Patatrack!

Compute Accelerator Forum – March 9th 2022

Andrea Bocci¹, Eric Cano¹, Angela Czirkos¹, Antonio Di Pilato²,
Gabrielle Hugo¹, Vincenzo Innocente³ Matti Kortelainen³, Martin Kwok³,
Felice Pantaleo¹, Wahid Redjeb¹,⁴ Marco Rovere¹

with contributions from

Taylor Childers⁵, Alexei Strelchenko³, Yunsong Wang⁶

¹ CERN, ² CASUS, ³ FNAL, ⁴ RWTH, ⁵ ANL, ⁶ LBL
Patatrack?
what is… Patatrack?

- loosely defined group of people working on R&D and the use of accelerators in CMS
  - based at CERN, Fermilab, CASUS, Aachen, …
  - overlap with HEP-CCE
- incubator of ideas and R&D efforts
  - algorithms and implementations
  - performance portability
  - development of standalone demonstrators
  - solutions and adoption in CMS and CMSSW
- promote the use of GPUs and accelerators in CMS
  - Patatrack Hackathons, documentation and knowledge transfer
  - collaboration with CASUS
- CMS pixel-only track and vertex reconstruction
  - developed for use at HLT
  - targeting both Phase-1 (ready) and Phase-2 (under development)
10th Patatrack Hackathon

- we had a very successful hackathon last November! (despite the covid restrictions)

- main topics: Alpaka, MPI, performance, …
Patatrack pixel-only track reconstruction
Pixel tracks and vertices workflow

- The overall approach
  - Work on individual events (no batching)
  - Reconstruct pixel-based tracks and vertices on the GPU
  - Leverage existing support for threading and on-demand reconstruction
  - Minimise data transfer

- The full workflow
  - Copy the raw data to the GPU (~250 kB/event)
  - Run multiple kernels to perform the various steps:
    - Decode the raw data
    - Build clusters from the pixel hits
    - Form hit doublets
    - Build ntuplets (triplets, quadruplets, …) with a Cellular automaton algorithm
    - Clean up duplicates with the Fishbone algorithm
  - Take advantage of the GPU computing power to improve the physics
    - Fit the track parameters (Riemann fit, broken line fit) and apply quality cuts
    - Reconstruct vertices
  - Copy only the final results back to the host (optimised SoA format)
    - Convert to legacy format if requested

Copy to host

Conversion to legacy formats

Two alternative fit algorithms
different parallelisation strategies

- raw data unpacking and decoding
  - parallelised across all input pixel hits
- clustering of the pixel hits
  - parallelised across the pixel detectors and across the input pixel hits
- conversion to global coordinates
  - parallelised across each cluster
- building doublets
  - parallelised on the hits of each layer
- building ntuplets
  - 2D parallelisation on the inner and outer layers
  - Cellular Automaton algorithm with depth-first search
- ntuplets cleaning
  - Fishbone algorithm merges overlapping ntuplets
  - 2D parallelisation over ntuplets and possible duplicates
- track fitting
  - two alternatives: Riemann fit, Broken Line fit
  - implemented using Eigen, parallelised over the ntuplets
- vertex reconstruction
  - applied to quadruplets above a given threshold
  - cluster tracks based on their position along the Z axes
  - parallelised across all input tracks
  - split low quality vertices
  - parallelised across the vertices
physics performance: pattern recognition
physics performance: resolution
for more information


Patatrack R&D and performance portability
all the results in the following slides should be considered preliminary

we have used the bleeding edge, latest and greatest version of the code to prepare the most up-to-date results
  • which means we have not validated them across different machines, software versions, etc.

please do not re-share them or used them as the basis for any assumptions
  • get in touch with us if you would like to!
R&D and performance portability

- Patatrack and HEP-CCE’s pixeltrack-standalone project
  - https://github.com/cms-patatrack/pixeltrack-standalone/
- simplified version of the CMSSW framework
  - full support for plugins, multithreading (based on Intel TBB), and asynchronous work
  - minimal external dependencies
- extract the CMS reconstruction algorithms from CMSSW into a self-contained project
  - pixel local reconstruction
  - Patatrack pixel tracks and vertices
- includes a minimal validation of the results of the reconstruction
- useful for prototyping different solutions
- simpler to port to different hardware and software architectures
R&D and performance portability

- with a standalone application is much easier to …
- … try different implementations of the accelerators support in the framework
  - call-backs ? blocking wait ? polling ?
- … prototype different data structures
  - user friendly SoA abstractions
- … port to different backends than the whole CMSSW application
  - AMD HIP / ROCm
  - Intel oneAPI (still in progress)
- … test different performance portability solutions
  - Kokkos, Alpaka
• compare the performance of the reconstruction on different GPUs
  • NVIDIA Tesla V100 (GV100)
    – Volta architecture
    – 5120 threads, 32 GB RAM, 300 W
  • NVIDIA Tesla T4 (TU104)
    – Turing architecture
    – 2560 threads, 16 GB RAM, 70 W
  • NVIDIA A10 (GA102)
    – Ampere architecture
    – 9216 threads*, 24 GB RAM, 150 W
    – * 4608 fp32 + 4608 int32/fp32 threads

• dual Intel Xeon Silver 4114
  • 2 x 10 cores, 2 x 20 threads
example: caching memory allocation

- GPUs memory allocations are expensive
  - device memory (cudaMalloc)
  - pinned host memory (cudaMallocHost)
- stream-ordered memory operations achieve better performance
  - available from CUDA 11.2
  - avoid global synchronisations
- even better, cache all memory allocations
  - based on the CUB caching allocator
  - extended to pinned host memory
  - does not require special support in the driver or the runtime

Patatrack Preliminary

Performance vs Number ofThreads

- Throughput (events/s)
- Number of Threads / Streams

Drop due to contention in the caching allocators

Native CUDA - NVIDIA A10
- Caching Allocator
- No Caching, Stream Ordered Allocations
- No Caching, No Stream Ordered Allocations
performance portability results

Patatrack Preliminary

all backends implement a caching memory allocator

• alpaka achieves ~ 95% of the native performance for the CUDA and CPU backends
  • we are investigating what causes the 5% drop
CMS choice of a *performance portability* solution for Run-3

- lengthy R&D and comparison of the features and limitations of Kokkos, Alpaka, oneAPI
  - joint effort with HEP-CCE
  - based on these and similar results
- SYCL / oneAPI deemed not mature enough
  - especially on NVIDIA hardware
- decided to adopt Alpaka as the solution for *performance portability* in Run-3

main use cases: run the High Level Trigger software originally developed for CUDA

- on different GPUs (i.e. AMD)
- automatically port to run on CPUs (replace in-house effort)
conclusions
conclusions

- the Patatrack effort has been the catalyst for the adoption of GPUs in the CMS reconstruction
  - CMS is deploying a GPU-equipped HLT farm from the beginnig of Run-3
    - currently being commissioned at Point-5, based on dual AMD EPYC “Milan” 7763 cpus and dual NVIDIA T4 GPUs
  - all CUDA-based algorithms are fully integrated in CMSSW
    - portability to run on cpu-only machines achieved with ad hoc solutions
  - choice of Alpaka for performance portability in CMSSW in Run-3
    - likely too late for the deployment at HLT this year, target the MC production campaigns later in the year and deployment at HLT in 2023

- the R&D landscape is more open than ever
  - integration of “portable” modules and data structures in CMSSW
  - development of new algorithms and porting of existing ones to GPUs
    - physics reconstruction for Phase-1 and Phase-2: calorimeters, tracking, particle flow, ...
    - integration with ML frameworks, efficiently sharing access to a GPU
  - extension of the functionalities provided by Alpaka
    - support CUDA and ROCm in the same binary
    - refactor CMS optimisation into a stand-alone libraries
  - use of GPUs on remote machines over efficient network fabrics
questions ?
more slides!
clustering the pixel hits

- the algorithm starts from the “unpacked” ADC readout
- each cluster will produce in parallel:
  - local position and errors
  - beamspot-subtracted global position

step 1: each pixel in parallel compares its index with the neighboring pixels.

step 2: the minimum between the indices spreads, overwriting all the others.

step 3: the seed gets the value of a global counter and decreases it; each seed will hence get a different and consecutive negative number whose absolute value is the cluster index in the pixel module.
pixel doublets

- the CMS Phase-1 Pixel detector includes 10 pixel layers
  - 4 barrel
  - 3+3 end-caps
- a layer can have more than one adjacent layer
  - 13 combinations of layer pairs
- pairs of hits on nearby layers are linked together to form doublets
- doublets are created in a window depending on the tracking region, beamspot, and layer-pair
  - can include a requirement on the compatibility of the cluster size along the beamline between the two hits (only in the barrel)
- parallelism is expressed over hits and doublets
  - doublets are created for each pair of layers, in parallel at the same time
Cellular Automaton-based ntuplet builder

The CA is a track seeding algorithm designed for parallel architectures. It requires a list of layers and their pairings:

- A graph of all the possible connections between layers is created.
- Fast computation of the compatibility between two connected doublets, in parallel.
- No knowledge of the world outside adjacent neighboring doublets required, making it easy and efficient to parallelize.
- Parallelization strategy: one thread per doublet.

A parallel Depth-first search is then used to build ntuplets:

- Quaruplets (ntuplets with 4 or more hits) are used for high purity and as input to the pixel building step.
- Triplets (ntuplets with 3 or more hits) are used to recover a higher efficiency at the cost of a large fake rate.
duplicate cleaning: Fishbone

- the same particle can produce more than one hit on the same layer due to module overlaps
  - for redundancy and hermeticity
- after building ntuplets with the CA algorithm, “fishbone” seeds can be produced to account for module/layer overlaps
- select the best ntuplet candidate and filter out the duplicates
Broken Line fit

A four-hits fit allows to better exploit information coming from the new pixel detector and improve parameter resolutions and fake rejection [BLFit]

- The Broken Line takes into account by design the kinks due to multiple scattering.

Fitting procedure:

- Fast circle fit: estimate of p for multiple scattering, estimate of the radius/center
- Circle fit: d0, pT, \(\phi\)
- Line fit: dz, cot(\(\theta\))

Implemented using Eigen matrix library, parallelizing using a CUDA thread per track.

- Allows perfect code portability between CPU and GPU implementation and bitwise-matching of the results.

Final cleaning:

- If tracks share a doublet, keep the track with the best \(\chi^2\)
- Track selection: pT>0.3 GeV, |d0|<0.5 cm, |z0|<12 cm
Pixel Vertices reconstruction

fitted pixel tracks are used to reconstruct pixel vertices

- consider only tracks with 4 or more hits, and pT > 0.5 GeV/c
- cluster tracks based on the z coordinate of the point of closest approach to the beam line
- calculate local linear density for each track

tracks with a local density above threshold are promoted to vertex seeds

- tracks are linked to seeds based on their distance and $\chi^2$

vertex seeds are promoted to vertex candidates if they contain at least 2 tracks

- candidates with good $\chi^2$ become final vertices
- candidates with bad $\chi^2$ are split into multiple vertices
• physics performance:
  • 20000 MC ttbar events
  • \( <\text{PU}> = 50 \), design conditions, 25ns, \( \sqrt{s} = 13 \text{ TeV} \)
  • matching of reconstructed tracks with simulated ones requires that all hits of the reconstructed track come from the same simulated track

• reconstruction efficiency:
  • number of reconstructed tracks matched to simulated tracks, divided by the number of simulated tracks
  • computed only with respect to the hard scatter
  • implicit cut: \(|d0| < 3.5 \text{ cm}\)

• fake rate:
  • number of reconstructed tracks not matched to any simulated tracks, divided by number of reconstructed tracks

• duplicate track
  • a reconstructed track is considered a "duplicate" it is matched to a simulated track that itself is matched to \( \geq 2 \) reconstructed tracks

• vertex merge rate:
  • the probability of having 2 different simulated vertices reconstructed as a single vertex
Performance portability results

**Patatrack Preliminary**

Performance vs Number of Threads

![Graph showing performance results](image-url)