Allen: LHCb GPU-based trigger

Diego Martínez Santos (GAIN), Adrián Casais Vidal (USC), Xavier Vilasís (La Salle)

GPU

- Each GPU core is slower than that of a CPU, but they outnumber CPU cores by a large amount (thousands vs tens or hundreds)

- For tasks that are paralelizable, GPU's are usually way better in computing power per euro than CPU's are

- eg, for a ML fit in Bs→J/ψΦ decays we find that a GPU 3090 (~2k euro) is ~3-4 times faster than a 7k euro Ryzen.

- **Typically computing power per euro is >~ 10x better in GPU than in CPU.**

GPU

- GPU's are programmed using CUDA (NVIDIA models) or OpenCL (any model). They are very similar to C/C++

- Can also be managed in python via pycuda/pyOpenCL

- Some useful models

Model	Memory (GB)	F32 speed (TFLOPS)	Price (NA, USD)
RTX 2080 Ti	11	12	~1000 (2018)
RTX 3080	10	25-30	700
RTX 3080 Ti	12	34	1200
RTX 3090	24	36	1500
PS4 (c. AMD)	8	1.8	~400 (2013)
PS5 (c. AMD)	16	10	~400

3

Host and device

- In CUDA world, 'host' refers to the CPU and 'device' to the GPU

- Functions that run on the GPU are defined as _device_ bla

- Functions that run on the CPU are defined as _host_ bla

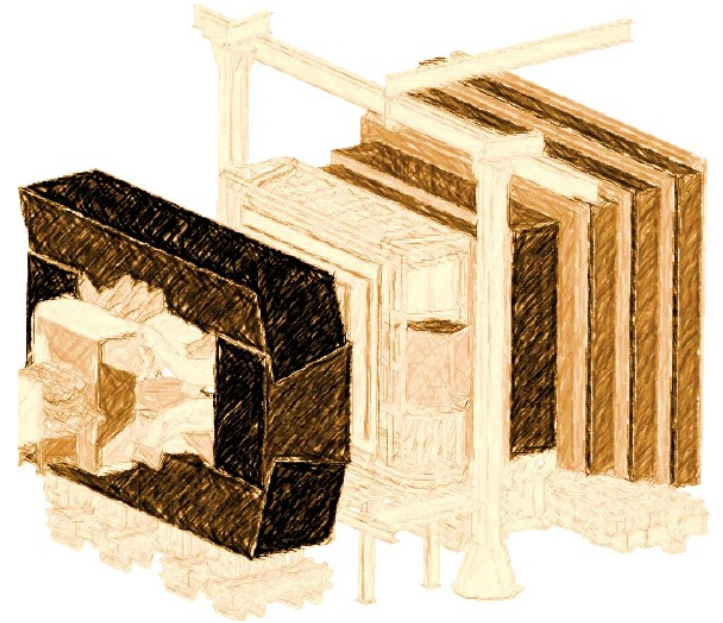- Functions that run on both are defined as _global_ bla
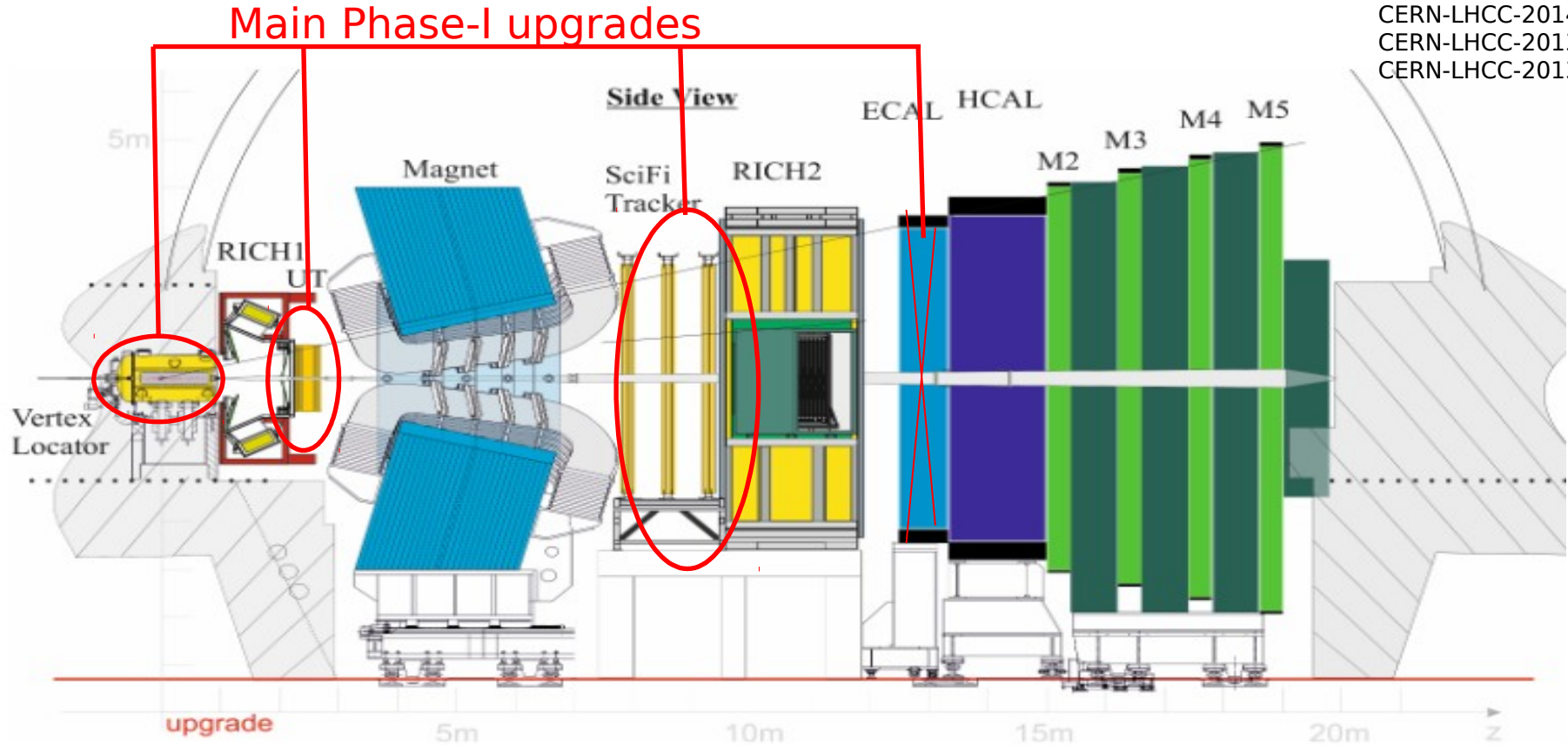
(+ RAM)

Host

Device

LHCb

- Experiment at the LHC

- Studies b and c decays

- LHC produces bunch crossings at a rate of 40 M per second

- ~25% are empty, so the "real" input rate is 30 M events per second (30 MHz)
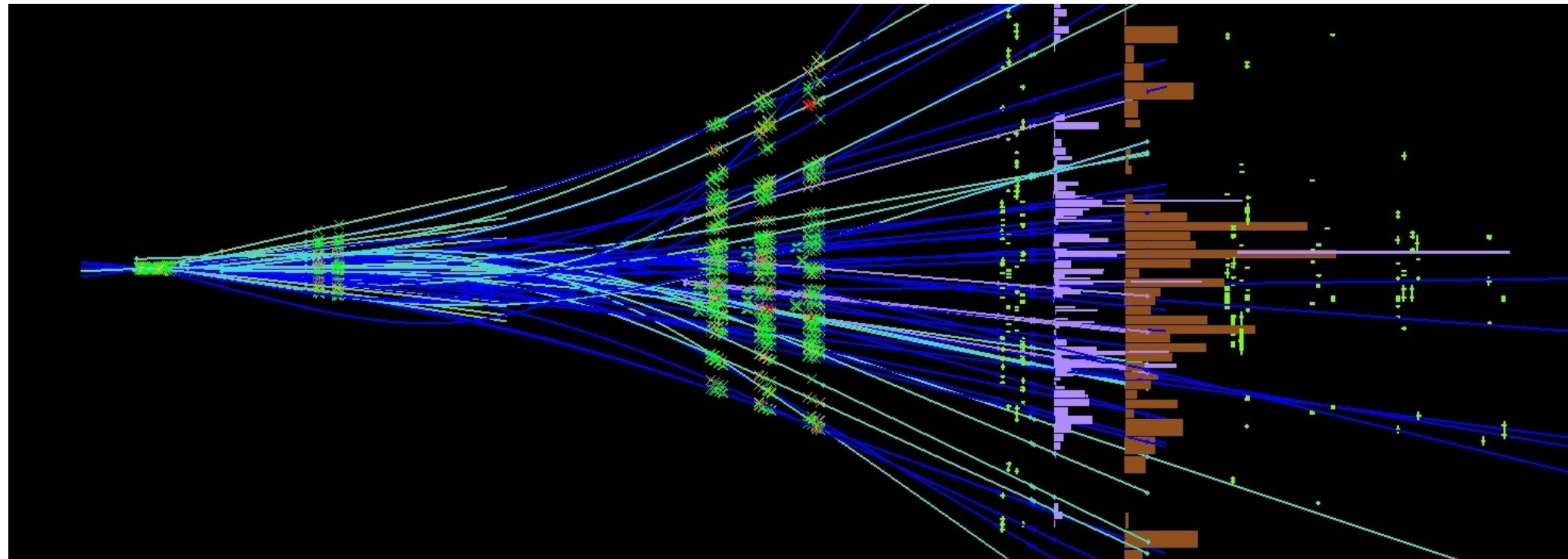
LHCb tracking

Main Phase-I upgrades

Side View

ECAL HCAL

M2 M3 M4 M5

Magnet

SciFi Tracker

RICH2

RICH1 UT

Vertex Locator

5m

upgrade   5m   10m   15m   20m   z

Long Track: hits in Vertex Locator & UT & SciFi

LHCb tracking

Long Track: hits in Vertex Locator & UT & SciFi

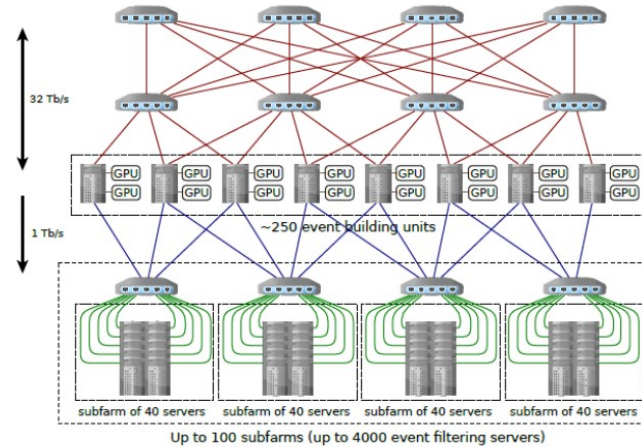# Allen: LHCb GPU-based trigger
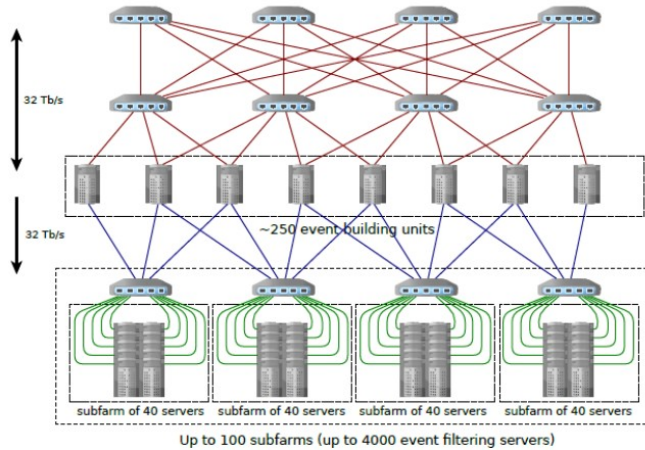
CPU paradigm (baseline):

* The CPU farm processes both HLT1 & HLT2
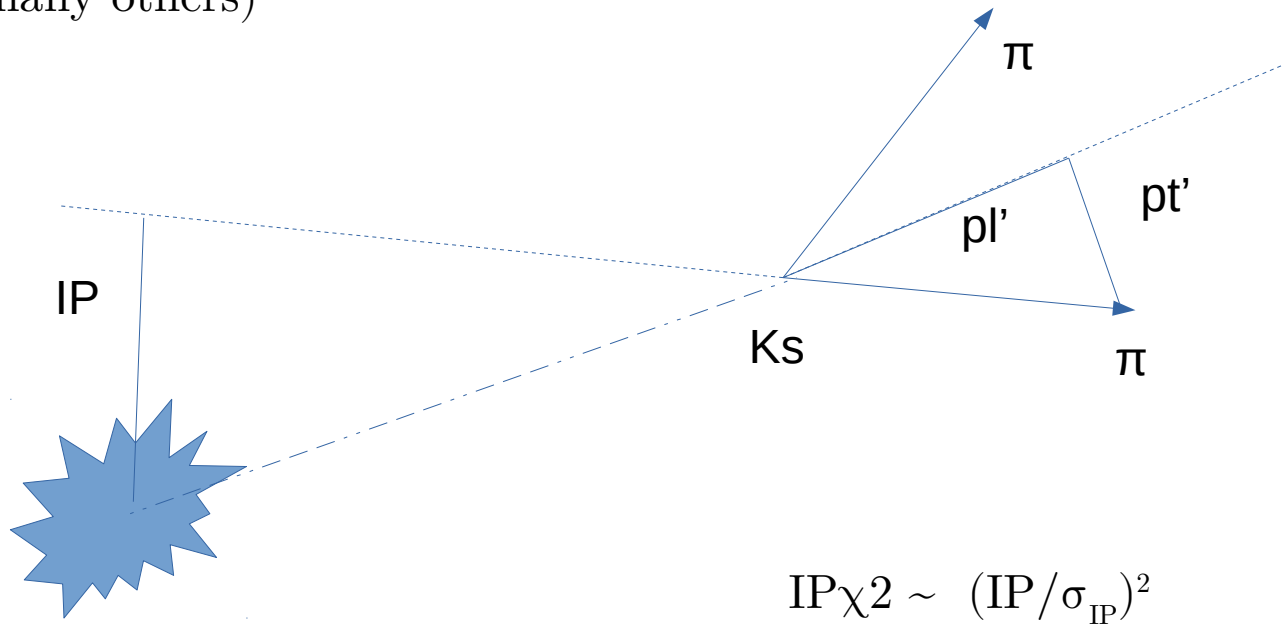
GPU paradigm:

* The GPU's replace the event building units, and run HLT1
* CPU farm has to process only HLT2
* Mind also the cheaper network system → the cost reduction in network is already enough to pay the GPU farm!

V0 decays

V0 decays are neutral particles decay to two charged particles (Ks→ ππ, Λ→ pπ but also many others)

π

pt'

pl'

IP

Ks

π

$$IP\chi2 \sim (IP/\sigma_{IP})^2$$

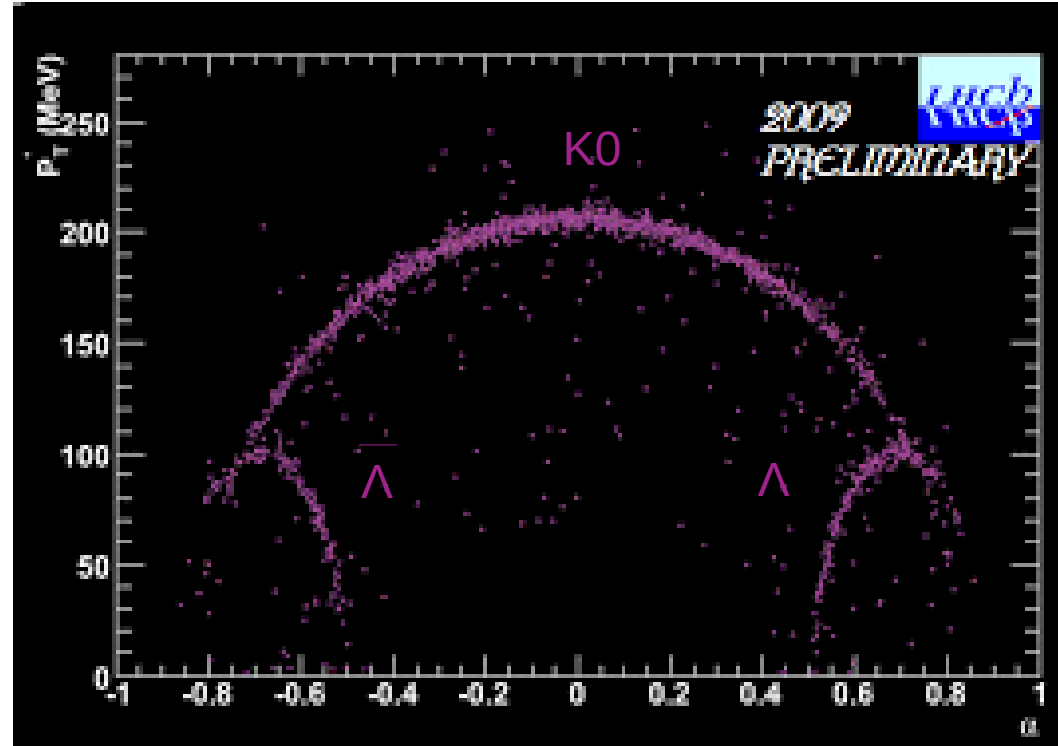# Armenteros-Podolanski plot

Y axis: pt'

X axis: α

$$\alpha = (pl'^+ - pl'^-)/ (pl'^+ + pl'^-)$$

Allen lab, part I

- Optimize threads/memory for your card:

+ Find which is your card: nvidia-smi -a (or -q)

+ Compile w/o ROOT (should be done already)

+ ./Allen **-n 500 -r 1000 -m 1000 -t 10** --device 0 --mdf `ls -d ../../DATA/*mdf | xargs echo | sed 's/ /,/g'`

 -m : memory per thread
 -t :number of threads
 --device: GPU ID (details later)
-n : number of events, -r: repetitions

+ Note that **m x t** should be smaller or equal than the card memory

Allen lab, part I: throughput


```
 Number of seconds between writes of the monitoring histograms (0: off) (--monitoring-sav
 Ignore signals to update non-event data with each run change (--disable-run-changes): 1

Opened ../../DATA/upgrade_mc_minbias_scifi_v5_000.mdf
Starting timer for throughput measurement
Processing complete          This is the throughput.
108395.883213 events/s
Ran test for 46.127213 seconds
[diego.santos@pi057109 build]$
```

This is the throughput.

This is NOT the throughput. Though you can also look at how does it change

Allen lab, part II

+ Edit the filter (LambdaFilter.cu) and the monitoring line (LambdasLine.cu)

\<bla\>AllenSt1(2)/device/LAMBDAS/lambda_filter/src/LambdaFilter.cu
\<bla\>AllenSt1(2)/device/selections/lines/inclusive_hadron/src/LambdasLine.cu

+ in build: rm -rf *, then **Compile w/ ROOT**

+ ./Allen **-m 1000 -t 10** --device 0 --mdf \`ls -d ../../DATA/*mdf | xargs echo | sed 's/ /,/g'\`

+ **Don't run repetitions here, just the full dataset**

+ Inspect the output   (build/output/*root)

Quick look at LambdaFilter.cu

```cpp
/**************************************************************************\
* (c) Copyright 2018-2020 CERN for the benefit of the LHCb Collaboration     *
\**************************************************************************/
#include "LambdaFilter.cuh"

void LambdaFilter::lambda_filter_t::set_arguments_size(
  ArgumentReferences<Parameters> arguments,
  const RuntimeOptions&,
  const Constants&,
  const HostBuffers&) const
{ //output arrays, mf means nothing, just historical
  set_size<dev_event_list_mf_t>(arguments, first<host_number_of_events_t>(arguments));
  set_size<dev_mask_mf_t>(arguments, first<host_number_of_events_t>(arguments)); //another event list, copy of the 1 above
  set_size<host_event_list_mf_t>(arguments, first<host_number_of_events_t>(arguments)); //event list declared in host
  set_size<dev_selected_events_mf_t>(arguments, 1); //counter (size is 1)
  set_size<host_selected_events_mf_t>(arguments, 1);
  set_size<dev_mf_decisions_t>(arguments, first<host_number_of_events_t>(arguments)); // array of bools, like real 'mask'
  set_size<dev_mf_track_atomics_t>(arguments, first<host_number_of_reconstructed_ut_tracks_t>(arguments)); //maybe we can remove it, but it is in many places
}

void LambdaFilter::lambda_filter_t::operator()(
  const ArgumentReferences<Parameters>& arguments,
  const RuntimeOptions& runtime_options,
  const Constants&,
  HostBuffers& host_buffers,
  const Allen::Context& context) const //sync different streams, eg, several instances of Allen running
{
  initialize<dev_event_list_mf_t>(arguments, 0, context);
  initialize<dev_selected_events_mf_t>(arguments, 0, context);
  initialize<dev_mf_decisions_t>(arguments, 0, context);
  initialize<dev_mf_track_atomics_t>(arguments, 0, context);
  initialize<host_selected_events_mf_t>(arguments, 0, context);

  global_function(lambda_filter)(dim3(size<dev_input_event_list_t>(arguments)), property<block_dim_t>(), context)(
    arguments); //runs the filter (declared later) !!
  Allen::copy<host_selected_events_mf_t, dev_selected_events_mf_t>(arguments, context);//copies 2nd argument over the first argument, i.e, list over mask
  Allen::copy<host_event_list_mf_t, dev_event_list_mf_t>(arguments, context);

  Allen::copy<dev_mask_mf_t, dev_event_list_mf_t>(arguments, context);

  Allen::synchronize(context);
  reduce_size<host_event_list_mf_t>(arguments, first<host_selected_events_mf_t>(arguments)); //free memory
  reduce_size<dev_event_list_mf_t>(arguments, first<host_selected_events_mf_t>(arguments));
  if (runtime_options.do_check) {
    assign_to_host_buffer<dev_selected_events_mf_t>(host_buffers.host_selected_events_mf, arguments, context);
    assign_to_host_buffer<dev_event_list_mf_t>(host_buffers.host_event_list_mf, arguments, context);
    unsigned* host_number_of_selected_events = &host_buffers.host_number_of_selected_events;
    assign_to_host_buffer<dev_event_list_mf_t>(host_buffers.host_event_list, arguments, context);
  }
}
```

```cuda
51    }
52
53    __global__ void LambdaFilter::lambda_filter(LambdaFilter::Parameters parameters)
54    {
55      const unsigned number_of_events = parameters.dev_number_of_events[0];
56      const unsigned event_number = parameters.dev_input_event_list[blockIdx.x];
57
58      SciFi::Consolidated::ConstTracks scifi_tracks {parameters.dev_atomics_scifi, //number of tracks. Consolidated SciTrack == (n) LoW Tracks here
59                                                     parameters.dev_scifi_track_hit_number,
60                                                     parameters.dev_scifi_qop,
61                                                     parameters.dev_scifi_states, //state of SciFi
62                                                     parameters.dev_scifi_track_ut_indices,
63                                                     event_number,
64                                                     number_of_events};
65
66      const unsigned event_offset = scifi_tracks.tracks_offset(event_number);
67      const unsigned number_of_tracks_event = scifi_tracks.number_of_tracks(event_number);
68      unsigned* event_mf_decision = parameters.dev_mf_decisions.get() + event_number; //not a bool cs u can't do atomic OR w/ Bools.
69
70      const unsigned sv_offset = parameters.dev_sv_offsets[event_number];
71      const unsigned n_svs = parameters.dev_sv_offsets[event_number + 1] - sv_offset;
72
73      //Fitted tracks
74      const ParKalmanFilter::FittedTrack* event_tracks = parameters.dev_kf_tracks + event_offset; //Kalman Filter
75      // Secondary vertices.
76      const VertexFit::TrackMVAVertex* event_secondary_vertices = parameters.dev_consolidated_svs + sv_offset; //Two Prong vertices
77
78      for (unsigned i_sv = threadIdx.x; i_sv < n_svs; i_sv += blockDim.x) {
79        auto vertex = event_secondary_vertices[i_sv];
80
81        float alpha = -1;
82        bool alpha_sign = alpha < 0;
83        if (parameters.alpha_positive==1) alpha_sign = alpha > 0; // config the filter if u want lambdas or anti-lambdas
84        bool dec = 0;
85
86        if (dec) {
87          atomicOr(event_mf_decision, dec);
88        }
89      }
90
91      __syncthreads();
92      unsigned* dev_selected_events_mf = parameters.dev_selected_events_mf.get();
93      // Passed cut.
94      if (threadIdx.x == 0 && event_mf_decision[0]) {
95        const auto selected_event = atomicAdd(dev_selected_events_mf, 1);
96        parameters.dev_event_list_mf[selected_event] = event_number;
97      }
98    }
```

Here your filter is declared (as __global__)

```cpp
51    }
52
53    __global__ void LambdaFilter::lambda_filter(LambdaFilter::Parameters parameters)
54    {
55      const unsigned number_of_events = parameters.dev_number_of_events[0];
56      const unsigned event_number = parameters.dev_input_event_list[blockIdx.x];
57
58      SciFi::Consolidated::ConstTracks scifi_tracks {parameters.dev_atomics_scifi, //number of tracks. Consolidated SciFiTracks == (pre) LongTracks here
59                                                      parameters.dev_scifi_track_hit_number,
60                                                      parameters.dev_scifi_qop,
61                                                      parameters.dev_scifi_states, //state in scifi
62                                                      parameters.dev_scifi_track_ut_indices,
63                                                      event_number,
64                                                      number_of_events};
65
66      const unsigned event_offset = scifi_tracks.tracks_offset(event_number);
67      const unsigned number_of_tracks_event = scifi_tracks.number_of_tracks(event_number);
68      unsigned* event_mf_decision = parameters.dev_mf_decisions.get() + event_number; //not a bool cs u can't do atomic OR // Bools
69
70      const unsigned sv_offset = parameters.dev_sv_offsets[event_number];
71      const unsigned n_svs = parameters.dev_sv_offsets[event_number + 1] - sv_offset;
72
73      //Fitted tracks
74      const ParKalmanFilter::FittedTrack* event_tracks = parameters.dev_kf_tracks + event_offset; //Kalman Filter
75      // Secondary vertices.
76      const VertexFit::TrackMVAVertex* event_secondary_vertices = parameters.dev_consolidated_svs + sv_offset; //Two Prong vertices
77
78      for (unsigned i_sv = threadIdx.x; i_sv < n_svs; i_sv += blockDim.x) {
79        auto vertex = event_secondary_vertices[i_sv];
80
81        float alpha = -1;
82        bool alpha_sign = alpha < 0;
83        if (parameters.alpha_positive==1) alpha_sign = alpha > 0; // config the filter if u want lambdas or anti-lambdas
84        bool dec = 0;
85
86        if (dec) {
87          atomicOr(event_mf_decision, dec);
88        }
89      }
90
91      __syncthreads();
92      unsigned* dev_selected_events_mf = parameters.dev_selected_events_mf.get();
93      // Passed cut.
94      if (threadIdx.x == 0 && event_mf_decision[0]) {
95        const auto selected_event = atomicAdd(dev_selected_events_mf, 1);
96        parameters.dev_event_list_mf[selected_event] = event_number;
97      }
98    }
```

We are just using the SciFi tracks OR to get some offset nos.

```cuda
51    }
52
53    __global__ void LambdaFilter::lambda_filter(LambdaFilter::Parameters parameters)
54    {
55      const unsigned number_of_events = parameters.dev_number_of_events[0];
56      const unsigned event_number = parameters.dev_input_event_list[blockIdx.x];
57
58      SciFi::Consolidated::ConstTracks scifi_tracks {parameters.dev_atomics_scifi, //number of tracks. Consolidated SciFiTracks == (pre) LongTracks here
59                                                     parameters.dev_scifi_track_hit_number,
60                                                     parameters.dev_scifi_qop,
61                                                     parameters.dev_scifi_states, //state in scifi
62                                                     parameters.dev_scifi_track_ut_indices,
63                                                     event_number,
64                                                     number_of_events};
65
66      const unsigned event_offset = scifi_tracks.tracks_offset(event_number);
67      const unsigned number_of_tracks_event = scifi_tracks.number_of_tracks(event_number);
68      unsigned* event_mf_decision = parameters.dev_mf_decisions.get() + event_number; //not a bool cuz can't do atomic OR w/ Bools.
69
70      const unsigned sv_offset = parameters.dev_sv_offsets[event_number];
71      const unsigned n_svs = parameters.dev_sv_offsets[event_number + 1] - sv_offset;
72
73      //Fitted tracks
74      const ParKalmanFilter::FittedTrack* event_tracks = parameters.dev_kf_tracks + event_offset; //Kalman Filter
75      // Secondary vertices.
76      const VertexFit::TrackMVAVertex* event_secondary_vertices = parameters.dev_consolidated_svs + sv_offset; //Two Prong vertices
77
78      for (unsigned i_sv = threadIdx.x; i_sv < n_svs; i_sv += blockDim.x) {
79        auto vertex = event_secondary_vertices[i_sv];
80
81        float alpha = -1;
82        bool alpha_sign = alpha < 0;
83        if (parameters.alpha_positive==1) alpha_sign = alpha > 0; // config the filter if u want lambdas or anti-lambdas
84        bool dec = 0;
85
86        if (dec) {
87          atomicOr(event_mf_decision, dec);
88        }
89      }
90
91      __syncthreads();
92      unsigned* dev_selected_events_mf = parameters.dev_selected_events_mf.get();
93      // Passed cut.
94      if (threadIdx.x == 0 && event_mf_decision[0]) {
95        const auto selected_event = atomicAdd(dev_selected_events_mf, 1);
96        parameters.dev_event_list_mf[selected_event] = event_number;
97      }
98    }
```

The actual tracks your candidates are made of

Your V0 candidates

⚠ Unable to watch for file changes in this large workspace folder. Please follow the instructions link to resolve this issue.

Instructions

AllenSt1 > device > LAMBDAS > lambda_filter > src > LambdaFilter.cu

```cuda
51  }
52
53  __global__ void LambdaFilter::lambda_filter(LambdaFilter::Parameters parameters)
54  {
55    const unsigned number_of_events = parameters.dev_number_of_events[0];
56    const unsigned event_number = parameters.dev_input_event_list[blockIdx.x];
57
58    SciFi::Consolidated::ConstTracks scifi_tracks {parameters.dev_atomics_scifi, //number of tracks. Consolidated SciFiTracks == (pre) LongTracks here
59                                                   parameters.dev_scifi_track_hit_number,
60                                                   parameters.dev_scifi_qop,
61                                                   parameters.dev_scifi_states, //state in scifi
62                                                   parameters.dev_scifi_track_ut_indices,
63                                                   event_number,
64                                                   number_of_events};
65
66    const unsigned event_offset = scifi_tracks.tracks_offset(event_number);
67    const unsigned number_of_tracks_event = scifi_tracks.number_of_tracks(event_number);
68    unsigned* event_mf_decision = parameters.dev_mf_decisions.get() + event_number; //not a bool cs u can't do atomic OR w/ Bools.
69
70    const unsigned sv_offset = parameters.dev_sv_offsets[event_number];
71    const unsigned n_svs = parameters.dev_sv_offsets[event_number + 1] - sv_offset;
72
73    //Fitted tracks
74    const ParKalmanFilter::FittedTrack* event_tracks = parameters.dev_kf_tracks + event_offset; //Kalman Filter
75    // Secondary vertices.
76    const VertexFit::TrackMVAVertex* event_secondary_vertices = parameters.dev_consolidated_svs + sv_offset; //Two Prong vertices
77
78    for (unsigned i_sv = threadIdx.x; i_sv < n_svs; i_sv += blockDim.x) {
79      auto vertex = event_secondary_vertices[i_sv];
80
81      float alpha = -1;
82      bool alpha_sign = alpha < 0;
83      if (parameters.alpha_positive==1) alpha_sign = alpha > 0; // config the filter if u want lambdas or anti-lambdas
84      bool dec = 0;
85
86      if (dec) {
87        atomicOr(event_mf_decision, dec);
88      }
89    }
90
91    __syncthreads();
92    unsigned* dev_selected_events_mf = parameters.dev_selected_events_mf.get();
93    // Passed cut.
94    if (threadIdx.x == 0 && event_mf_decision[0]) {
95      const auto selected_event = atomicAdd(dev_selected_events_mf, 1);
96      parameters.dev_event_list_mf[selected_event] = event_number;
97    }
98  }
```

Your selection criteria go here

Quick look at LambdasLine.cu

```cuda
/**************************************************************************\
* (c) Copyright 2020 CERN for the benefit of the LHCb Collaboration        *
\**************************************************************************/
#include "LambdasLine.cuh"
#include <ROOTHeaders.h>
#include "ROOTService.h"
INSTANTIATE_LINE(lambda_line::lambda_line_t, lambda_line::Parameters)

void lambda_line::lambda_line_t::set_arguments_size(
  ArgumentReferences<Parameters> arguments,
  const RuntimeOptions&,
  const Constants&,
  const HostBuffers&) const
{
  set_size<typename Parameters::dev_decisions_t>(
    arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));
  set_size<typename Parameters::dev_decisions_offsets_t>(
    arguments, first<typename Parameters::host_number_of_events_t>(arguments));
  set_size<typename Parameters::host_post_scaler_t>(arguments, 1);
  set_size<typename Parameters::host_post_scaler_hash_t>(arguments, 1);

  set_size<typename Parameters::dev_sv_masses_t>(
    arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));
  set_size<typename Parameters::host_sv_masses_t>(
    arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));

  set_size<typename Parameters::dev_sv_masses_lambda_t>(
    arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));
  set_size<typename Parameters::host_sv_masses_lambda_t>(
    arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));

  set_size<typename Parameters::dev_pt_t>(arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));
  set_size<typename Parameters::host_pt_t>(arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));

  set_size<typename Parameters::dev_alpha_t>(arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));
  set_size<typename Parameters::host_alpha_t>(arguments, lambda_line::lambda_line_t::get_decisions_size(arguments));


}
  __device__ bool lambda_line::lambda_line_t::select(
  const Parameters&,
  std::tuple<const VertexFit::TrackMVAVertex&> input)
{
  const auto& vertex = std::get<0>(input);
  return 0;
}

#ifdef WITH_ROOT
```

The arrays in which you will add information from the event, to be sent later to the ROOT ntuple

This says which candidates do you want to save (so the same selection as the filter)

Activities — Visual Studio Code — Sat 14:27 — en

File   Edit   Selection   View   Go   Run   Terminal   Help

EXPLORER

∨ ALLEN_INFIERI_21 [SSH: 172.16.57.91]
 > AllenBackup
 > AllenSt1
 > AllenSt2
 > DATA
 ∨ INITS
   • LambdaFilter.cu
   • LambdasLine.cu
   ≡ LambdasLine.cu~
 ∨ SOLS
   • LambdaFilter.cu
   • LambdasLine.cu
 ≡ rebootclass.csh
 ≡ usefulcommands.crap

Tabs: LambdasLine.cu INITS × | LambdasLine.cu SOLS

INITS > • LambdasLine.cu

```cpp
57    }
58
59    __device__ void lambda_line::lambda_line_t::monitor(
60      const Parameters& parameters,
61      std::tuple<const VertexFit::TrackMVAVertex&> input,
62      unsigned index,
63      bool sel)
64    {
65      const unsigned event_number = parameters.dev_event_list[blockIdx.x];
66      const unsigned number_of_events = parameters.dev_number_of_events[0];
67
68
69      const auto& vertex = std::get<0>(input);
70
71      if (sel) {
72        // Consolidated SciFi tracks.
73        SciFi::Consolidated::ConstTracks scifi_tracks {parameters.dev_atomics_scifi,
74          parameters.dev_scifi_track_hit_number,
75          parameters.dev_scifi_qop,
76          parameters.dev_scifi_states,
77          parameters.dev_scifi_track_ut_indices,
78          event_number,
79          number_of_events};
80
81        float pti = 0.;
82        parameters.dev_pt[index] = pti;
83      }
84    }
85    |
86    void lambda_line::lambda_line_t::output_monitor(
87      const ArgumentReferences<Parameters>& arguments,
88      const RuntimeOptions& runtime_options,
89      const Allen::Context& context) const
90    {
91
92      auto name_str = name();
93      std::string name_ttree = "monitor_tree" + name_str;
94      Allen::copy<host_sv_masses_t, dev_sv_masses_t>(arguments, context);
95      Allen::copy<host_pt_t, dev_pt_t>(arguments, context);
96      Allen::synchronize(context);
97
98      auto handler = runtime_options.root_service->handle();
99      handler.file("monitorLambda.root");
100
101      auto tree = handler.ttree(name_ttree.c_str());
102
103      float pt;
104
```

Make sure you feed here the arrays w/ the information you'll want to add later to the ROOT file
(Right now just adds zeroes)

⚠ Unable to watch for file changes in this large workspace folder. Please follow the instructions link to resolve this issue.

[Instructions]

SSH: 172.16.57.91   infieri*+   ⊗0 ⚠0   Ln 85, Col 1   Spaces: 2   UTF-8   LF   CUDA C++

```cuda
80
81
82        }
83   }
84
85   void lambda_line::lambda_line_t::output_monitor(
86       const ArgumentReferences<Parameters>& arguments,
87       const RuntimeOptions& runtime_options,
88       const Allen::Context& context) const
89   {
90
91       auto name_str = name();
92       std::string name_ttree = "monitor_tree" + name_str;
93       Allen::copy<host_sv_masses_t, dev_sv_masses_t>(arguments, context);
94       Allen::copy<host_pt_t, dev_pt_t>(arguments, context);
95       Allen::synchronize(context);
96
97       auto handler = runtime_options.root_service->handle();
98       handler.file("monitorLambda.root");
99
100      auto tree = handler.ttree(name_ttree.c_str());
101
102      float pt;
103
104
105      handler.branch("pt", pt);
106
107
108      unsigned n_svs = size<host_sv_masses_t>(arguments);
109      float* sv_pt;
110      int i0 = tree->GetEntries();
111      for (unsigned i = 0; i < n_svs; i++) {
112          sv_pt = data<host_pt_t>(arguments) + i;
113
114          pt = sv_pt[0];
115          tree->Fill();
116      }
117      tree->Write(0, TObject::kOverwrite);
118  }
119  #endif
120
```

Tell Allen what to save to ROOT file

Loop over the candidates and save info to the TTree

Allen lab, part II

π

pl'   pt'

IP

K
s

π

$IP\chi2 \sim (IP/\sigma_{IP})^2$
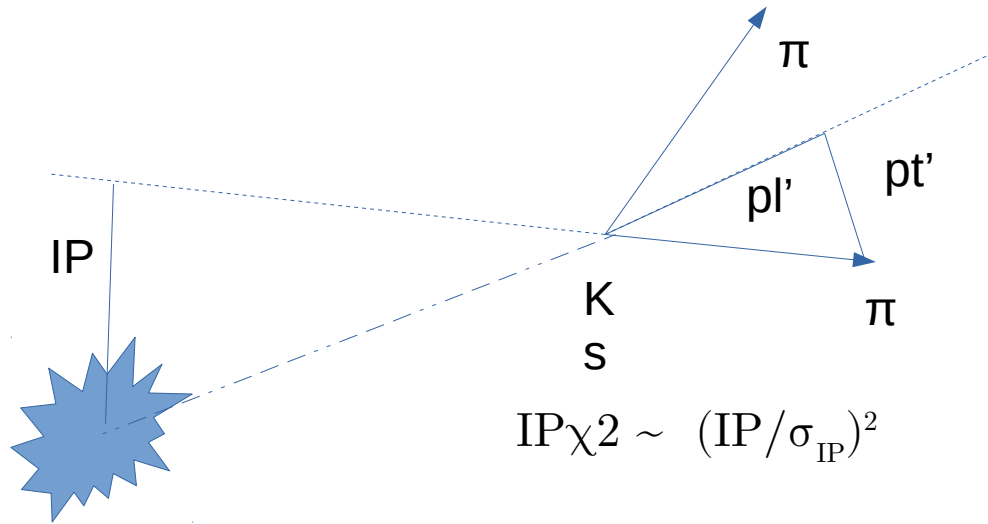
Some useful variables for selecting V0's (and/or to monitor in your ROOT file)

* Minimum $IP\chi2$ of the pions
* $\chi2$ of the decay vertex fit
* Armenteros variables : (these you have to compute from the Kalman tracks)

Some examples implementing a filter and/or instantiating vertices:

https://gitlab.cern.ch/lhcb/Allen/-/blob/master/device/muon/muon_filter/src/MuonFilter.cu
https://gitlab.cern.ch/lhcb/Allen/-/blob/master/device/vertex_fit/vertex_fitter/src/VertexFitter.cu#L66
https://gitlab.cern.ch/lhcb/Allen/-/blob/master/device/selections/lines/muon/src/DiMuonSoftLine.cu

Backup

Setup your session

Usernames are infieri1, infieri2, and infieri3. To connect to the machines:

ssh -X infieri1(2,3)@mastercr1.igfae.usc.es
ssh -X infieri1(2)@172.16.57.91   (or ssh -X infieri3@172.16.57.223)

        (for MAC users is ssh – Y)

source /cvmfs/sft.cern.ch/lcg/views/setupViews.sh LCG_99 x86_64-centos7-clang10-opt
source /cvmfs/sft.cern.ch/lcg/contrib/cuda/11.2/x86_64-centos7/setup.sh

You have these lines (plus other useful commands) in the file 'usefulcommands' just under ALLEN_INFIERI_21. For copypasting purposes

Paths and devices

+ infieri1: /ALLEN_INFIERI_21/AllenSt1/   uses --device 0

+ infieri2: /ALLEN_INFIERI_21/AllenSt2/   uses --device 1

+ infieri3: /scratch/ALLEN_INFIERI_21/AllenSt1/ uses --device 0   (or 1, doesn't matter)

Allen lab, part I compilation flags

Inside <bla>/build/ directory:

cmake -DSTANDALONE=ON -DTARGET_DEVICE=CUDA -
DCMAKE_BUILD_TYPE=Release -DSEQUENCE=hlt1_pp_default -DUSE_ROOT=OFF ..
&& make -j50

Students 1 and 2: -j50 or -j60 or so

Switch OFF ROOT

Student 3 : -j20 or so   (CPU is worse in this machine)

Allen lab, part II compilation flags

cmake -DSTANDALONE=ON -DTARGET_DEVICE=CUDA -
DCMAKE_BUILD_TYPE=Release -DSEQUENCE=hlt1_pp_default -DUSE_ROOT=ON ..
&& make -j50

Students 1 and 2: -j50 or -j60 or so

Student 3 : -j20 or so   (CPU is worse in this machine)

Switch ON ROOT

A look at Allen printout

configurable algorithm constants (--configuration): Sequence.json

number of events to process (-n, --number-of-events): 500
number of input slices to allocate (-s, --number-of-slices): 0
number of events per slice (--events-per-slice): 1000
number of threads / streams (-t, --threads): 16
number of repetitions per thread / stream (-r, --repetitions): 1000
memory to reserve on the device per thread / stream (megabytes) (-m, --memory): 500
memory to reserve on the host per thread / stream (megabytes) (--host-memory): 200
verbosity [0-5] (-v, --verbosity): 3
offload part of the computation to CPU (--cpu-offload): 1
Write selected event to output file (--output-file):
select device to use (--device): 0, NVIDIA GeForce RTX 3090