Performance optimization of the online data processing software of CERN's LHCb experiment

Thesis report

Péter Kardos

2018-2019

1 Abstract

write at the end
100-200 words
- what's the problem
- how was it solved
- what are the results
- conclusion: what it means for the future
must be understandable without extra info

Don't read this, it's just a placeholder. So this abstract should be about 100-200 words so I'm just writing some natural text to act as a placeholder. By looping this text a few times, I can probably make a 150 word section. So this abstract should be about 100-200 words so I'm just writing some natural text to act as a placeholder. By looping this text a few times, I can probably make a 150 word section. So this abstract should be about 100-200 words so I'm just writing some natural text to act as a placeholder. By looping this text a few times, I can probably make a 150 word section. So this abstract should be about 100-200 words so I'm just writing some natural text to act as a placeholder. By looping this text a few times, I can probably make a 150 word section. So this abstract should be about 100-200 words so I'm just writing some natural text to act as a placeholder. By looping this text a few times, I can probably make a 150 word section.

2 Introduction

describe the problem in detail
specific to my thesis:
environment:
- CERN's goals/activity
- CERN's hardware infrastructure (accelerators, experiments)
- LHCb's hardware infrastructure
- LHCb's software reconstruction system
problem:
- event rate from detector
- slow trigger → loss of physics (ACTUAL PROBLEM)
- by optimizing individual algorithms (in this thesis)

2.1 About CERN

CERN (European Organization for Nuclear Research) is an international high energy experimental physics research organization situated near Geneva, on the Franco-Swiss border. CERN is host to the world's largest particle accelerator and numerous experiments which aim to provide a better understanding of the universe. The goals of the experiments, among others, are to verify the standard model of particles. TODO: list more concrete goals. [1]

2.2 The accelerator complex [2]

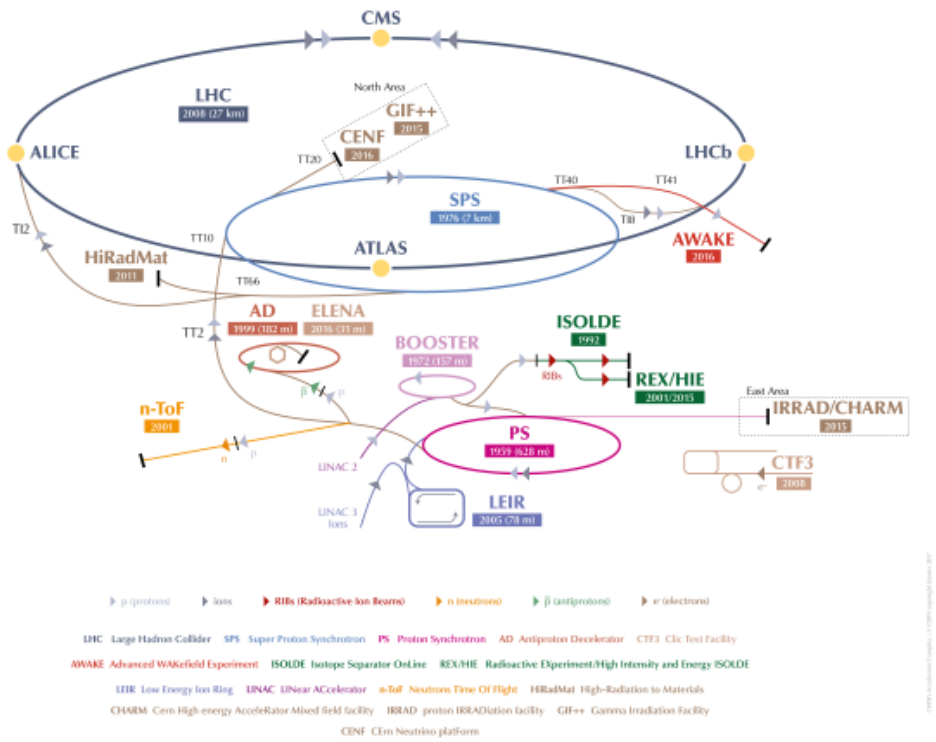


Figure 1: Schematic view of CERN's particle accelerators and experiments. LHC is shown on top by the largest circle. The four main experiments, CMS, ALICE, ATLAS and LHCb are marked with yellow dots along the LHC's circle.

While CERN is mostly known for its Large Hadron Collider (LHC) which this thesis is concerned with, it is home to many more particle accelerators. These accelerators are useful on their own, but from the perspective of the LHC they are used to provide high energy protons that the LHC can further accelerate. Too low energy protons cannot be directly accepted into the LHC, so a sequence of progressively larger accelerators bump the energies up in steps. When a particular accelerator reached its top energy, its beam is simply transferred to a bigger one, finally getting injected into LHC.

The LHC is CERN's largest machine. It can be found inside a circular underground tunnel of a circumference of 27km. There are two accelerators inside the tunnel which accelerate protons so that there is one beam clockwise and another counter-clockwise. Protons are not equally distributed in the beams as they circle around, rather, they can be found in many equally spaced *bunches*.

At specific points along the circle of the LHC, the two beams of opposing directions are made to cross each other's path. As two bunches go through the crossing point at the same time and the individual protons collide[7]. Since each proton carries around 7 TeV

of energy, the collision's yield is about 14 TeV. In the collision, other particles might be born, and that's exactly what scientists are looking forward to analyze.

2.3 Experiments on LHC

As seen on figure 1, the four main experiments dedicated to analyze LHC collisions are ATLAS, CMS, ALICE and LHCb. Consequently, these experiments have huge underground rooms around the collisions points, where they can fit their instruments.

The instruments are meant to track and identify particles created in the collisions, and are thus called particle *detectors*. The type and properties of particles created in the collisions provide valuable data to physicists who are trying to verify and extend the standard model of particles. In most cases, the raw data provided by the detectors is processed by software, which does the tracking and identification.

2.4 LHCb's detector

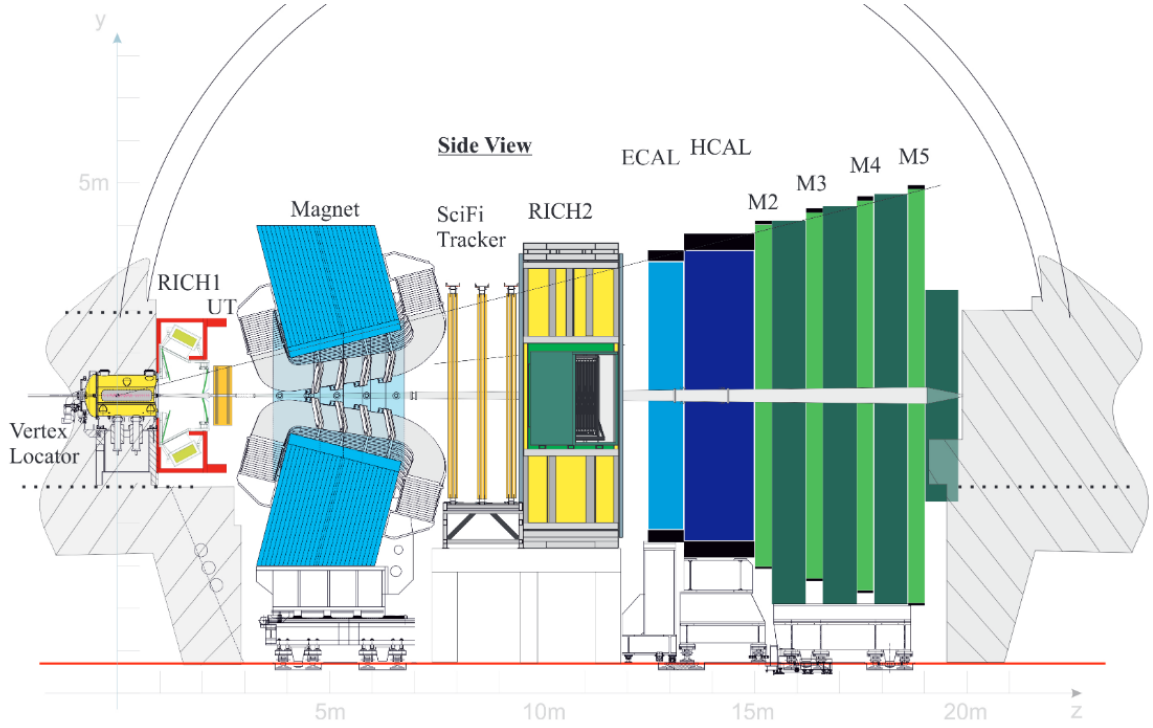


Figure 2: Side view of the LHCb detector.

Figure 2 shows the LHCb detector from the side, which means that the two beams of LHC are going in horizontal directions on the drawing through the middle of the detector. The middle of the detector coincides with the horizontal axis of symmetry.

As seen on the labels, the detector consists of multiple layers of sub-detectors. Each layer has a hole in the center to let the beam pipes through. The two particle beams cross each other inside the Vertex Locator (VELO, at the right in yellow).

While full reconstruction uses all sub-detectors, real-time reconstruction only uses the VELO, the UT (in orange, left of VELO) and the FT (SciFi Tracker, in the middle in

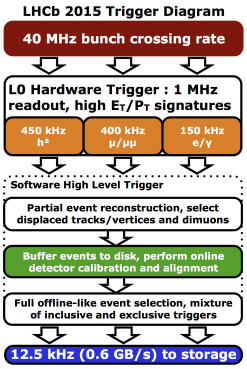
orange). Let's follow the life of a particle from its birth inside the VELO. While flying away from the collision point, it first crosses multiple layers of the VELO. Each layer consists of many pixels, and whenever the particle touches a pixel, the VELO forwards this information to the software. Eventually, the particle leaves the VELO and goes through the UT. The UT has a different mechanism compared to the VELO, but essentially it also records if a particle has gone through one of its layers. Leaving the UT, the particle goes through the large magnet, where its path is bent due to the Lorentz force by an amount dependent on its electric charge and momentum. The particle then keeps going straight through the FT, where its location is registered as usual. Note that for each bunch collision, hundreds of new particles go through the detector as described above. The reconstruction software tries to make sense out of the large number of registered particle positions by using knowledge about the magnetic field, and eventually identifies the individual particles and their tracks.

The purpose of real-time (sometimes called *online*) reconstruction is to determine what data to store. Most collision events are not interesting at all from a physics perspective, and only a small fraction is kept for long-term storage. Storing all data would be unfeasible because of its sheer amount. The software doing the online reconstruction and selection is called the *trigger*.

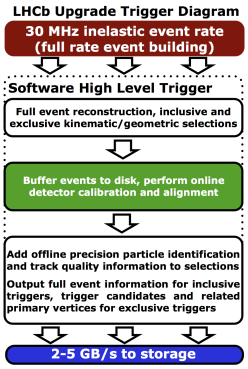
2.5 The 2018/19 upgrade of LHCb

The catch with the above described detector is that it does not really exist yet. The LHC will be shut down in 2018 december for maintenance, and that's when the LHCb collaboration will upgrade its detector to the one above. (The current detector is similar in construction, that's why it's referred to as *upgrade*.)

With the upgrade, software data processing will change significantly as well. Around 30 million bunch collision events occur every second, each of which go through the trigger. With the current detector, triggering is first done by hardware electronics, selecting only 1 million events per second, which are then further culled on a big server farm by software written in C++. After the upgrade, the hardware trigger is dropped and the entire trigger runs in software. This puts the high requirement on the software trigger to reconstruct and cull all the 30 million events every second.



(a) Trigger data flow scheme before the upgrade



(b) Trigger data flow scheme post-upgrade

Figure 3: Comparison of the two triggering solutions

2.6 Overview of the trigger software

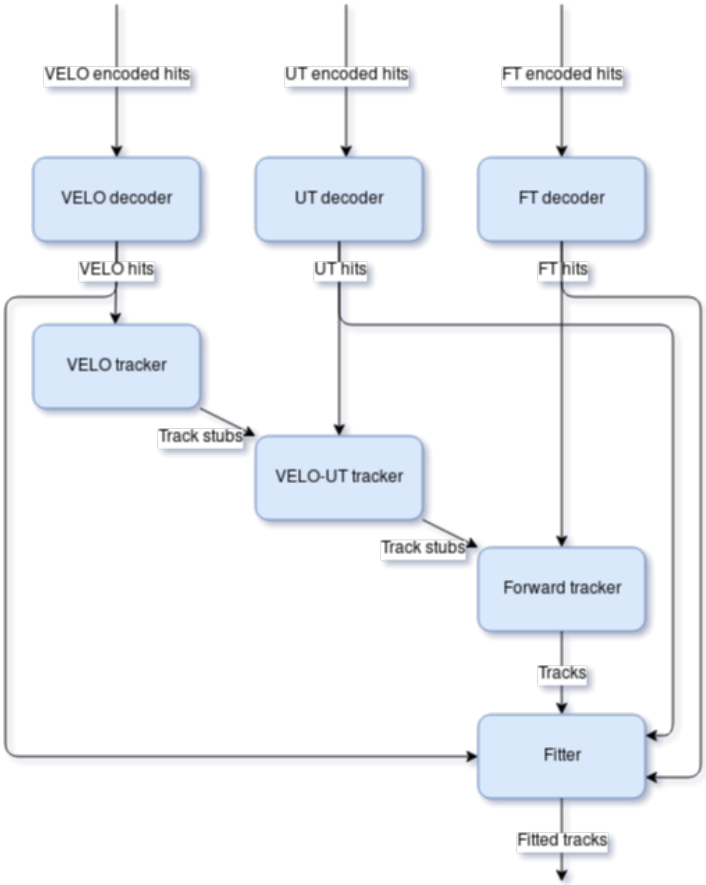


Figure 4: Simplified view of algorithms that perform online reconstruction.

The trigger software is basically the online reconstruction code and additional logic to select which events to keep. The online reconstruction software can be broken down to individual pieces referred to as *algorithms*, each of which serves a specific purpose. First of all, the highly compressed data that arrives from the detector has to be decoded to acquire hits that describe the position and measurement error of points where a particle has been seen. Second, the particles' tracks are progressively reconstructed by starting from the collision point in the VELO, and appending hits to an existing track. In the VELO and UT, we are looking for hits that form a straight line, and we try to match this straight track to another similarly straight track inside the FT. Finally, the track goes through fitting, which tries to modify the existing rough track to more closely line up with the hits it was produced from.

2.7 The aim of this thesis project

As mentioned, the abandoning of the hardware trigger stage highly increases the load on the software trigger. The main goal of this project is to optimize the current software

trigger to make it about 3 times as fast. Failure to do so will result in valuable events being dropped, thus reducing the physics potential of the experiment.

Current computing hardware has changed significantly from the ones the software trigger was originally made for. The even larger gap between memory and CPU speeds demands a more efficient use of CPU caches. Additionally, CPU instruction sets now include SIMD operations, which can, for example, do 4 floating point operations in place of one in the same amount of time. Furthermore, modern CPUs have a complex logic for branch prediction and instruction pipelining, which require code to be tailored to serve them.

To exploit the full capability of current hardware, not only individual pieces of the trigger software need to be changed, but the global data flow also has to be rethought and optimized.

During this thesis project, I will be helping the LHCb collaboration to reach its optimization goals for the software trigger.

3 Choosing optimization targets

Explain the choice of initial choice of algorithms, based on the pie chart diagram and logical reasoning of our goals (i.e. what's needed).

As mentioned in 2.6, the reconstruction consists of individual algorithms which account for the bulk of the computation. (Scheduling the algorithms and culling decisions account for a much smaller CPU load.) It is straightforward to first start optimizing the algorithms which take the largest chunk of available computing power.

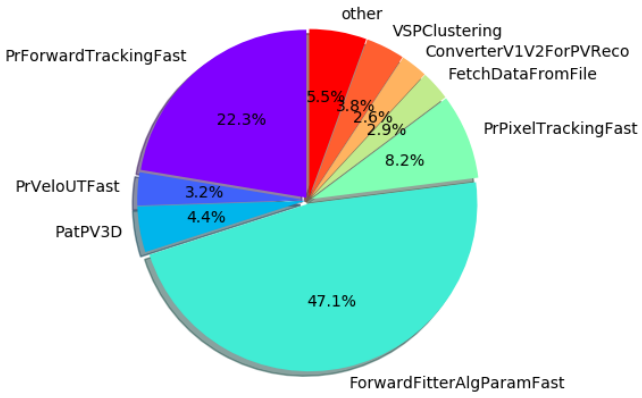


Figure 5: Workload split among HLT1 algorithms.

Looking at figure 5, we can see that the parametrized Kalman fitter takes nearly half the CPU budget, followed by the forward tracking which takes roughly a quarter. Based on this and initial performance profiling of the algorithms for hotspots, I decided to first examine and optimize the Kalman fitter.

4 Parametrized Kalman Fitter

As described in 2.6, the track is reconstructed incrementally, start with velo hits, extended by UT hits and finally adding the FT hits. This process, however, is not so accurate. This manifests itself in the creation of *ghost tracks* and missed tracks, and generally, tracks are only roughly aligned with the hits they were made from. Ghost tracks are tracks that did not exist in the real collision, they are merely artifacts of the reconstruction algorithms. As such, ghost tracks are highly undesirable, but this is where the Kalman fitter comes into play. The Kalman fitter basically refines the rough tracks that are spit out by preceding algorithms. The state of a particle can be described by its position, direction, and the quotient of its charge and momentum. The Kalman fitter first estimates the particle's state at its birth position based on the Velo hits alone. After that, it extrapolates the state of the particle to the next hit, or in other words, simulates the particle's travel until the next hit using the laws of physics. The new, *predicted* state will have some deviation to the

observed state (that is, the hit), however, the Kalman fitter can make a mathematically optimal estimate for the true state based on the prediction and observation. The very new optimal state estimate will then be extrapolated to the next hit again, and this repeats for all the hits of the track. As a result, the estimated state or path of the particle aligns more closely with the observed hits. In the case of ghost tracks, we can expect to have large deviations between the optimal estimated states and the observed hits, which could slipped through initial reconstruction algorithms but show up for the fitter. Such tracks are removed from the list of tracks, and that's why fitting is important.

4.1 Performance profiling for hotspots in the Kalman fitter

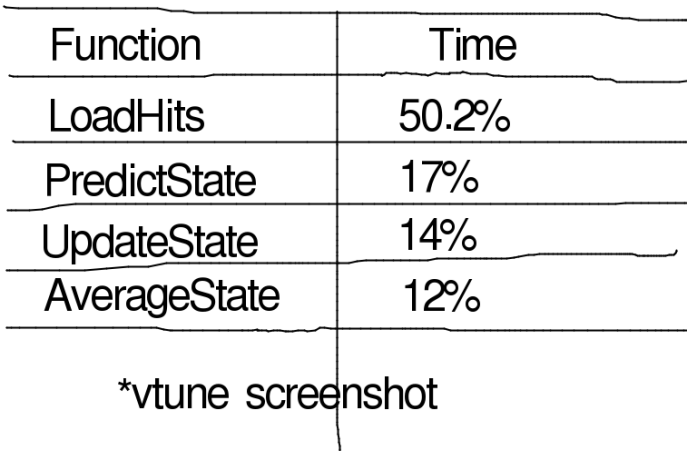


Figure 6: Hotspots, or which parts of the Kalman fitter takes most of the time. Measured by Intel VTune Amplifier XE. MAYBE add appendix explaining profiling and vtune.

Figure 6 shows what fraction of the CPU time is spent in each individual function of the code. We can nicely see how the theoretical steps of the Kalman fitting map to the functions:

- LoadHits: acquires position and measurement error of hits
- PredictState: extrapolates the state to the next hit
- UpdateState: makes an optimal estimate for the true state using the predicted state and the measured hit
- AverageState, ExtrapolateToVertex, etc.: various operations

There is a major and obvious problem however: just acquiring the data on which the computation is done should not take over 50% of the Kalman fitting, but more like 1%.

4.2 Loading hits in detail

Careful examination reveals the way hits are loaded through the so-called *Measurement providers*.

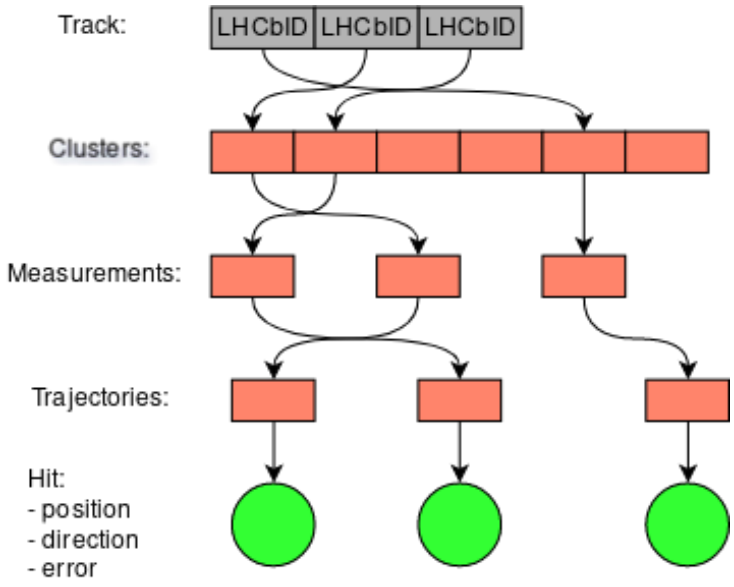


Figure 7: Illustration of how hit information is acquired from the array of LHCbIDs stored inside the Tracks. Contiguous array of clusters correspond to contiguous DRAM memory regions, while distinct objects, i.e. measurements have no spatial locality.

When a particle hits a detector, the identifier of the element of the detector that was hit is recorded. (Detector elements are analogous to the pixels of a digital CCD camera.) These elements are basically unambiguously identified by the so-called *LHCbIDs*, so it is enough to store the IDs inside the Track object and all information (such as location of the hit, measurement error) can be recovered.

Over the years however, this system grow unnecessarily complex resulting in a dramatic slowdown. Clusters, containing some basic information about the hit, such as its location, are stored inside measurement providers as a large array. In order to find the cluster that corresponds to the ID, this whole array is searched linearly. Once the cluster is found, a *Measurement* object is allocated on the heap and initialized from it. Finally, another object, called a *Trajectory*, is queried from the measurement, from which the data actually required can be extracted. The storage of clusters and creation of measurements is handled by *MeasurementProviders*. Additionally, we can distinguish separate measurement objects for the Velo, UT and FT hits.

As seen, this is a convoluted process, involving an asymptotically unacceptable linear search and a lot of dynamic memory allocation. Dynamic allocation is not only slow, it highly suffers from thread contention at the operating system level in our multi-threaded software. Additionally, the individually allocated objects are scattered around in memory, resulting in poor CPU cache performance MAYBE add appendix explaining caches.

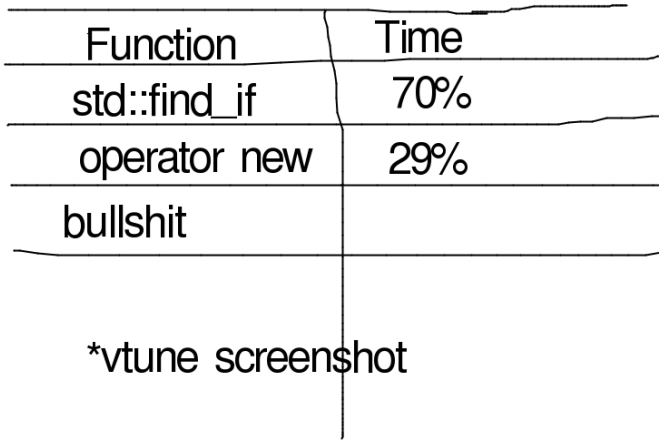


Figure 8: Breakdown of CPU usage of the LoadHits function

Figure 8 clearly shows that the linear searches (std::find_if) account for the largest part of loading hits, while memory allocation and additional calculations fill the rest. This aligns with the theory based on analysing and understanding the code.

4.3 Simplifying the data loading

To avoid this long chain to acquire the required data, the hits should be directly stored inside the Track rather than only by their IDs. Ideally, this would not incur any performance penalty, since the algorithms preceding the Kalman fitter all use the position and error information associated with a hit, so the detector element identifier is fully decoded anyway.

As described, the Track object has the following content (largely simplified):

```
struct Track {
        std::vector<LHCbID> ids;
};
```

In the new model, the following structure is used:

```
struct TrackHit {
        Vector3D beginPosition;
        Vector3D endPosition;
        float errorX;
        float errorY;
};

struct Track {
        std::vector<LHCbID> ids;
        std::vector<TrackHit> veloHits;
        std::vector<TrackHit> utHits;
```

```
        std::vector<TrackHit> ftHits;
};
```

Notice how the IDs are kept: the unfortunate reason for this is that other algorithms rely on these, and they cannot be removed in this first iteration. This structure, however, completely eliminates clusters, measurement and trajectories from the chain, and the Kalman fitter reads the contiguously stored information straight out of the track. This does not stress the memory allocator and is friendly for the caches.

4.4 Performance profiling of the simplified model

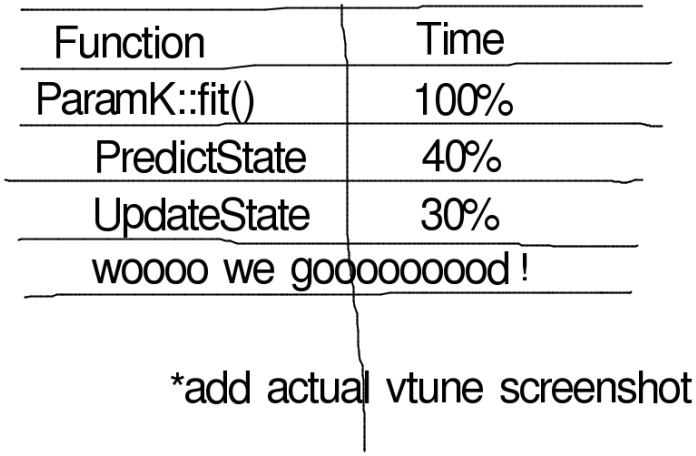


Figure 9: Breakdown of the Kalman fitter

Figure 9 shows that with the new data model, the previous CPU hog, LoadHits, has completely disappeared, now accounting only for 2% of the fitting.

As the parametrized Kalman fitter takes about 47% of the entire reconstruction sequence, and about 50% of the fitter's computing load was removed by the above described code changes, we would expect an overall speedup of 31%. When measured, throughput increases from 4450 events processed per second to about 4850 events/second, or about 9.2%. As this is way less than the predicted 31%, the question arises as to where the performance is gone. First of all, three new data members were added to the Track to store the new TrackHits, and nothing has been removed. As Tracks are copied in the code, the three std::vectors also have to be copied, which involves dynamic memory allocation (a well-known performance drag) and memory copying. Second, part of the code that produces the TrackHits from other objects was not removed, but merely moved out of the fitter to other algorithms. The data conversion to TrackHits, along with adding the TrackHits to the vectors and allocating the memory of the vectors adds additional overhead. In order to achieve the projected performance improvements, these issues have to be fixed and optimized.

4.5 Second round of optimisation

Besides the TrackHits (or IDs), the Track contained three additional dynamically allocated std::vectors, which were filled with valid data but were not necessary from a computing point of view. Removing these data members confirmed the hypothesis by which the additional data members in the track slowed down the algorithm sequence: I observed an increase of 17% in throughput (on top of the 9.2%) when removing these members. While this can be regarded as an optimization independent to the fitter itself, it gains back the speed lost with the additional members required by the fitter.

4.6 Third round of optimisation

To trade physics quality for performance, event culling decisions can be made earlier in the reconstruction sequence, before fitting. This results in fitting taking a lot smaller part of the entire sequence while other algorithms become more prevalent. My work, although sped the fitting up, slightly slowed other algorithms down. Consequently, the *best physics* case experienced a large increase in throughput, but the *best throughput* case got slightly slowed down. In an attempt to restore the performance of the other algorithms, I had to further analyze performance.

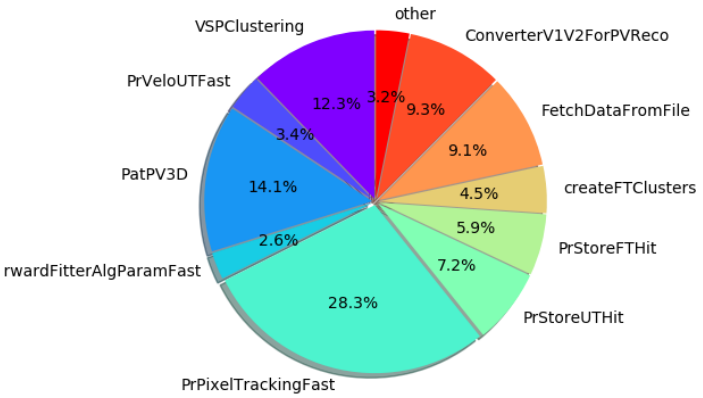


Figure 10: Distribution of CPU time among algorithms in the *best throughput* case with early event culling

TODO:
yeah, cool, should get like 130 percent scaling!
but we only get 109.2% so WTF?
bastard memory allocations to blame as usual
add some shit about disassembly and fp32 to fp64 conversion so i sound smart

5 TBD

6 Conclusion

- summarize my own contributions
- summarize achieved results
- make conclusions about them
- how it affects the future
BRIEFLY

7 References

[1] About CERN:
https://home.cern/about

[2] The accelerator complex:
https://home.cern/about/accelerators

[3] About the Large Hadron Collider:
https://home.cern/topics/large-hadron-collider

[4] About the Large Hadron Collider beauty experiment:
https://home.cern/about/experiments/lhcb

[5] Why collide lead ions:
http://alicematters.web.cern.ch/?q=FAQ-why-lead-ions

[6] Energy of the LHC:
https://home.cern/about/engineering/restarting-lhc-why-13-tev

[7] https://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/collisions.htm