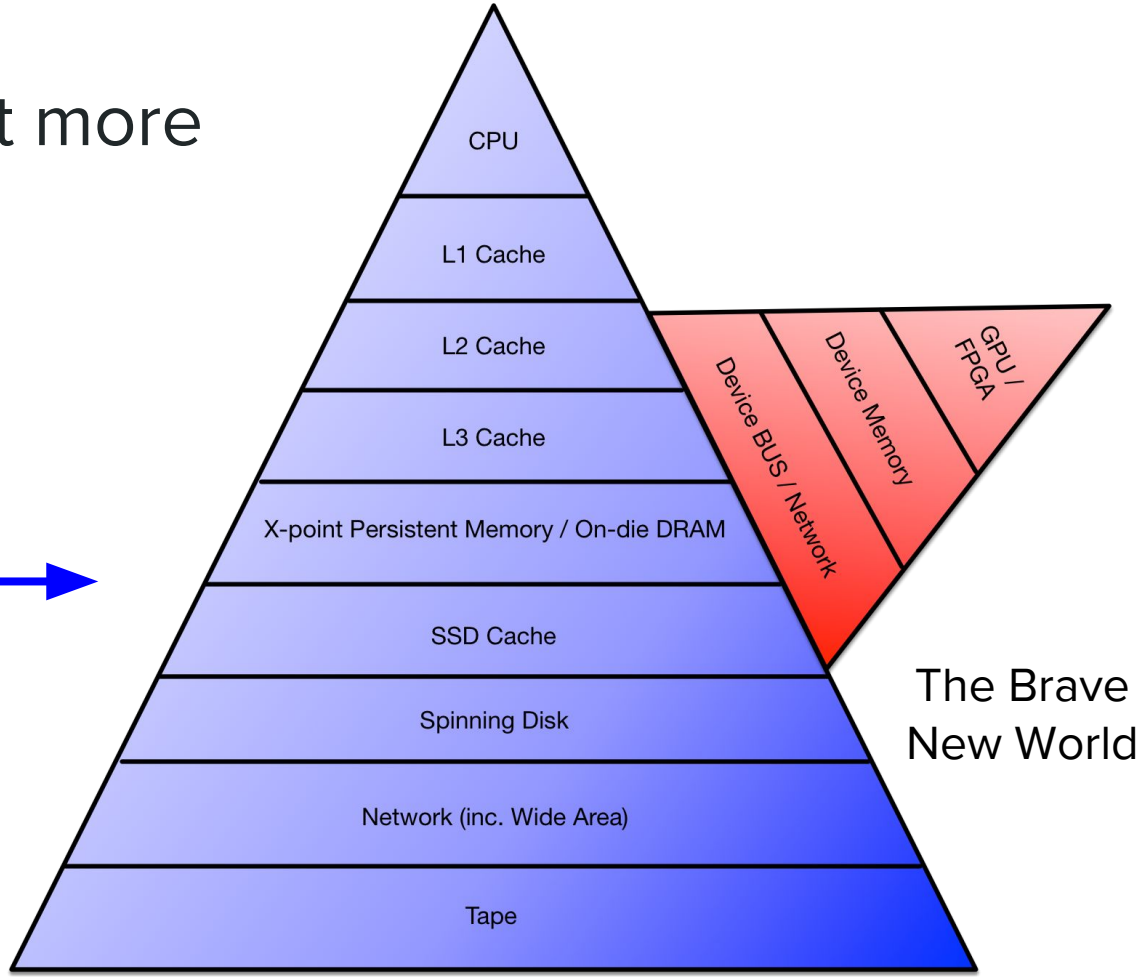


Caveat Emptor - GPUs may use silicon more effectively than CPUs for certain operations, but they are *based on the same fundamental CMOS technology*

So the world got more complicated…

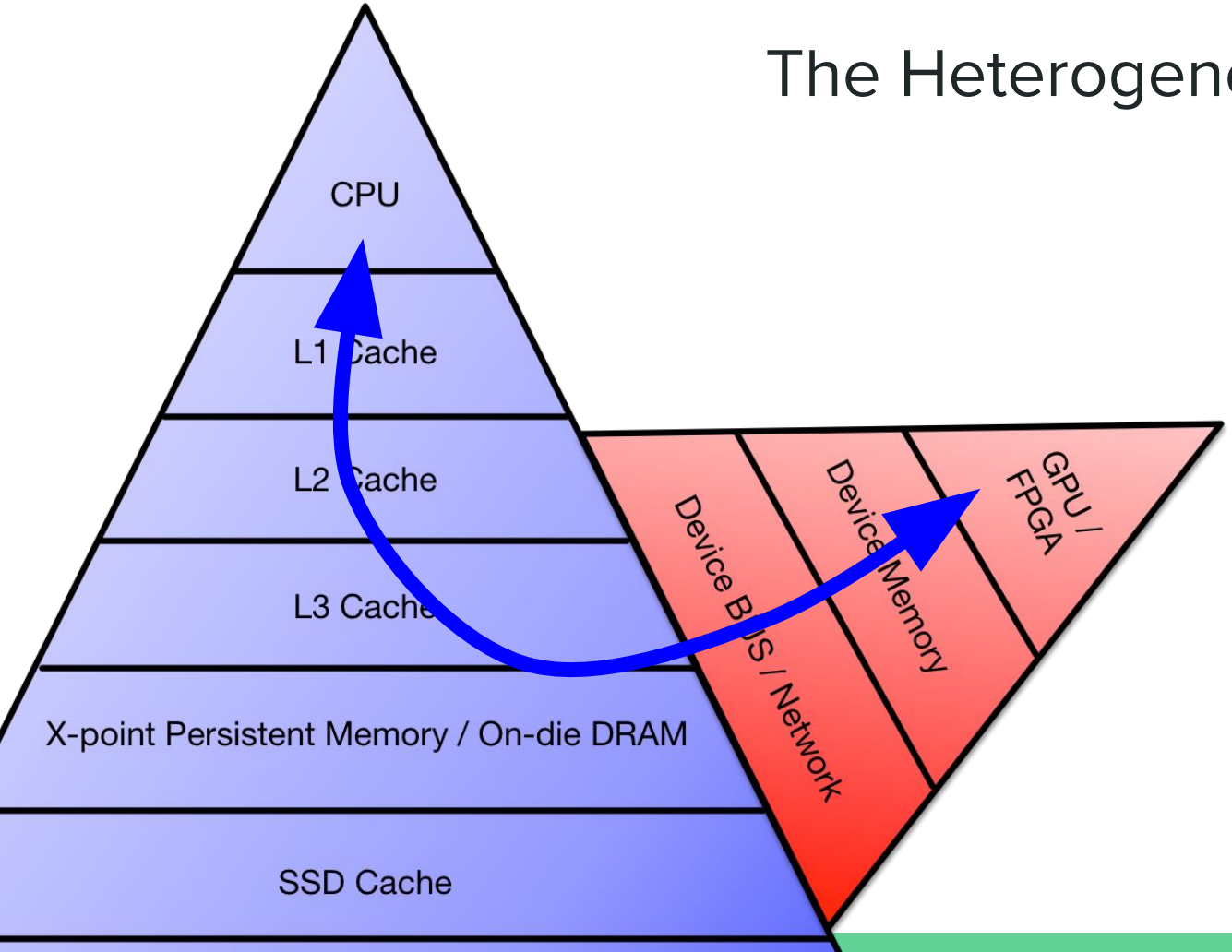


The Good Old Days



The Brave New World

The Heterogenous Corner

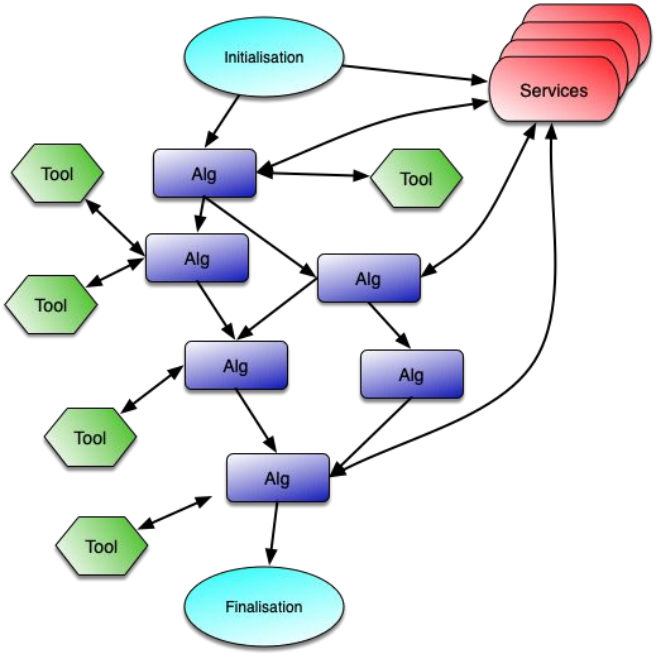


Challenges

- This is an integrated problem
  - We are trying to maximise throughput and minimise the costs of the system
  - Super-optimisation of 1% of the workflow doesn't help
- Desiderata
  - Keep the resources busy
    - Usually meaning don't drop processor cycles
    - Data starvation is the usual problem
      - *Can't to task Y until data X is ready*
      - *I can't yet do Z until condition C is fulfilled…*
      - *Y is ready, but it needs transferred from device M to N*
  - Do useful work
    - It's ok to maximise throughput by doing some extra work (e.g. avoid branches)
    - Work done should be *effective* for attaining the overall goal

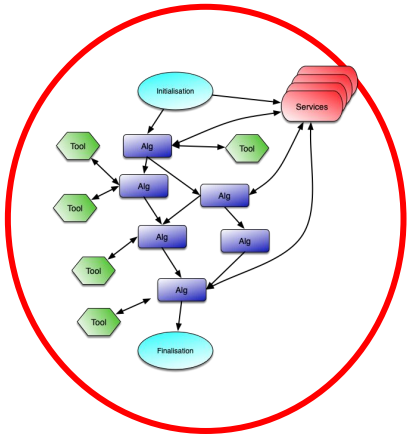
Experiment Software Frameworks

- In a nutshell, these are the applications that manage the processing tasks of the experiment
- The provide a structure into which code that achieves a specific physics goal can fit, e.g.
  - Identify clusters from hit
  - Provide a magnetic field value
- These applications are the payload launched by the experiment production system
- As the comptroller of actual experiment processing, these applications need to marshal resources on the node or nodes where they are running
  - In the past, a single CPUs; now many cores *and* accelerator resources

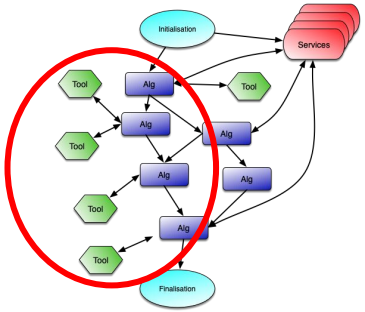


Sketch of traditional HEP software framework. Original versions were all serial. Extant frameworks are/have migrated to add concurrency

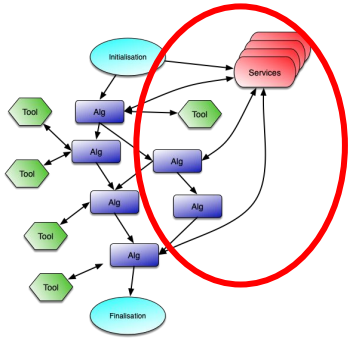
What could you run on the accelerator?



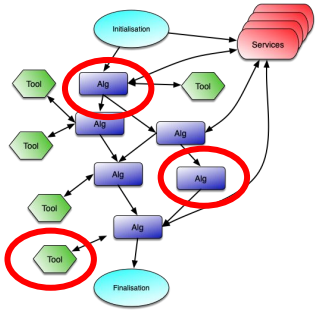
The whole application? Hallelujah!



A substantial chunk? Still pretty good.



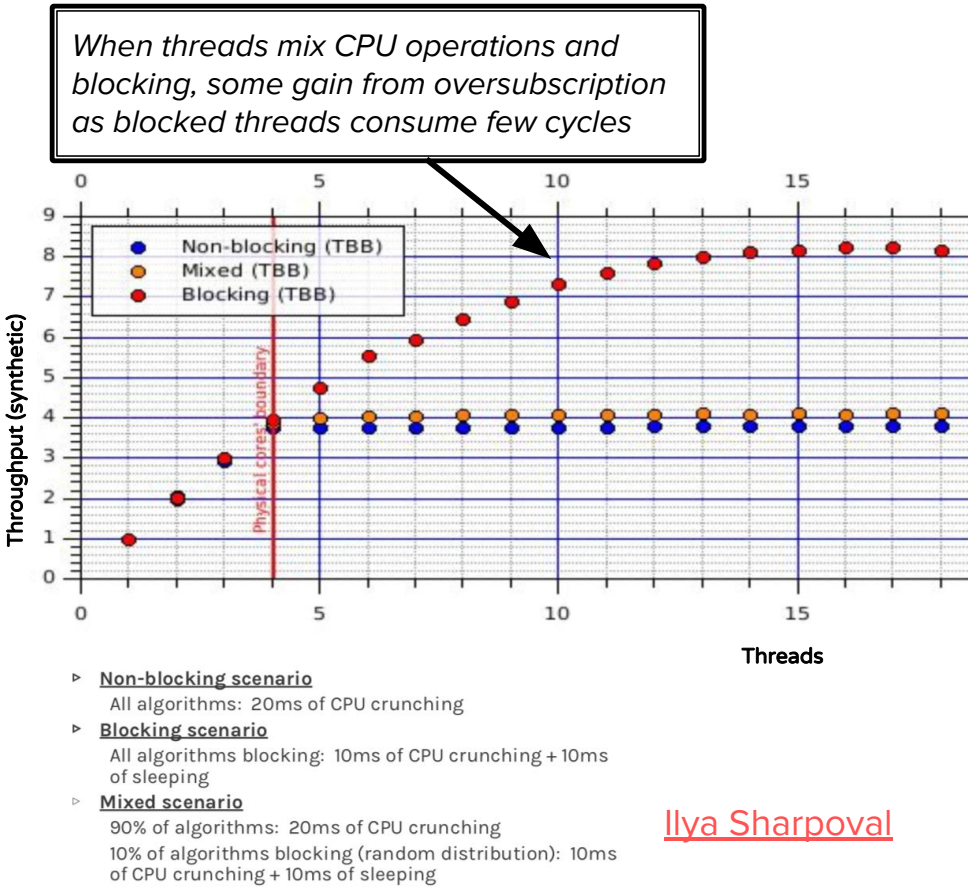
Algorithms that need services? Oh, now I need to port all of that code…



Bits and pieces? That's a lot of internal data movement…

CPU Thread Blocking

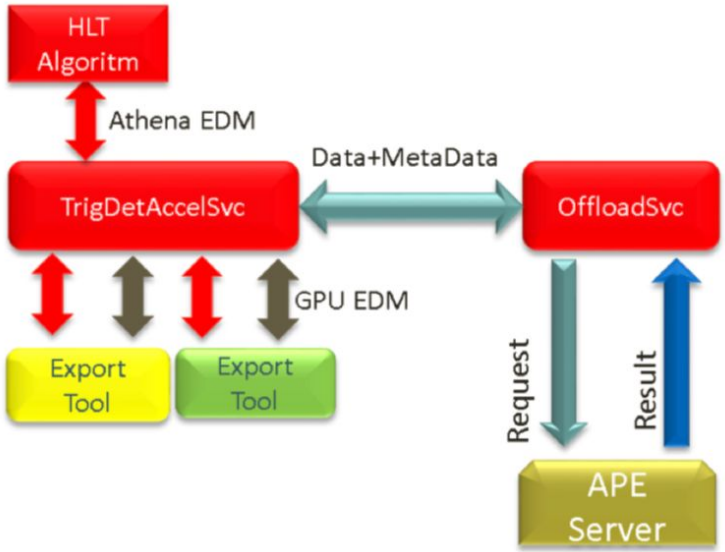
- Intel TBB (used in Gaudi and CMSSW) is designed for running CPU bound applications
  - Blocked TBB threads will pretty much block a core
- Can oversubscribe threads
  - Linux kernel is good at unscheduling blocked threads
  - However, context switches on CPUs are expensive (~3us)
    - Having more CPU bound threads than cores hurts (also consider CPU caches)
  - So better to avoid this



Ilya Sharpoval

Actual offloading in Gaudi/Athena

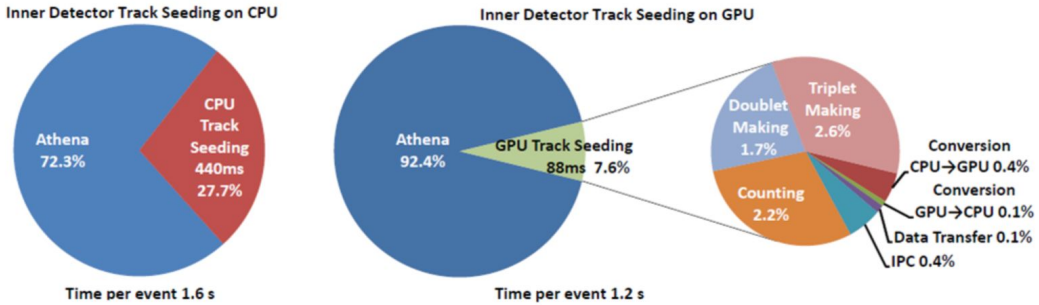
- APE Server developed by Sami Kama
- Algorithm requests offloading
- Data is transformed into GPU EDM
- Athena offload service ships to APE server
  - Allows multiple clients to use the same server
- APE server schedules execution on GPU
- Output data returned to CPU and undergoes EDM transformation
- Algorithm's thread is blocked while waiting for external calculation to complete

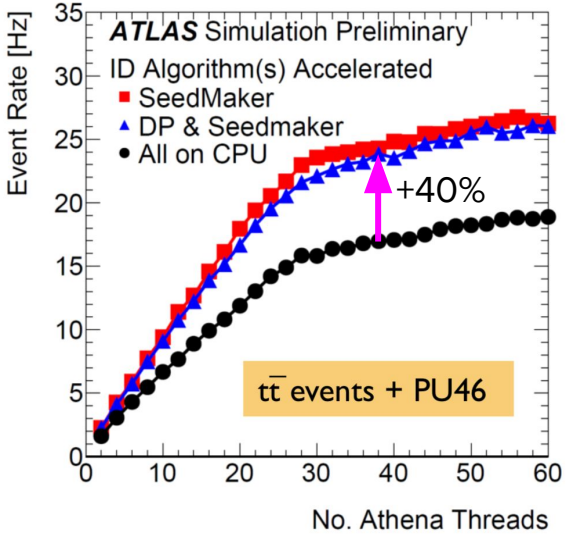


Dependencies:
- YAMPL message passing library
- Intel Threaded Building Blocks

ATLAS HLT Tests

- Conversion of significant part of ATLAS HLT code to GPU
  - Ported code can run significantly faster than on CPU (x5 for single E5-2695 vs. Tesla K80)
  - Overall speed-up limited to x1.4
    - Data transfer/conversion costs
    - Acceleration only applies to part of the workload
  - N.B. GPU resource barely used (1 GPU per 60 CPUs)



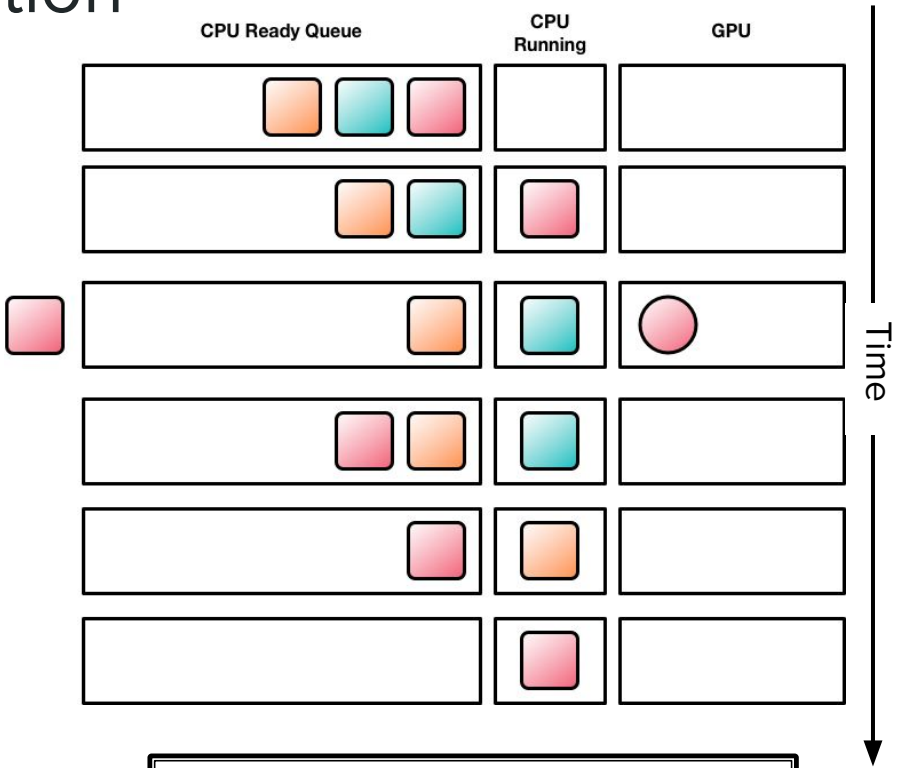


Dmitry Emeliyanov

Chris Jones

CMSSW: Accelerator Integration

- Avoid blocking threads
  - Accelerated module prepares data for accelerator execution
  - Then exits - other waiting tasks can be run
  - Scheduler gets a callback when the accelerator has finished
  - Module then gets popped onto the waiting queue
  - When a free slot is available it can run and pull processed data
- Tests indicate good scalability
  - More details this afternoon in Chris Jones' talk



Note only one CPU thread and 3 modules shown for clarity

CMSSW: Accelerator Integration

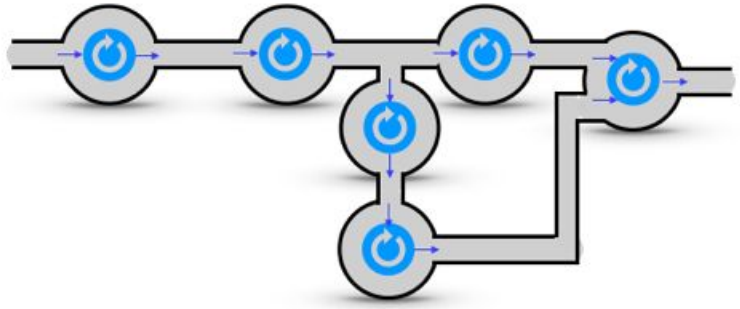
- Can batch events so that module data for multiple events can be sent to the accelerator in one go
  - Too many will eventually lead to many events waiting on the CPU
  - Developers prefer the per-event paradigm
    - [Q. Can we effectively hide batching from them?]
- Code for different architectures can live in different modules
  - Runtime decision on which modules to load when application starts and looks at available resources
  - Provenance is encoded in the output
- Maintaining the code for different architectures is a serious issue
  - What are the best generic mechanisms for writing portable heterogenous code?

Frameworks and Fabric Integration

- Experiment computing has traditionally divided into online and offline sectors
  - Online is close to the experiment, processes data immediately after the DAQ
    - Throughput and latency are both constrained, data selection is usually lossy (events discarded)
  - Offline is physically less constrained to be close to the experiment - Tier0 facilities up to the grid
    - Latency constraints are more relaxed (days to weeks) and system is optimised for throughput; data processing is usually lossless
    - Facilities are less under direct experiment control, traditionally more homogeneous and minimal
- Recent developments are more and more breaking down these barriers
  - Efficient use of resources demands more and more done in software and with a blurring between processing done on ‘raw’ events and steps producing analysis ready data
  - This can go hand in hand with new framework models that reorient the problem more explicitly around data flow
  - Facilities that are under our control are always going to be easier to target

Data Flow and Microservices

- All frameworks are about data transformation
  - Data flow visualises the problem as a graph of processing nodes with data as edges
- Processing nodes do not communicate except via data objects
  - A good paradigm for this conceptual model is to envisage *data as messages*, passed between nodes
  - ⇒ *Message Passing*

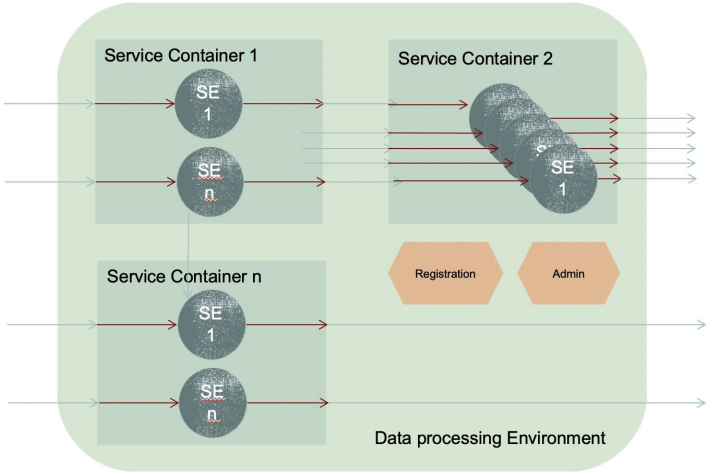


Vardan Gyurjyan

- Processing node independence…
  - Loose coupling (language, dependencies)
  - Flexible deployment
  - ⇒ *Microservices*

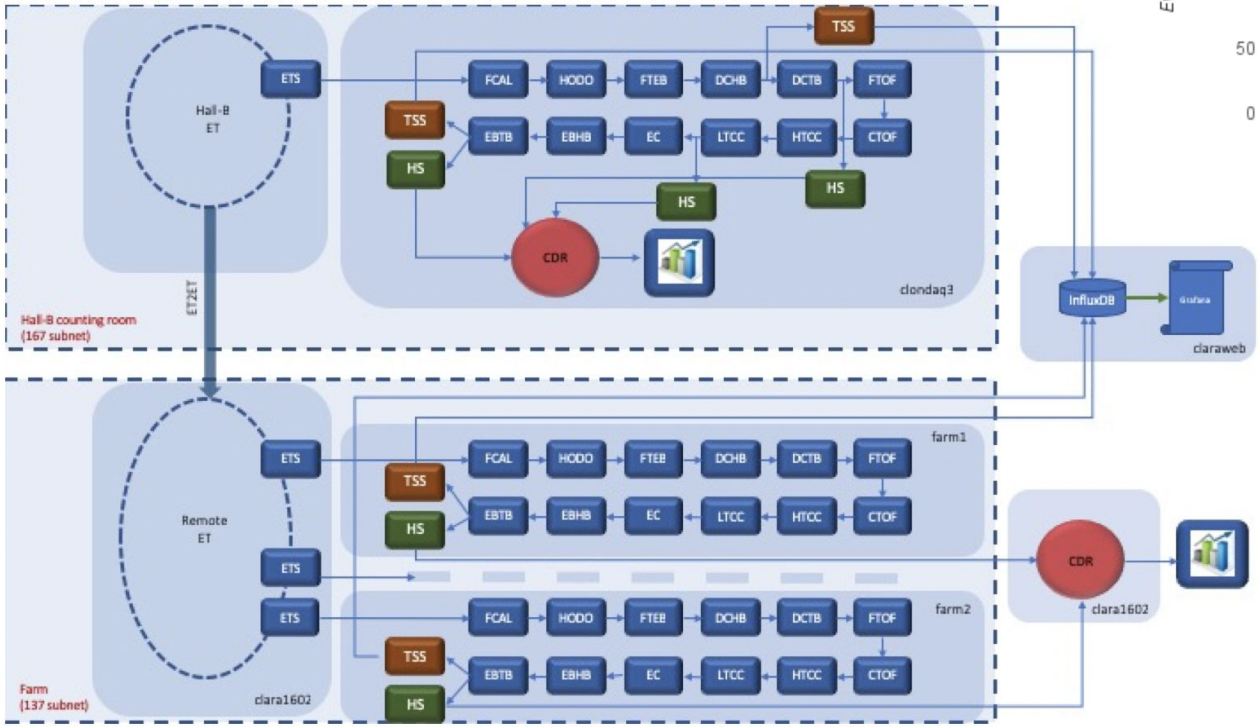
CLARA Framework

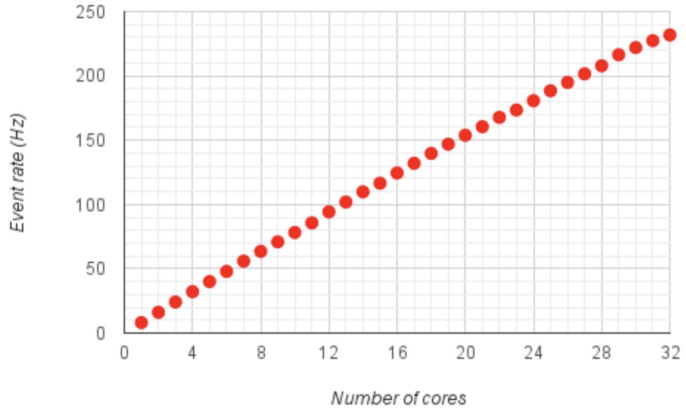
- Implementation of flow based programming with microservices
  - Application is defined as a network of loosely coupled processes
    - Loose coupling of services makes polyglot data access and processing solutions possible (C++, Java, Python, Fortran)
  - Services exchange data by message passing, with connections are specified externally to the services
    - These messages are the ‘data quanta’
  - Services can be requested from different data processing applications



Vardan Gyurjyan

CLAS12 Reconstruction with CLARA





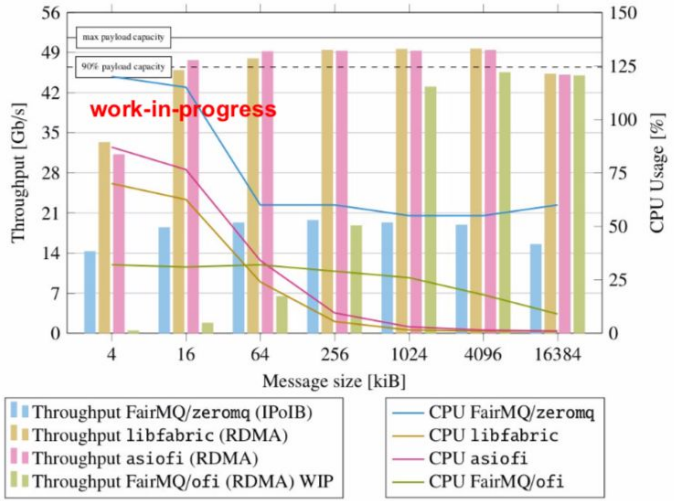
- Scales well
- Handling event reconstruction (online & offline), calibration, analysis
- Also used by NASA
  - AWS deployment
- Accelerator offloading possible

ALFA - ALICE FAIR Software Framework

- ALICE in Run3 and FAIR experiments have very similar challenges
  - Massive data reduction by partial reconstruction
  - Online and offline integration
- Data flow based model
  - Message queues and multi-processing
  - FairMQ transport layer, built on ZeroMQ, shared memory, remote direct memory access (RDMA)
  - Configuration, management and monitoring tools
  - Unified access to configuration and databases
- ALICE quite advanced in the use of GPUs
  - See David Rohr’s talk this afternoon

FairMQ

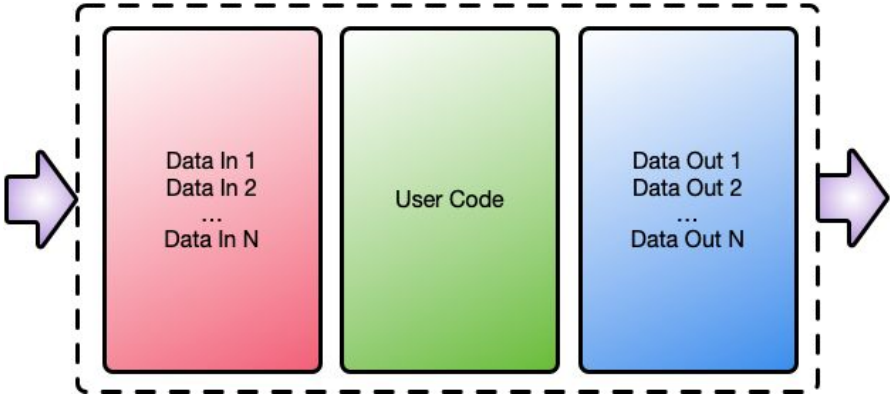
- Message passing efficiency is paramount for performance
  - Can try to hide latency, but the data still needs to flow
- If processing is done on the same node… shared memory is the fastest method to pass data
  - No transfer needed for bulk data (`boost::interprocess`), small additional message to pass ‘metadata’
- Message passing between nodes or devices
  - Use RDMA if possible
    - Pure network layer transfer of data
  - Use ZeroMQ
    - Popular message abstraction layer
- Optimisation of these layers is significant work, but only needs to be done once



Throughput tests for asynchronous RDMA - maximise rate and minimise CPU consumption (Denis Klein)

- ALICE and FAIR experiments benefit from large data chunking
  - E.g. 1000 bunch crossings for ALICE
  - Amortise device movement and easier to load GPU

Devices and Functional Programming



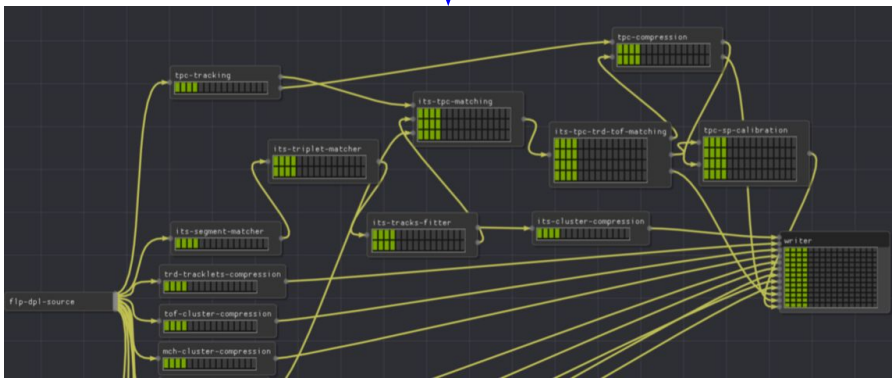
- User code lives in a ‘device’ in ALFA
  - Devices are agnostic to how input data arrives and how output data is consumed
- Model is that developers write functions that deal with data transformations only
- This isolates the developer from needing to worry about many low level details
  - This was always the point of experiment frameworks - isolation is here very robust
  - Remove unnecessary controls and opportunities for mistakes
  - Definition of Global State n. *insanity* (Gerhard Raven)
- Not just a feature of message passing frameworks
  - cf. Gaudi functional approach for physics algorithms
- However, providing services remains a real issue for non-CPU code
  - Geometry, magnetic field, etc. and not trivial

Robustness and Deployment

- Deploying multi-node applications becomes a *complex task*
  - Lots of new failure modes
  - Need an application control layer that deploys components onto a set of heterogeneous resources
  - *Conditions may change during a run*
- E.g., ALFA Dynamic Deployment System
  - Spawn and control hundreds of thousands of different tasks
  - Driven by topology
  - Online clusters or computing clusters or laptops
- CLARA control system manages similarly
  - Gives elasticity in services in response to rate changes
  - Failure recovery helped by isolation

Topology

```
<topology id="myTopology">
   [... Definition of tasks,
properties, and collections ...]
   <main name="main">
      [… Definition of the topology
itself, including groups...]
   </main>
</topology>
```



Outlook

- Experiment frameworks are evolving and being re-invented
- Integration with heterogeneous resources is one of the targets for such developments
  - In tandem with facing many other challenges, e.g., multi-threading and vectorisation
  - Fabrics close to the experiments will play a leading role in testing and deployment
- Even among varying evolutionary and revolutionary approaches there are common points
  - Portable, simple data structures
  - Isolation and functional design
  - Latency hiding
- Clear need for continuing R&D from on frameworks and in algorithms and workloads
  - Should be a focus for collaboration in HSF

Open Questions

- Accelerator hardware is evolving quickly - much faster than CPUs
  - Big gains, but risk of code not working well/at all for the next generation is high
    - cf. experiment lifetimes, so maintenance is a real worry
  - What are the best methods to abstract the code from the specifics of the device?
    - TBB, OpenMP, OpenCL, Vulcan?
    - (Or just accept CUDA and pay later?)
- Coping with heterogeneous sites and maintaining ease of use and efficient use of resources is hard
  - Need to have robust solutions that don't lose everything if a failure occurs
- Efficient processing predicated on low overheads
  - Significant pieces of the workload have to run on non-CPU devices for accelerators to make economic sense
  - Can developers be insulated from event batching?