R&D: HEP computing software

Graeme A Stewart, CERN EP-SFT
Acknowledgement

- Preparing this talk would not have been possible without the many interesting submissions to the EPPSU
- And without the impressive work done by the HEP software community
  - Recent papers at ACAT2019 and HOW2019 were particularly helpful, as well as the HEP Software Foundation CWP Roadmap

Thank you!

Of course, I take responsibility for any mistakes and misunderstandings and it was my choice as to which work, in particular, to highlight
An Overview of HEP Software

- ~50 millions of lines of code, mainly C++
- Significant pieces of software are already shared by most experiments:
  - Event generators, Geant4, ROOT
- Poses the question, can we be doing better?

This is the “traditional” view and how this changes in the future is an important topic for our discussions.
HEP Software Anatomy

- “Task” - do one thing on a significant amount of data
  - Production system breaks tasks down into manageable chunks
- “Job” - one isolated process that runs on a computing node
  - Single or multiple threads, maybe even extending beyond a single device
- “Events” - usually running over each event in turn
  - In most of HEP the discrete event is very meaningful
- “Algorithms/Modules” - data transformation or generation reflecting a single step of a job
  - Data read vs. operations performed varies a lot
Hardware Evolution in a Nutshell

Oh brave new world!
That has such people in it...

CPU
L1 Cache
L2 Cache
L3 Cache
Persistent Memory / On-die DRAM
SSD Cache
Spinning Disk
Network (inc. Wide Area)
Tape

Device BUS / Network
Device Memory
CPU / FPGA

c. 2000

c. 2019
Challenges and Opportunities
Concurrency

- The one overriding characteristic of modern processor hardware is concurrency
  - SIMD - Single Instruction Multiple Data (a.k.a. vectorisation)
    - Doing exactly the same operation on multiple data objects
  - MIMD - Multiple Instruction Multiple Data (a.k.a. multi-threading or multi-processing)
    - Performing different operations on different data objects, but at the same time

- Because of the inherently parallel nature of HEP processing a lot of concurrency can be exploited at rough granularity
  - Run many jobs from the same task in parallel
  - Run different events from the same job in parallel

- However, the push to highly parallel processing (1000s of GPU cores) requires parallel algorithms
  - This often requires completely rethinking problems that had sequential solutions previously, e.g. finding track seeds via cellular automata (TrickTrack library, CMS and FCC)
Heterogeneity

- There are a lot of possible parallel architectures on the market
  - CPUs with multiple cores and wide registers
    - SSE4.2, AVX, AVX2, AVX512, Neon, SVE, Altivec/VMX, VSX
  - GPUs with many cores; FPGAs
    - Nvidia (many generations - often significantly different), AMD, Intel
- In addition there are ‘far out’ architectures proposed, like Intel’s Configurable Spatial Architecture
- Many options for coding, both generic and specific:
  - Cuda, TBB, OpenACC, OpenMP, OpenCL (→ Vulcan), alpaka, Kokkos, ...
- Frustratingly no clear winner, mutually exclusive solutions and many niches
  - One option for now is to isolate the algorithmic code from a ‘wrapper’ that targets a particular device or architecture - approach of ALICE for their GPU/CPU code
  - Hiding details in a lower level library (e.g. VecCore) also helps insulate developers
Data Layout and Throughput

- Original HEP C++ Event Data Models were heavily inspired by the Object Oriented paradigm
  - Deep levels of inheritance
  - Access to data through various indirections
  - Scattered objects in memory
- Lacklustre performance was ~hidden by the CPU and we survived LHC start
- In-memory data layout has been improved since then (e.g. ATLAS xAOD)
  - But still hard for the compiler to really figure out what’s going on
  - Function calls non-optimal
  - Extensive use of ‘internal’ EDMs in particular areas, e.g. tracking
- iLCSoft / LCIO also proved that common data models help a lot with common software development
- Want to be flexible re. device transfers and offer different persistency options
  - e.g. ALICE Run3 EDM for message passing and the code generation approaches in FCC-hh PODIO EDM generator
Machine Learning

- Machine learning, or artificial intelligence, used for many years in HEP
  - Algorithms learn by example (training) how to perform tasks instead of being programmed
- Significant advances in the last years in ‘deep learning’
  - Deep means many neural network layers
  - Fast differentiability and use of GPUs
- Rapid development driven by industry
  - Vibrant ecosystem of tools and techniques
  - *Highly optimised for modern, specialised hardware*

#53, 79, 162, 5, 150, 126, 16, 34, 43, 127

ML minimisation problem - do this minimisation with $10^6$ variables...

An example of a modern ML architecture
Machine Learning in HEP

- Better discrimination
  - Important input for analysis (see improvements with Higgs)
  - Also used at HLT as inference can be fast (N.B. training can be slow!)
  - HEP analogies to image recognition or text processing
- Replace expensive calculations with trained output
  - E.g. calorimeter simulations and other complex physical processes
- There are significant opportunities here
  - Need to combine physics and data science knowledge
  - Field evolves rapidly and we need to deepen our expertise
- Integration into our workflows is not at all settled
  - Resource provision, efficient use, heterogeneity and programming models pose problems

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Years of data collection</th>
<th>Sensitivity without machine learning</th>
<th>Sensitivity with machine learning</th>
<th>Ratio of $P$ values</th>
<th>Additional data required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS$^{54}$ $H \rightarrow \gamma \gamma$</td>
<td>2011–2012</td>
<td>2.2$\sigma$, $P = 0.014$</td>
<td>2.7$\sigma$, $P = 0.0035$</td>
<td>4.0</td>
<td>51%</td>
</tr>
<tr>
<td>ATLAS$^{43}$ $H \rightarrow \tau^+ \tau^-$</td>
<td>2011–2012</td>
<td>2.5$\sigma$, $P = 0.0062$</td>
<td>3.4$\sigma$, $P = 0.00034$</td>
<td>18</td>
<td>85%</td>
</tr>
<tr>
<td>ATLAS$^{99}$ $VH \rightarrow bb$</td>
<td>2011–2012</td>
<td>1.9$\sigma$, $P = 0.029$</td>
<td>2.5$\sigma$, $P = 0.0062$</td>
<td>4.7</td>
<td>73%</td>
</tr>
<tr>
<td>ATLAS$^{41}$ $VH \rightarrow bb$</td>
<td>2015–2016</td>
<td>2.8$\sigma$, $P = 0.0026$</td>
<td>3.0$\sigma$, $P = 0.00135$</td>
<td>1.9</td>
<td>15%</td>
</tr>
<tr>
<td>CMS$^{100}$ $VH \rightarrow bb$</td>
<td>2011–2012</td>
<td>1.4$\sigma$, $P = 0.081$</td>
<td>2.1$\sigma$, $P = 0.018$</td>
<td>4.5</td>
<td>125%</td>
</tr>
</tbody>
</table>

Machine learning at the energy and intensity frontiers of particle physics,

[https://doi.org/10.1038/s41586-018-0361-2](https://doi.org/10.1038/s41586-018-0361-2)

Use of Generative Adversarial Networks to simulate calorimeter showers, trained on G4 events (S. Vallacorsa)

#53, 79, 162, 5, 150, 126, 16, 34, 43, 127
Far Future Ideas: Quantum Computing, Neuromorphic...

- Intensely active area of research
  - Europe have invested 1B€ in Quantum Flagship Program; US invest heavily as well (including for HEP)
- Certainly a game changer if engineering of sufficient, stable q-bits can be achieved
  - Rapid progress in the last 5 years, but still far from being practical and useful
  - Even with some spectacular breakthroughs commercialisation would take time
- Maria’s talk gave some specific projects
- How should HEP be involved? And at what level?
  - Are these with extra resources or some effort that we dedicate from our pool?
  - Mapping QC to current HEP algorithms? New algorithms enabled by QC? Programmable? Maintainable?
HEP R&D
Event Generation

- Starting the simulated events chain from theory
  - Previously was very small part of LHC computing budget (cf. detector simulation), no pressure to optimise

- Increasing use of higher precision to drive down errors (NLO, NNLO, ...) - negative weights become a serious problem
  - Greatly increases the CPU budget fraction given over to event generation
  - Possibility of sharing matrix element calculations between experiments being explored (new HSF WG coordinating)

- Theory community not rewarded for providing generators to experiments
  - Not experts and lack incentives to adapt to modern CPU architectures

- From the technical point of view, these codes are a good target for optimisation
  - A lot of pure maths, floating point intensive
  - No inputs, small outputs
  - Ideal for HPC environments
  - Some parts of the code has been ported to GPU as well (MadGraph)
  - Can we find ways to collaborate with software engineers?
Simulation

● A major consumer of LHC grid resources today
  ○ Experiments with higher data rates will need to more simulation
● At the same time flat budget scenarios don’t give a lot more cycles
  ○ So need faster simulation
● Technical improvement programme helps (and helps everyone)
  ○ GeantV R&D modernises code and introduces vectorisation; serious studies of GPU porting are starting (US Exascale Computing Project)
● But this will probably not be sufficient to meet future needs
  ○ Will need to trade off accuracy for speed with approximate and hybrid simulation approaches
    ■ Combine full particle transport with faster techniques for non-core pieces of the event
● Machine learning techniques are gaining ground, but yet to be really proven
  ○ Need to decide when they are good enough cf. Geant4
  ○ Integrating these into the lifecycle of simulation software and developing toolkits for training and inference is needed - this is a software and a computing problem
Reconstruction and Software Triggers

- Hardware triggers no longer sufficient for modern experiments (LHCb, ALICE)
  - More and more initial reconstruction needs to happen in software
- Close to the machine, need to deal with tremendous rates and get sufficient discrimination
  - Pressure to break with legacy code is high
  - Lots of experimentation with rewriting code for GPUs
    - In production for ALICE (since Run2)
    - NA62 a non-LHC example
    - Advanced prototypes for CMS (Patatrack) and LHCb (Allen)
- Orienting the design around the data (optimal layouts) is critical
- Bulk data and exploit concurrency
- Be as asynchronous as possible
- Transfers between host and device are expensive
  - Port blocks of algorithms, even ones where gain is small
- Even the physics performance can improve when revisiting code!
Reconstruction and Software Triggers

- **Real Time Analysis (HEP Version)**
  - Design a system that can produce analysis useful outputs as part of the trigger decision
    - If this captures the most useful information from the event, can dispense with raw information
  - *This is a way to fit more physics into the budget*

- **Challenges**
  - Have to convince physicists this works
  - Buffer for raw events is limited (‘real-time’ decisions)
  - Calibration needed for final output needs to be \( \sim \) fast
    - Two reco passes used in ALICE and LHCb
  - Validation is very important
  - Selectively storing information requires a lot of physics inputs

LHCb Run2 Turbo took 25% of events for only 10% of bandwidth

LHCb charm physics analysis using Turbo Stream (arXiv:1510.01707)
Analysis

- Scaling for analysis level data also a huge challenge
- Efficient use of analysis data can come with combining many analyses as carriages in a train like model (pioneered by ALICE)
  - Also goes well with techniques like tape carousels
- Reducing volume of data needed helps hugely
  - CMS ~1kB nanoAOD makes a vast difference to analysis efficiency and “papers per petabyte”
- Improve analysis ergonomics - how the user interacts
  - Declarative models (ROOT’s RDataFrame)
    - Say what, not how and let the backend optimise
  - Notebook like interfaces gain ground, as do containers
  - Cluster power, laptop convenience - analysis clusters
- Interest in data science tools and machine learning is significant for this community - inspiring new approaches (e.g. Coffea)
  - This is an ecosystem into which HEP can contribute
Frameworks and Integration

- Increasingly heterogeneous world requires advanced software support infrastructure
  - Software frameworks support use of different devices as well as insulate developers from many of the details of concurrency and threading models
    - Latency hiding is critical to maintaining throughout
  - Framework development has traditionally been quite fragmented, but new experiments should offer a chance to increase convergence
    - Better to start off together than try to re-converge later (iLCSoft, LArSoft examples of success, albeit without concurrency)

- Actually software integration, into a working stack, is very desirable (‘Turnkey Stack’)
  - Integrate common components (geometry, simulation, reconstruction toolkits)
  - Saves time in conceptualisation and performance studies
    - Projects like AIDA/AIDA2020 have done this rather well
Training and Careers

- Many new skills are needed for today’s software developers and users
- Base has relatively uniform demands
  - Any common components help us
- LHCb StarterKit initiative taken up by several experiments, sharing training material
  - Links to ‘Carpentries’ being remade (US training projects)
- New areas of challenge
  - Concurrency, accelerators, data science
  - Need to foster new C++ expertise (unlikely to be replaced soon as our core language, but needs to be modernised)
- Careers area for HEP software experts is an area of great concern
  - Need a functioning career path that retains skills and rewards passing them on
  - Recognition that software is a key part of HEP now
Organising for the Future

- **HSF**
  - Overarching umbrella organisation, at the international level (strongest in Europe and North America)
  - Builds community efforts, very inclusive; defined the [Community White Paper Roadmap](#).

- **Software Institutes**
  - IRIS-HEP in US
    - NSF funded at US$25M over 5 years
    - Machine Learning, DOMA, Innovative Advanced Algorithms, Analysis
  - Should Europe do more here?
    - Traditionally labs (CERN, DESY) have played this role, but time to break out beyond HEP?
    - A lot of shared problems - critical architecture changes, new techniques affect us all
      - Value of the institute is in breaking boundaries (experiment, region, science)
    - Linking to *academic experts in software engineering* could be mutually very beneficial
    - Also helps us to tackle the training problem (pass on skills) and careers (better defined path) and solve practical software problems
Summary

- The landscape has shifted significantly in the last decade
  - Concurrency, Accelerators, Heterogeneity, Data Layout, ...
- We are constantly adapting and evolving our legacy software
  - Challenges are not just current experiments, but R&D for future detectors
- Adopting a more radical approach involves committing **a lot of effort**
  - It really pays off - *improved software improves our physics*
- We understand the main engineering issues, but not at all problems solved
  - How best to factorise from the specific technologies to avoid lock-in?
- Pyramid of skills and expertise
  - Need a lot of software engineering and physics talent
  - Address training needs
  - Long term career prospects for HEP software experts need to improve
- Huge opportunities for software to improve *that we have to grasp*
  - Organise around this goal and reach out to industry, software engineers, other sciences
Backup
Optimal Experiment Software - The Golden Roles

- Orienting the design around the data (optimal layouts) is critical
- Bulk data together and exploit concurrency where ever possible
- Be as asynchronous as possible
  - Framework should hide latency
- Transfers between host and device are expensive
  - Port blocks of algorithms, even ones where gain is small
- The physics performance can improve when revisiting code!
  - We have a lot of legacy; revisiting the code oriented to the primary goal simplifies and improves maintainability
Summary of EPPSU Inputs re. Software

● The EPPSU inputs that made mention of software are summarised here:

https://docs.google.com/spreadsheets/d/1mjN6AaSUUFY-r_HxkKvV4E4f2cgPkEaLcHEFIHm0LxA/edit?usp=sharing