

Tracking in LHCb, lessons learned

GDR-InF annual workshop

Renato Quagliani (LPNHE)



October 8, 2020



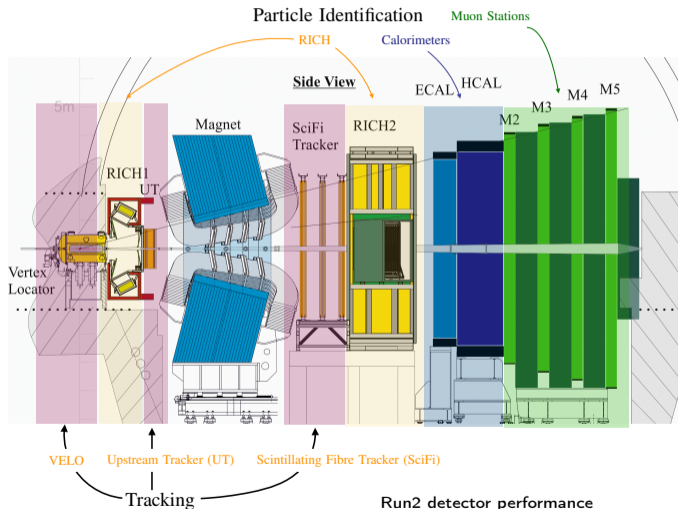
Overview

- LHCb detector and its upgrade for Run III
- Tracking in LHCb (for Run III): reinventing the wheel
- Computing aspects related to tracking
- The need for speed: q/p parameterisation(s)
- Performances of HLT1 reconstruction at collision rate
- Conclusion

The LHCb detector

LHCb is a high precision experiment at LHC optimized for b and c hadrons decays

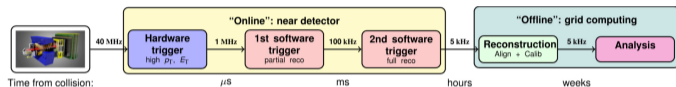
- Forward arm spectrometer in $\eta \in [2, 5]$
- Excellent track and vertex reconstruction
 - 1 $\sigma_{IP} \sim 20 \mu\text{m}$ ($p_T > 2 \text{ GeV}/c$)
 - 2 $\epsilon_{\text{tracking}} > 96\%$
 - 3 $\sigma_p/p \sim 0.5 - 1\%$
 - 4 $\sigma_\tau \sim 45 \text{ fs}$ for b hadrons.
- Excellent particle identification
 - 1 $\epsilon_{K-ID} \sim 95\%$
 - 2 $\epsilon_{\mu-ID} \sim 97\%$
- Benefit of large $b\bar{b}$ and $c\bar{c}$ cross section in pp collision in forward region.



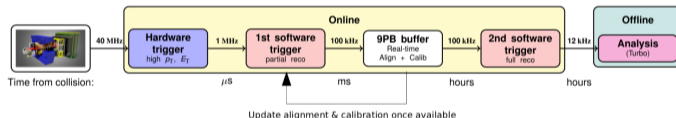
Run2 detector performance
 Int. J. Mod. Phys. A30 (2015)
 1530022

LHCb DAQ and trigger in Run1-2-3 : a continuous evolution

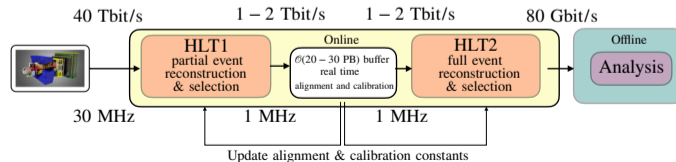
• Run 1 (2011-2012):



• Run 2 (2015-2018):



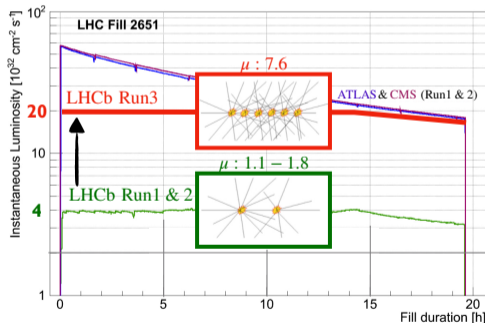
• Run 3 (2021-2025++):



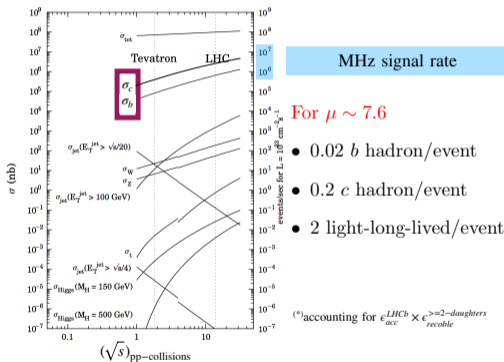
- **Hardware trigger**: 40 → 1 MHz read-out limit in **Run1,2** based on Muon and Calorimeter signatures
- **HLT1**(partial) and **HLT2**(full) event reconstruction split in **Run2**
- **Buffer** data to disk to perform real time alignment and calibration
- Offline quality reconstruction and selection in the online system
- **Run3** : remove **Hardware trigger** in favour of a fully software based one.
- Event reconstruction at collision rate
- Full detector read-out at 40 MHz

From Run 1,2 to Run3: b, c physics at LHC

- Run 3 data taking period planned to start in 2021
- LHC pp collisions at $\sqrt{s} = 14$ TeV, 25 ns bunch spacing \rightarrow 40 MHz collision rate.
- LHCb aims at boosting the physics output increasing the instantaneous luminosity and the signal rate.

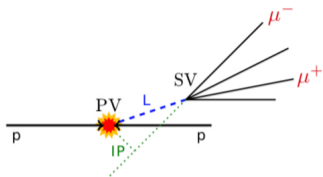


- More PVs, more tracks, more signal
- Almost all events will have a b or c hadron in Run 3

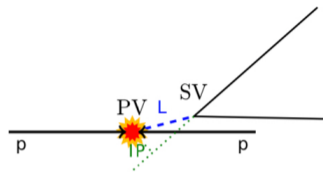


LHCb-PUB-2014-027

Signatures in LHCb from b and c hadrons for triggering



- $m_{head} \sim 5.28 \text{ GeV} \rightarrow p_T^{daughters} \sim \mathcal{O}(\text{GeV})$
- $\tau_B \sim 1.16 \text{ ps}$. $\Delta(SV - PV) \sim 1 \text{ cm}$.
- Dispaced tracks carrying high p_T .



- $m_{head} \sim 1.86 \text{ GeV} \rightarrow p_T^{daughters} \sim \mathcal{O}(\text{GeV})$
- $\tau_B \sim 0.4 \text{ ps}$. $\Delta(SV - PV) \sim 0.4 \text{ cm}$.
- Dispaced tracks carrying high p_T .

Key ingredients for efficient triggering and signal discrimination

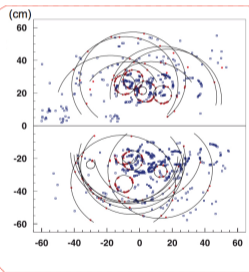
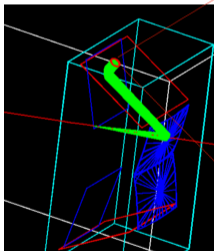
- Primary vertex finding, high p_T tracks reconstruction and optimal μ -Identification
- Inclusive triggers on 1&2 track signatures.
- **Challenge in Run3** is not only to have an efficient trigger, but also be able to identify the topology of events as early as possible in the triggering process: more information than single sub-detector read-out needed
- \rightarrow Track reconstruction at collision rate required : **huge computing** challenge

Tracking at LHCb

Importance of tracking at LHCb

- Tracking is **the bridge between detector readout and physics analysis**
- Determine p, p_T of particles, crucial for PID and Primary Vertex (PV) reconstruction as well.

$$m = \frac{p}{c\beta\gamma} \quad \begin{array}{l} \swarrow \text{from tracking} \\ \downarrow \text{Ring Radius} \end{array} \quad \cos\theta_C = \frac{1}{n\beta}$$



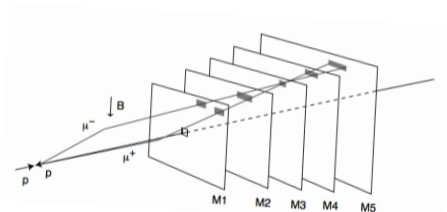
Tracking and PID (RICH)

- RICH detectors based on cherenkov radiation
- Ring center from track projections into RICH detector
- Radius of ring measured

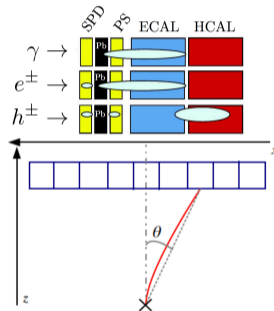
Importance of tracking at LHCb

- **The bridge between detector readout and physics analysis** creating particles
- Determine p, p_T of particles, crucial for PID and Primary Vertex (PV) reconstruction as well.

Muon Station



Calorimeters

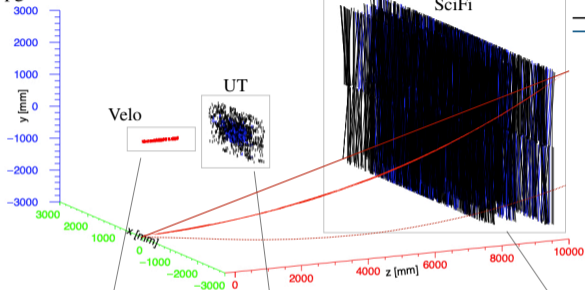


Tracking and PID (Muon&Calo)

- Fire all muon stations in Field of Interest regions: it's a μ^\pm .
- Calorimeter clusters matching track projection : e^\pm/h^\pm ID combined to RICH.

Tracking system in LHCb: what measurements do we combine?

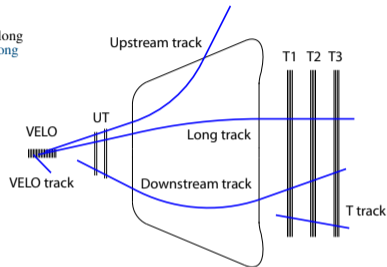
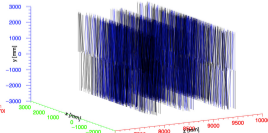
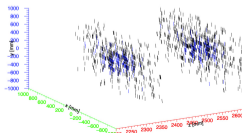
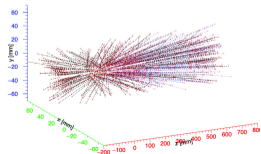
1 LHCb Upgrade event



Direct (x, y, z) measurements

(x, [y_{min}, y_{max}], z) measurements

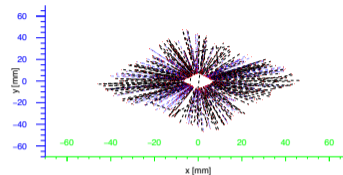
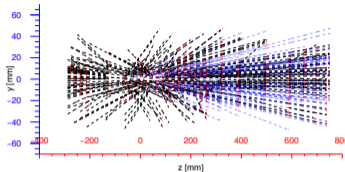
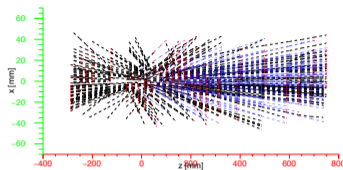
(x, [0, ± 2.7m], z) measurements



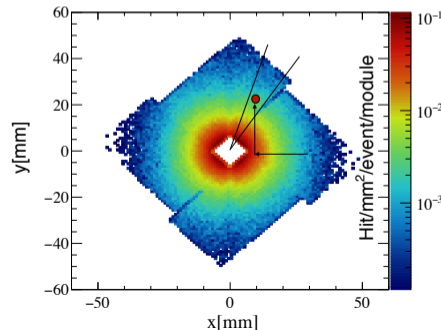
Track types for physics analysis

- Velo: from collision point.
- Long: from decays and from PVs.
- Downstream: from long-lived particle decays (no Velo segment)

Velo track reconstruction

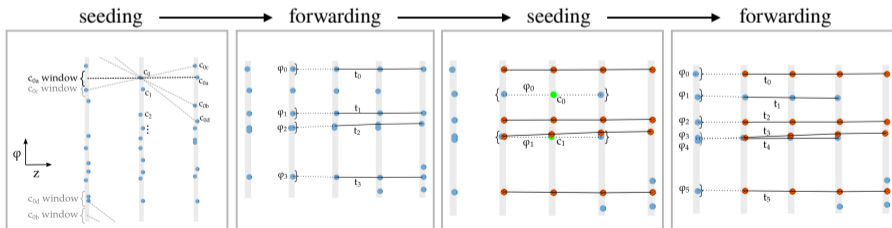
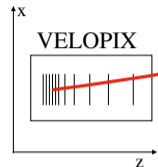


- Find all Velo tracks to reconstruct PVs.
- 26 modules providing (x, y, z) with $\sigma_{x,y} = 5/9\mu\text{m}$
- No \vec{B} field in Velo region
- Only multiple scattering leads to *tiny* bending
- Tracks are almost fully contained in small ϕ windows.
- Maximise spatial locality in memory for pattern recognition sorting hits by ϕ .
- Boost timing with no physics losses in searching of hits with ϕ sorted hits container.

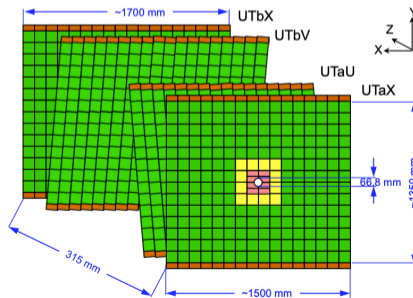
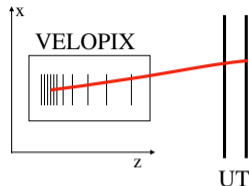
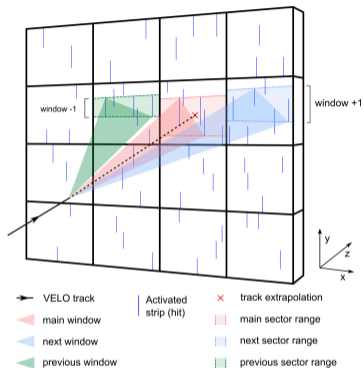


Velo track reconstruction: implementation

- Search for combinations of hits in parallel given 3 input modules
- *Seeding* : Iterate over all possible triplets of VELO modules
- Choice of triplets based on alignment in ϕ and 3 hit 3D-alignment
- *Forwarding*: Forward triplet to next layer.
- Algorithm interleaves seeding with forwarding to maximize spatial and temporal locality.



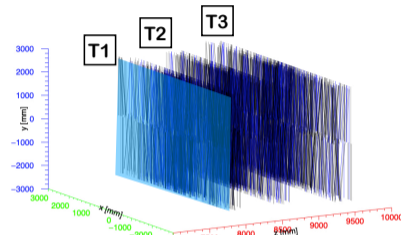
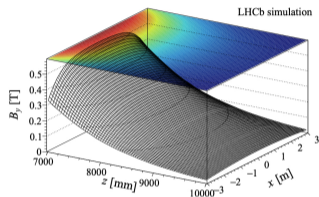
VELO-UT tracking



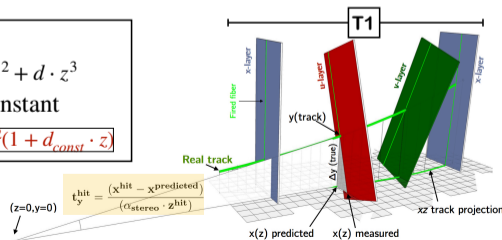
- Find hits in the UT tracker (4 layers) matching the Velo input tracks projections after *small* magnetic field bending.
- Define search regions in each UT plane: hits are stored in sector ranges and optimized for parallel processing.
- Tracklets finding inside windows from the 4 layers building combinatorics in parallel.

Forward tracking: Local SciFi fit model

- Tracks keep bending inside z_{SciFi}
- Fit model simplified to 3 degrees of freedom with the B field constraints applied.
- More bending from $T_1 \rightarrow T_2$ than $T_2 \rightarrow T_3$
- Becomes critical when $\sigma_x \sim 100\mu\text{m}$ in any (x, y) SciFi region.

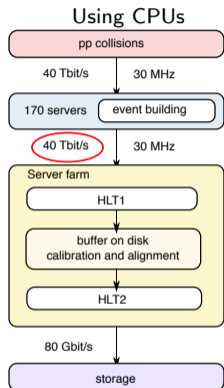


$$\begin{aligned} \sim B_y^{SciFi} &= B_0 + B_1 \times z \\ \rightarrow x(z) &= a + b \cdot z + c \cdot z^2 + d \cdot z^3 \\ B_1(x, y)/B_0(x, y) &\sim \text{Constant} \\ \rightarrow x(z) &= a + b \cdot z + c \cdot z^2(1 + d_{const} \cdot z) \end{aligned}$$

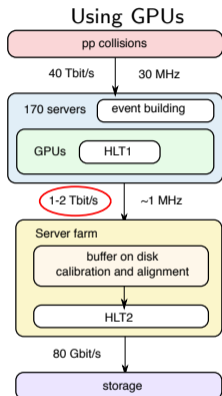


Computing aspects

Reconstruction at collision rate for the LHCb upgrade: 2 TDRs



Trigger TDR (2014)

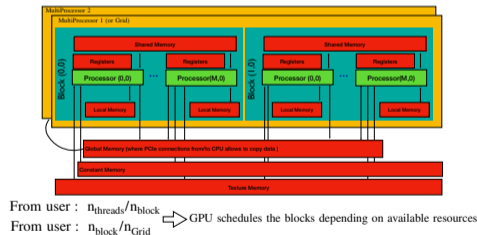
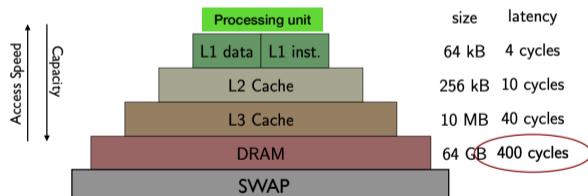


GPU HLT TDR (2020)
Allen project

- Both proposals carried out in the last years
- Extensive studies and developments on both architectures
- Brand new algorithms and ideas on pattern recognition developed on both architectures
- **Final decision : use GPUs for HLT1**
- All the work and experience gained for HLT1 reconstruction using CPUs crucial to achieve large speed-up also for the HLT2 reconstruction.
- Benefit of running HLT1 on GPUs :
 - 1 Reduce network bandwidth between EventBuilder and filter farms
 - 2 Free up filter farm CPUs for HLT2 only

Heterogeneous computing for Event reconstruction in LHCb

- Different way of using memory between CPU and GPU: the closer to processing unit the memory used is, the faster the processing.
- Parallel and fast programming with CPU requires the programmer to force data structures to fit in caches and avoid different threads to modify shared objects.
- Parallel and fast programming with GPU is easier for programmers since memory handling is fully defined and handled by the user.



Heterogeneous computing for Event reconstruction in LHCb

- Memory layout of data is crucial to achieve fast memory access and it depends on algorithm implementation.

```
//Array Of Structure layout
struct Point{
    float x, y, z;
}
std::array<Point,1024> data; //create 1024 (x,y,z) positions
for( int i= 0; i< 32; ++i){ //access 32 of them
    if( data[i].x <0 || //access x in memory
        data[i].y <0 || //access y in memory
        data[i].z < 0){ //access z in memory
        data[i].x =0;
        data[i].y =0;
        data[i].z =0;
    }
}
```

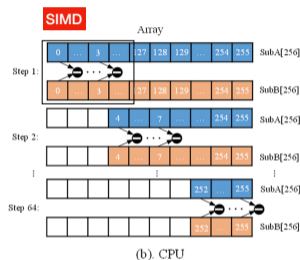
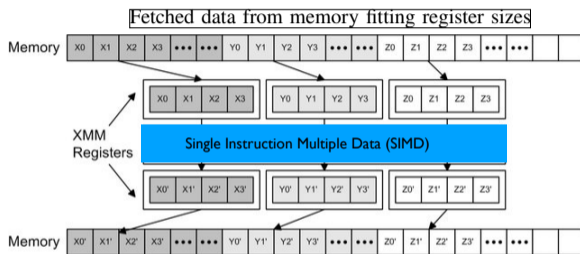
AOS layout

```
//Structures Of Array layout
struct SOAPoint{
    std::array<float,1024> x,y,z;
}
SOAPoint data; //1024 (x,y,z) positions
for( int i= 0; i< 32; ++i){ //access 32 of them
    if( data.x[i] <0 || //access x[i] in memory
        data.y[i] <0 || //access y[i] in memory
        data.z[i] < 0) //access z[i] in memory
    {
        data.x[i] =0;
        data.y[i] =0;
        data.z[i] =0;
    }
}
```

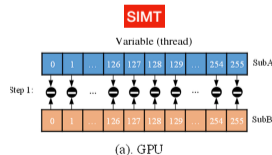
SOA layout

Heterogeneous computing for Event reconstruction in LHCb

Parallelization example of subtraction between 2 sets of data



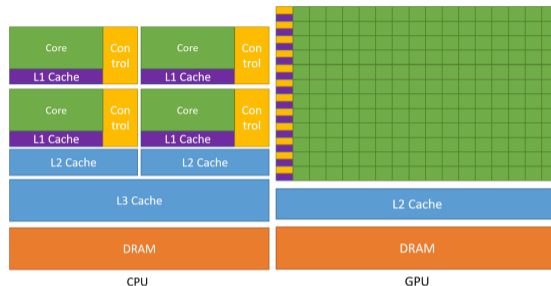
Pretty much the same concept for Single Instruction Multiple Thread (SIMT) on GPU



Heterogeneous computing for Event reconstruction in LHCb

Final consideration about architectures

- CPU: tens of *threads*
branching **IS NOT** a penalty
 - $n_{evt}/s \propto n_{threads}^{--} \times f_{clock}^{++}$
- GPU: thousands of *threads*,
branching **IS** a penalty
 - $n_{evt}/s \propto n_{threads}^{++} \times f_{clock}^{--}$



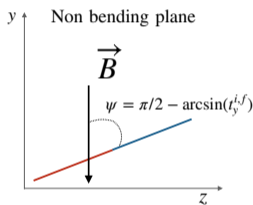
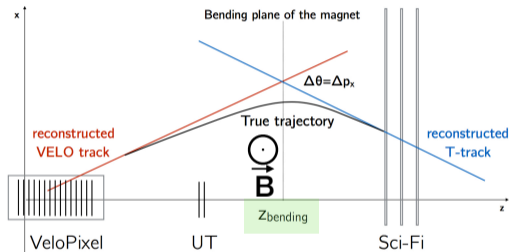
CPU

GPU

Parameterizations of \vec{B} field for fast tracking

Build parameterizations (q/p example)

- Access full \vec{B} field map and material interaction map is extremely time consuming
- Instead parameterize effect of B on tracks with polynomials.
- Example: evaluate q/p from reconstructed tracks.



$$\frac{q}{p} = \frac{1}{\int |d\vec{L} \times \vec{B}|_x} \cdot \left(\frac{t_{x,f}}{\sqrt{1 + t_{x,f}^2 + t_{y,f}^2}} - \frac{t_{x,i}}{\sqrt{1 + t_{x,i}^2 + t_{y,i}^2}} \right)$$

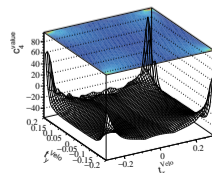
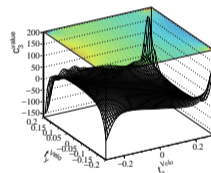
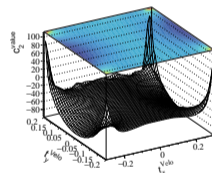
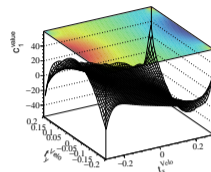
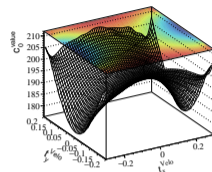
Assuming $\vec{B} = (0, B_y, 0)$

$$\int |d\vec{L} \times \vec{B}| \propto \int \sin(\psi) \cdot B_y(x, y, z) \cdot dz$$

- $t_x^{i,f}, t_y^{i,f}$ available in pattern recognition.
- Parameterize $\int \vec{B} \times d\vec{L} = f(t_x^i, t_y^f, dSlope)$.
- In order to do this, generate toy tracks in t_x^i, t_y^f and for all the possible momentum spectrum.

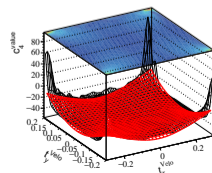
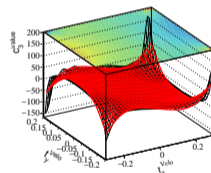
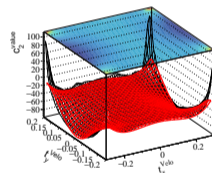
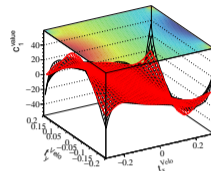
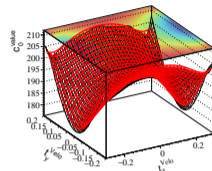
Build parameterizations (q/p example)

- Expand $f(t_x^i, t_y^i, dSlope) = \sum_{i=0}^n c_i dSlope^i$.
- Fit for it in each $(t_{x,i}, t_{y,i})$ generated (4th order)



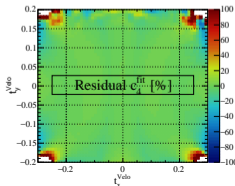
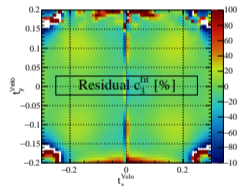
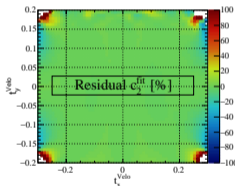
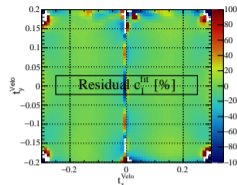
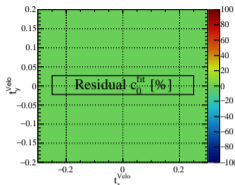
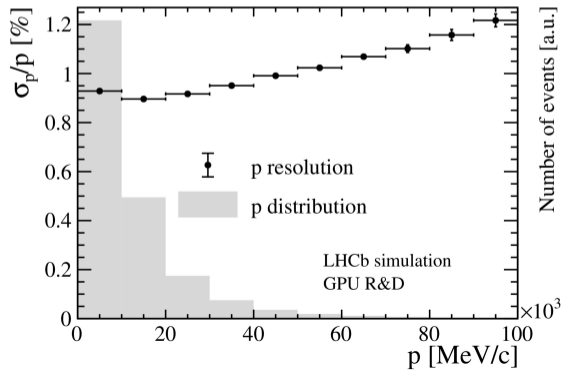
Build parameterizations (q/p example)

- Expand $f(t_x^i, t_y^i, dSlope) = \sum_{i=0}^n c_i dSlope^i$.
- Fit for it in each $(t_{x,i}, t_{y,i})$ generated (4th order)
- Construct and fit 2D polynomials in $t_{x,i}, t_{y,i}$ respecting observed symmetries in c_i .



Build parameterizations (q/p example)

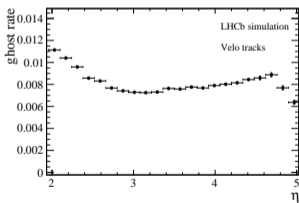
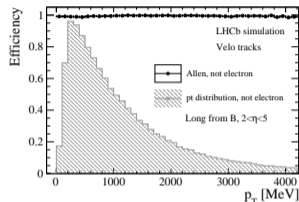
- Expand $f(t_x^i, t_y^i, dSlope) = \sum_{i=0}^n c_i dSlope^i$.
- Fit for it in each $(t_{x,i}, t_{y,i})$ generated (4th order)
- Construct and fit 2D polynomials in $t_{x,i}, t_{y,i}$ respecting observed symmetries in c_i .
- Check residuals of parameterization and inject it back in pattern recognition using reconstructed quantities



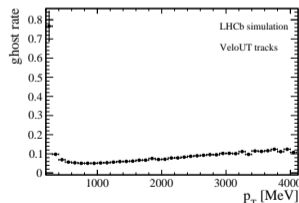
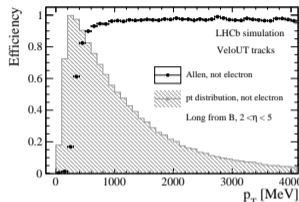
HLT1 performance

HLT1 physics performance: Track reconstruction efficiencies

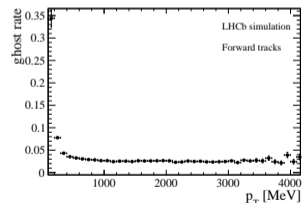
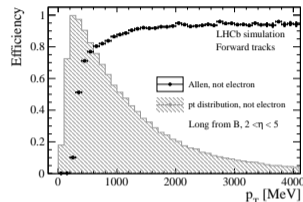
VELO tracking



VELO-UT tracking



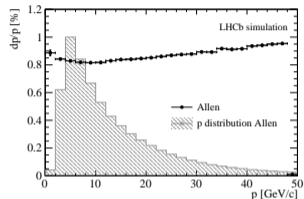
SciFi Tracking



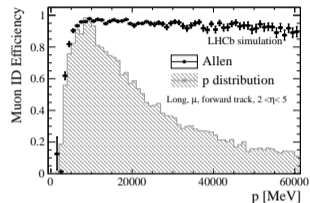
- Tracking down to 0 p_T would cost 20% extra in GPU resources.

HLT1 physics performance: Resolution, PV & Muon ID

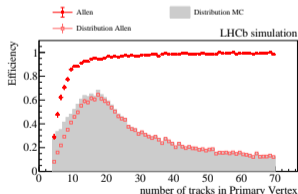
Momentum resolution



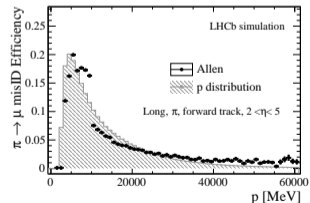
Muon ID efficiency



Primary Vertex reconstruction efficiency



$\pi \rightarrow \mu$ mis-ID efficiency



HLT1 physics performance: Selections

Trigger	Rate [kHz]
ErrorEvent	0 ± 0
PassThrough	30000 ± 0
NoBeams	5 ± 3
BeamOne	18 ± 5
BeamTwo	8 ± 3
BothBeams	4 ± 2
ODINNoBias	0 ± 0
ODINLumi	1 ± 1
GECPassthrough	27822 ± 52
VeloMicroBias	26 ± 6
TrackMVA	409 ± 23
TrackMuonMVA	23 ± 6
SingleHighPtMuon	7 ± 3
TwoTrackMVA	503 ± 26
DiMuonHighMass	131 ± 13
DiMuonLowMass	177 ± 15
DiMuonSoft	8 ± 3
D2KPi	93 ± 11
D2PiPi	34 ± 7
D2KK	76 ± 10
Total w/o pass through lines	1157 ± 39

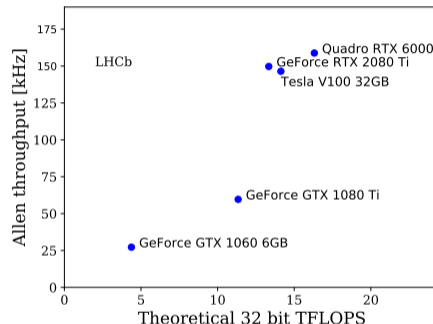
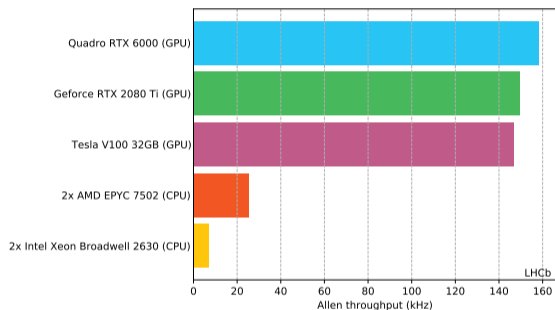
- From 30 MHz \rightarrow 1 MHz event rate reduction
- Can execute $\mathcal{O}(100)$ lines with almost no effect on throughput
- Selection efficiencies fulfill HLT1 requirements for broad range of decays of interest for LHCb

Signal	GEC [%]	TIS-OR-TOS [%]	TOS [%]	GEC \times TOS [%]
$B^0 \rightarrow K^{*0} \mu \mu$	89 ± 2	91 ± 2	89 ± 2	79 ± 3
$B^0 \rightarrow K^{*0} e e$	84 ± 2	69 ± 2	62 ± 2	52 ± 3
$B_s^0 \rightarrow \phi \phi$	83 ± 3	76 ± 3	69 ± 3	57 ± 3
$D_s^+ \rightarrow K^- K^+ \pi^+$	82 ± 4	59 ± 5	43 ± 5	35 ± 4
$Z \rightarrow \mu \mu$	78 ± 1	99 ± 0	99 ± 0	77 ± 1

GEC : Global Event Cut, TIS: Trigger Independent of Signal, TOS: Trigger On Signal

- Selections for alignment and monitoring implemented as well
- On going: adding more selections

HLT1 computational performance



- Full HLT1 at 30 MHz input rate can be processed using 215 GPU cards. Available slots are 500.
- Computing performance scales well with GPU generations: improvements expected.
- Room already available to include more algorithms to further expand LHCb capabilities, e.g. *PID*, long-lived track reconstruction, e optimized track reconstruction....

Conclusion

What we learned

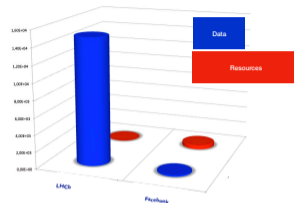
- LHCb upgrade trigger strategy: reconstruct tracks at 40 MHz using available budget.
- Reconstruct tracks at collision rate using new/upgraded subdetectors.
- Track reconstruction in LHCb done from scratch for Run III.
- We learned a lot about new detectors and how to take out the best from them.
- We learned a lot about new architectures and how to take out the best from them (CPU, GPU).
- Redesign EventModel and Algorithms for easy parallelization
- Crucial to have a strong synergy between computing and physics aspects

My (personal) conclusion

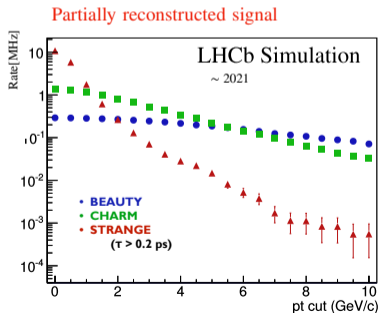
- Tracking is all about problem solving.
- There is **never** an ideal algorithm.
- Often, track reconstruction \sim *let's try this approach and see if it will work or re-tuning* parameters.
- However, when implementing algorithms (aiming to be fast and efficient) one can always derive from first principle considerations meeting *computing, detector* and *physics* knowledge, if an approach is better than another.
- The more advanced detectors and triggers will be, the wider the multi-domain expertise has to be.

Why triggering?

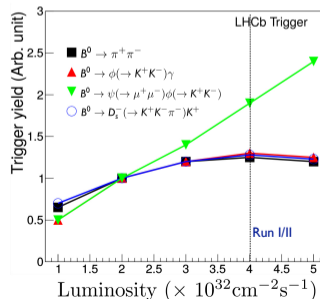
- LHCb produces ~ 1 TB/second \rightarrow **15kPB/year**
- LHCb budget is $\mathcal{O}(10)$ M€/year $\rightarrow \mathcal{O}(600)$ €/PB
- By comparison, Facebook process **180 PB/year**
- Facebook budget is $\mathcal{O}(500)$ M€/year $\rightarrow \mathcal{O}(2.700.000)$ €/PB



- A problem of signal saturation

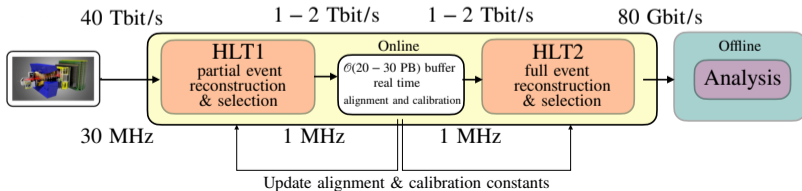
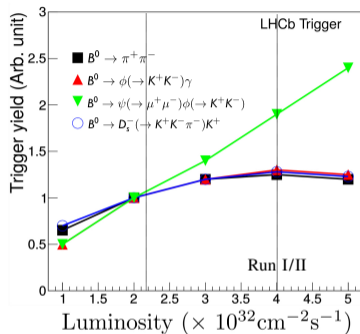


Hardware hits a limit of effectiveness



LHCb trigger strategy for the upgrade

- **L0 Hardware trigger** output rate of 1 MHz imposed by read-out system fully saturates already in Run 2.
[Higher rate \rightarrow higher $p_T^{L0}(\mu)/E_T^{L0}(h^\pm/e^\pm)$ cuts to keep 1 MHz]
- \rightarrow Full event readout at bunch crossing rate
- \rightarrow Event reconstruction and triggering in real time
- \rightarrow Upgrade and replacement of subsystems
 - Cope with higher occupancy
 - Faster/higher precision tracking
 - Full replace of DAQ to support 40 MHz detector read-out
- **LHCb upgrade trigger strategy:** full software based trigger at 30 MHz (non-empty bunch crossing collision rate)



HLT1 reconstruction: tasks

Highly parallelizable tasks across sizeable set of algorithms

- Full event information copied to GPU (Raw event size 100 kB)
- Process HLT1 at 30 MHz on less than 500 state of the art GPUs.
- Selection reports copied back to CPUs.



Data preparation

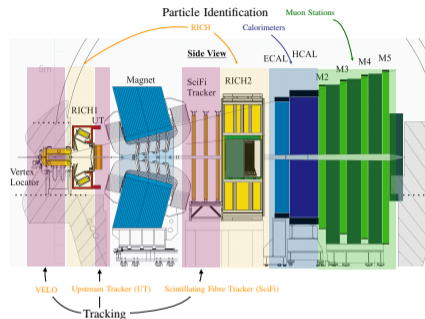
- Decode raw data in
 - 1 VERTex LOcator (VELO)
 - 2 Upstream Tracker (UT)
 - 3 Scintillating Fibre Tracker (Sci-Fi)
 - 4 Muon chambers
- Clustering of VELO pixels into hits

Reconstruction

- Velo tracks reconstruction
- Primary Vertex reconstruction
- Add UT hits to Velo tracks
- Find matching segments in Sci-Fi
- Match tracks to Muon hits
- Make 2-track secondary vertices
- Fit tracks with a (fast) Kalman Filter

Selection

- 1-track selections
- 2-track selections



HLT1 reconstruction on GPUs: parallelization using GPUs

Efficient parallelization can be achieved

- Repeating the same *kernel* or function thousands of times: parallelize intra-event reconstruction.
- Linearize algorithms and algorithm workflows as much as possible
- Organize and redesign data structures in a parallel friendly way for the algorithm purpose
- Pipeline the HLT1 reconstruction in parallel across thousands of events

Raw data decoding in Velo, SciFi, UT, Muon

- Decode binary information from subdetector readout: *parallelize* across readout units and/or sensors.

VELO pixels clustering

- Parallelize across small detector units.

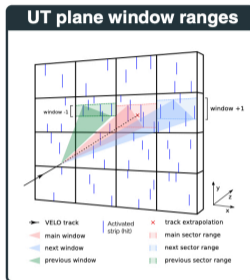
Track reconstruction

- Pattern Recognition: assign/add hits to a track candidate, *parallelize* across hit combinations

Vertexing

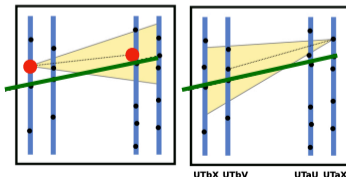
- Combine tracks to form primary and secondary vertices. *parallelize* across tracks and vertex seeds.

Velo UT algorithm



Within Panel(layers)

sector ranges for each input VELO track
forward backwards

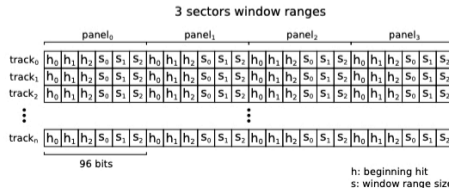


UT decoding (5 kernels):

1. Calculate number of hits to pre-allocate memory
2. Get offsets for efficient access using prefix sum
3. Sort hits into X regions defined by sectors
4. Calculate the permutations needed to sort by Y
5. Decode and sort by Y into sector groups

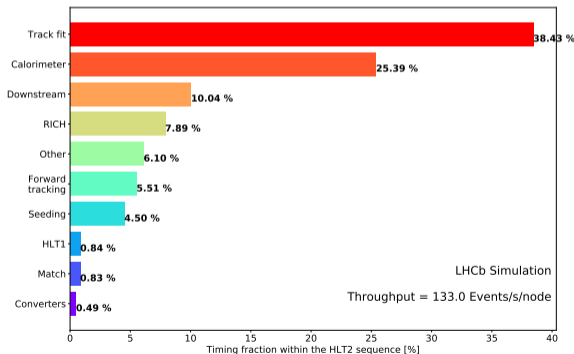
CompassUT tracking:

- Binary search the regions extrapolating VELO track
- Calculate all window ranges for N sectors
- Minimize branching, filter non-valid tracks with shared memory, cache window ranges into shared memory
- Search for triplet/quadruplet with combined forward-backward loop
- Look for best cluster based on configured parameter



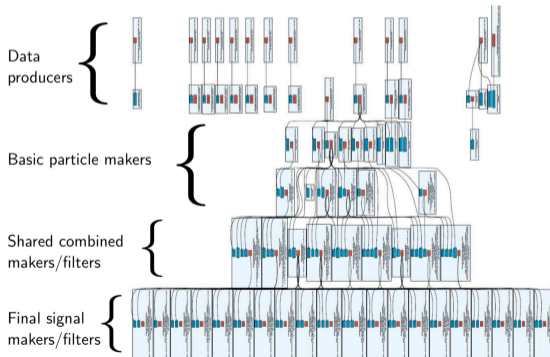
HLT2 reconstruction: tasks

- Using a fully aligned and calibrated detector.
- Offline quality track fit and Particle Identification at 1 MHz input rate
- Knowledge acquired on speeding up CPU solution for HLT1 ported into HLT2



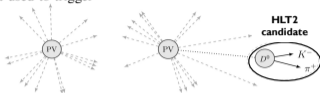
HLT2 selections: the real time analysis paradigm

- Using a fully aligned and calibrated detector.
- Offline quality track fit and Particle Identification at 1 MHz input rate
- Knowledge acquired on speeding up CPU solution for HLT1 ported into HLT2
- Build offline-like candidates in the online system and perform analysis on direct trigger output.

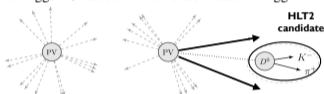


Selective persistency: what is saved to disk?

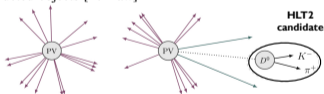
- Only object used to trigger



- Object used to trigger + subset of tracks associated to trigger decision



- All reconstructed objects [no Raw]



15 kB/evt

Increasing persisted event size

Decreasing information

70 kB/evt

Extrapolated throughput to tape during the upgrade

STREAM	rate fraction	throughput (GB/s)	bandwidth fraction
FULL	20%	5.9	59%
Turbo	68%	2.5	25%
TurCal	6%	1.6	16 %
Total	100%	10	100%

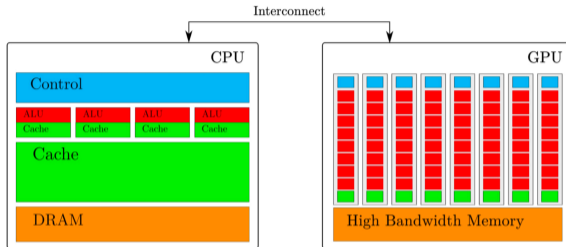
Bandwidth optimization : Trigger output rate [kHz] \times event size [kB] crucial for final storage [up to 80 Gbit/s].

- Offline quality *flexible*-selections available in online system.
- Choose what to store to disk to optimize bandwidth.
- Reduced event format and size \rightarrow keep high signal efficiency using the same bandwidth.
- **Real Time Analysis** concept implemented in Run 2 with Turbo stream becomes the baseline in Run 3.

GPU architecture design

Interconnect between CPU and GPU

- PCIe 3.0: up to 16 GB/s
- PCIe 4.0: up to 32 GB/s



- Avg bandwidth between CPU and host memory
- Low core count/Powerful ALU
- Complex control unit
- Large caches

Latency

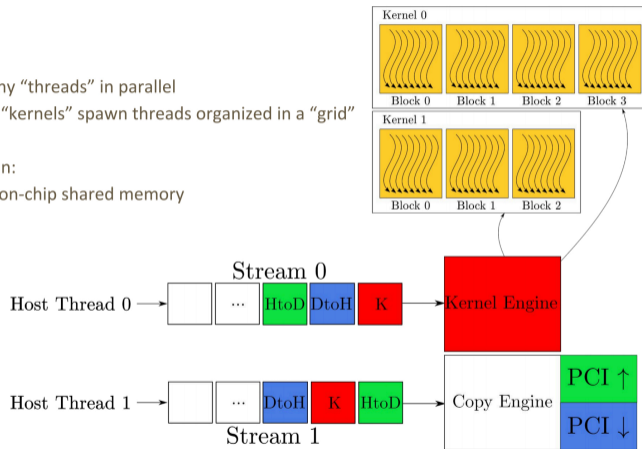
- High bandwidth between GPU cores and GPU memory
- High core count
- No complex control unit
- Small caches

Throughput

Slide taken from [here](#)

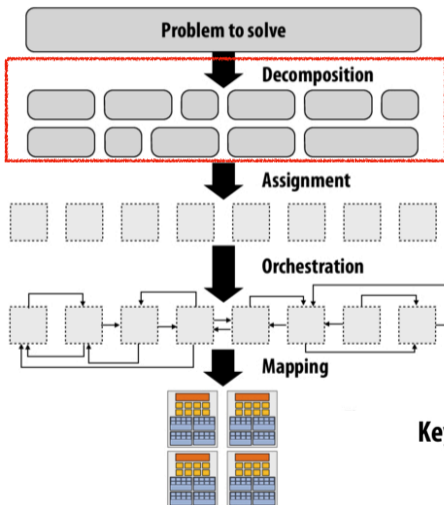
GPU programming model

- GPU code is executed by many “threads” in parallel
 - Parallel functions, aka “kernels” spawn threads organized in a “grid” of blocks
- Threads in the same block can:
 - Communicate via fast on-chip shared memory
 - Synchronize



Slide taken from [here](#)

Create a parallel program



Creating a parallel program

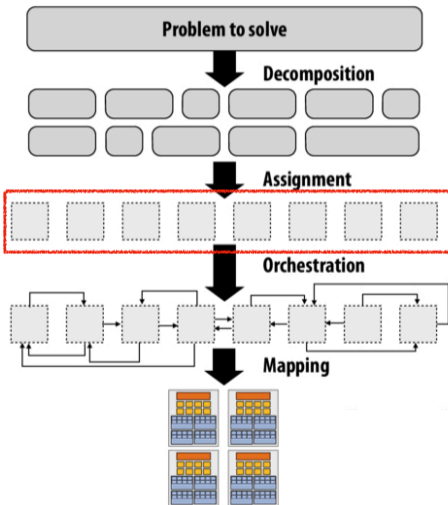
- Thought process:
 1. Identify work that can be performed in parallel
 2. Partition work (and also data associated with the work)
 3. Manage data access, communication, and synchronization

Decomposition

- Break up problem into tasks that can be carried out in parallel
 - Decomposition need not happen statically
 - New tasks can be identified as program executes
- Main idea: create at least enough tasks to keep all execution units on a machine busy

**Key aspect of decomposition: identifying dependencies
(or... a lack of dependencies)**

Create a parallel program



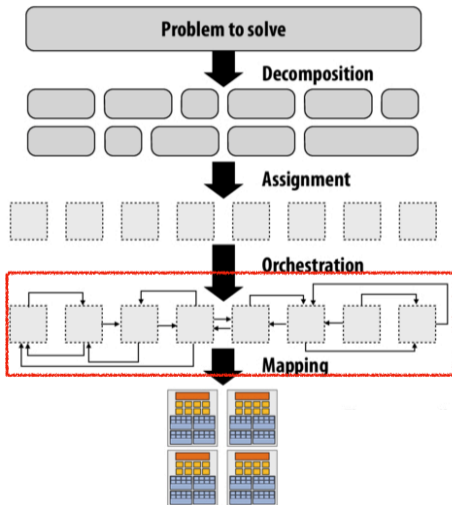
Creating a parallel program

- Thought process:
 1. Identify work that can be performed in parallel
 2. Partition work (and also data associated with the work)
 3. Manage data access, communication, and synchronization

Assignment

- Assigning tasks to threads **
 - Think of “tasks” as things to do
 - Think of the threads as “workers”
- Goals: balance workload, reduce communication costs
- Can be performed statically, or dynamically during execution
- While programmer often responsible for decomposition, many languages/runtimes take responsibility for assignment.

Create a parallel program



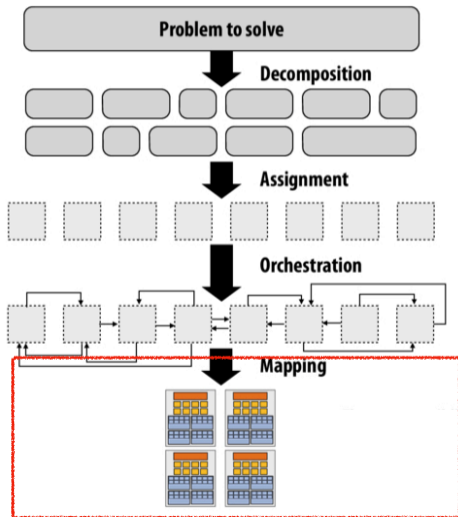
Creating a parallel program

- **Thought process:**
 1. Identify work that can be performed in parallel
 2. Partition work (and also data associated with the work)
 3. Manage data access, communication, and synchronization

Orchestration

- **Involves:**
 - Structuring communication
 - Adding synchronization to preserve dependencies if necessary
 - Organizing data structures in memory
 - Scheduling tasks
- **Goals:** reduce costs of communication/sync, preserve locality of data reference, reduce overhead, etc.
- **Machine details impact many of these decisions**
 - If synchronization is expensive, might use it more sparsely

Create a parallel program



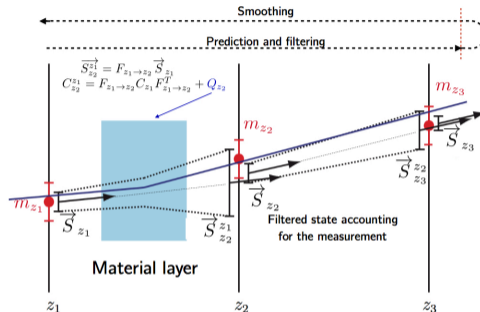
Creating a parallel program

- Thought process:
 1. Identify work that can be performed in parallel
 2. Partition work (and also data associated with the work)
 3. Manage data access, communication, and synchronization

Mapping

- Mapping “threads” (“workers”) to hardware execution units
- Example 1: mapping by the operating system
 - e.g., map kernel thread to CPU core execution context
- Example 2: mapping by the compiler:
 - Mapping ISPC program instances to vector instruction lanes
- Example 3: mapping by the hardware:
 - mapping CUDA thread blocks to GPU cores
- Some interesting mapping decisions:
 - Place related threads (cooperating threads) on the same processor (maximize locality, data sharing, minimize costs of comm/sync)
 - Place unrelated threads on the same processor (one might be bandwidth limited and another might be compute limited) to use machine more efficiently

Full (expensive) track fit



- Track state: $\vec{S}_{z_i} = (x, y, t_x, t_y, q/p)_{z_i}$
- **Prediction:** propagate between 2 \vec{S} with 5×5 propagation matrices.
- **Filtering:** compare propagated \vec{S} to actual measurements m_{z_i} using State-to-Measurements projectors. Minimise χ^2 residual.
- Evaluate best estimate of updated \vec{S} .
- Iterate over all measurements.
- **Smoothing:** perform previous steps in reversed direction.
- Material interaction and noise accounted for enlarging errors when propagating states.