Profiling and data access patterns in Allen

Douglas Li

Stanford University

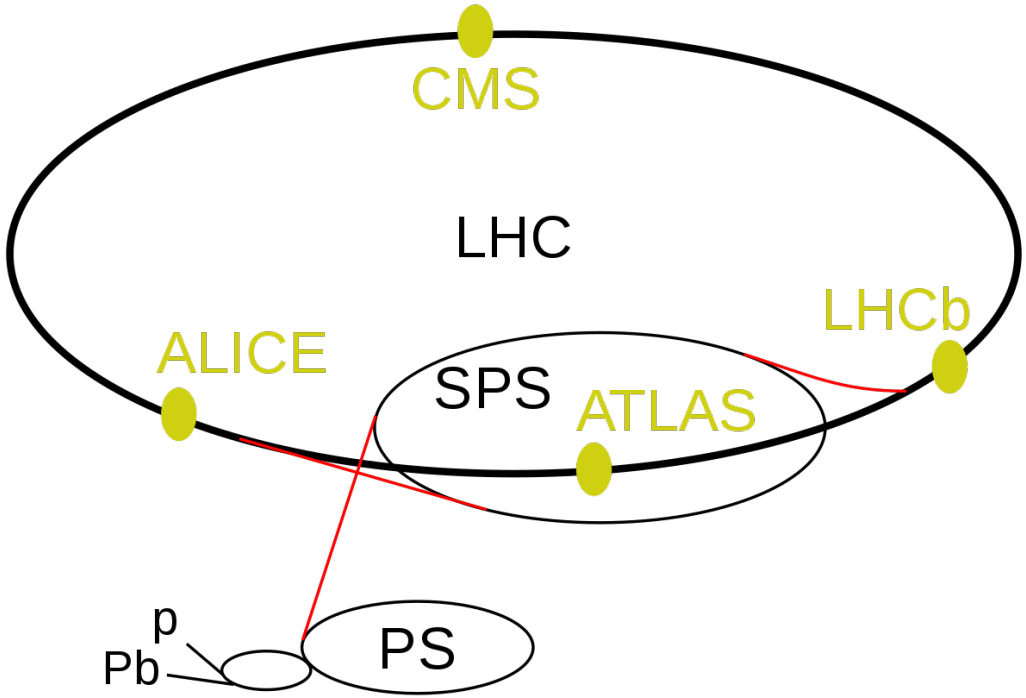
October 2021

Table of Contents

Table of Contents

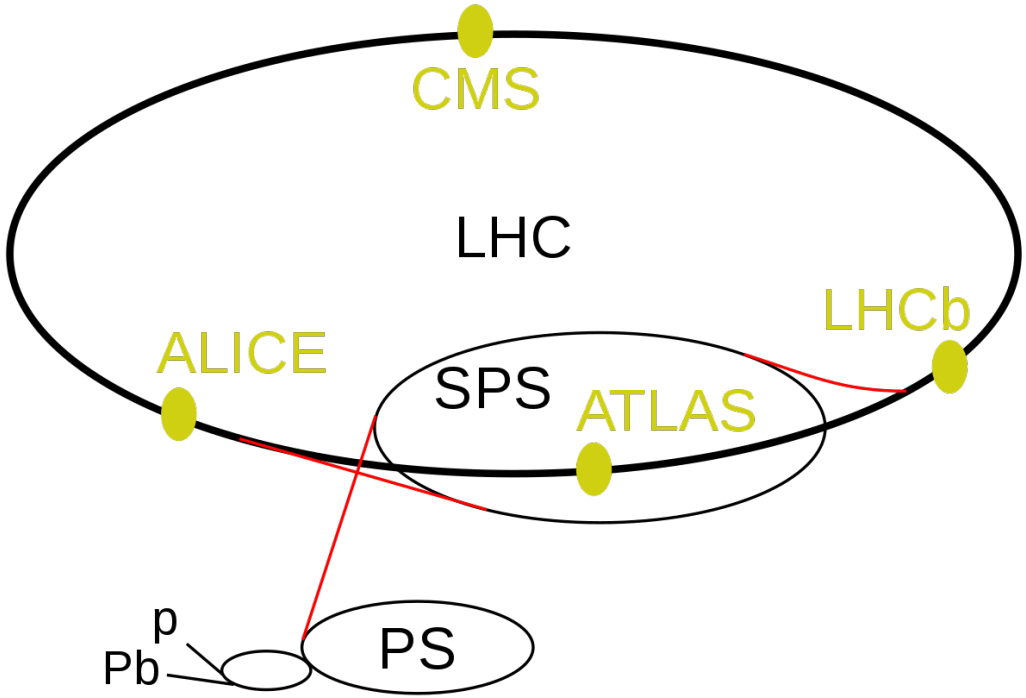
LHCb detector

- LHCb is one of four large detectors at the Large Hadron Collider (LHC)



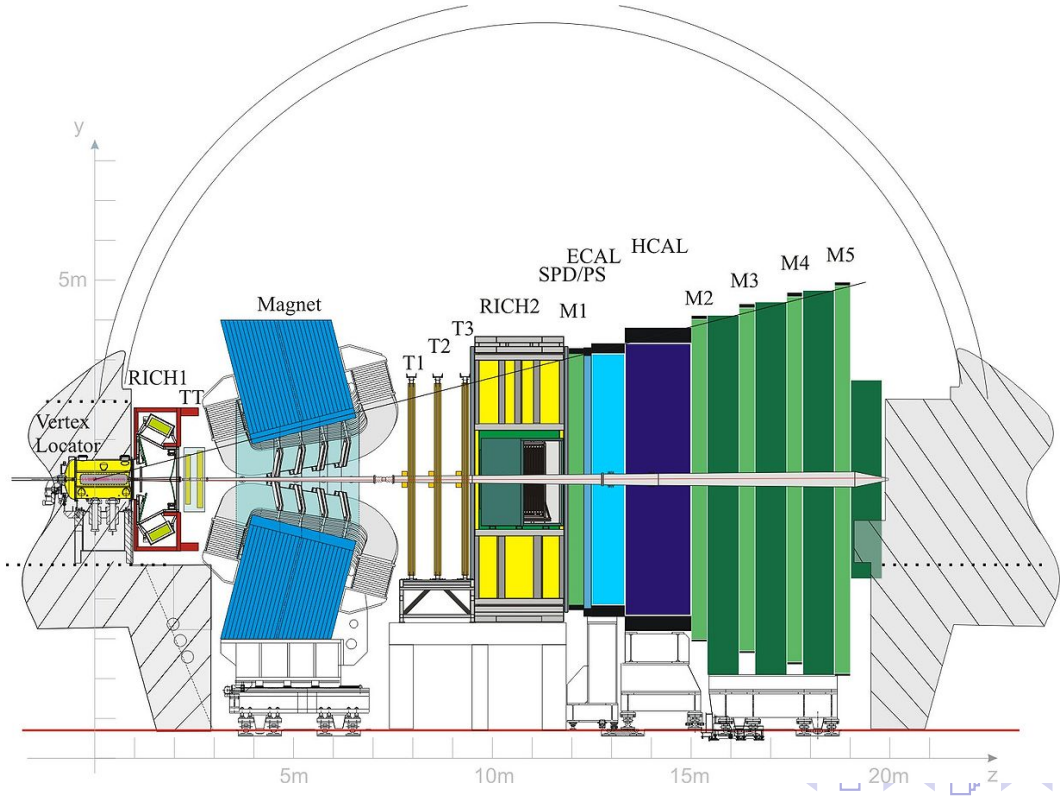
LHCb detector

- LHCb is one of four large detectors at the Large Hadron Collider (LHC)
- The LHC collides high energy beams of protons into each other to study the particles produced from such collisions.



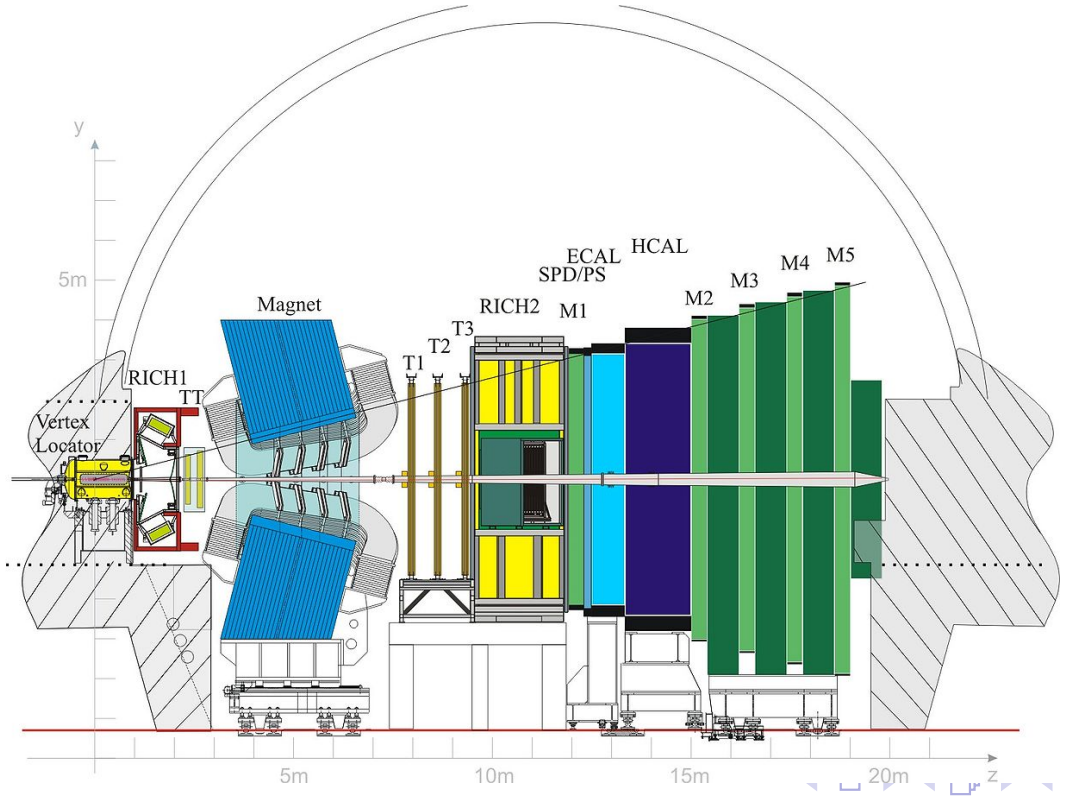
LHCb upgrades

- The LHCb is currently shut down for upgrades before Run 3 of data taking. Importantly, Run 3 will not use a hardware trigger.



LHCb upgrades

- The LHCb is currently shut down for upgrades before Run 3 of data taking. Importantly, Run 3 will not use a hardware trigger.
- Only a small fraction of the data read out from subdetectors can be kept for study: the trigger selects which events to keep.



LHCb upgrades

- The LHCb is currently shut down for upgrades before Run 3 of data taking. Importantly, Run 3 will not use a hardware trigger.
- Only a small fraction of the data read out from subdetectors can be kept for study: the trigger selects which events to keep.
- Without a hardware trigger, the software trigger must be redesigned to handle the full event rate of 30 MHz.

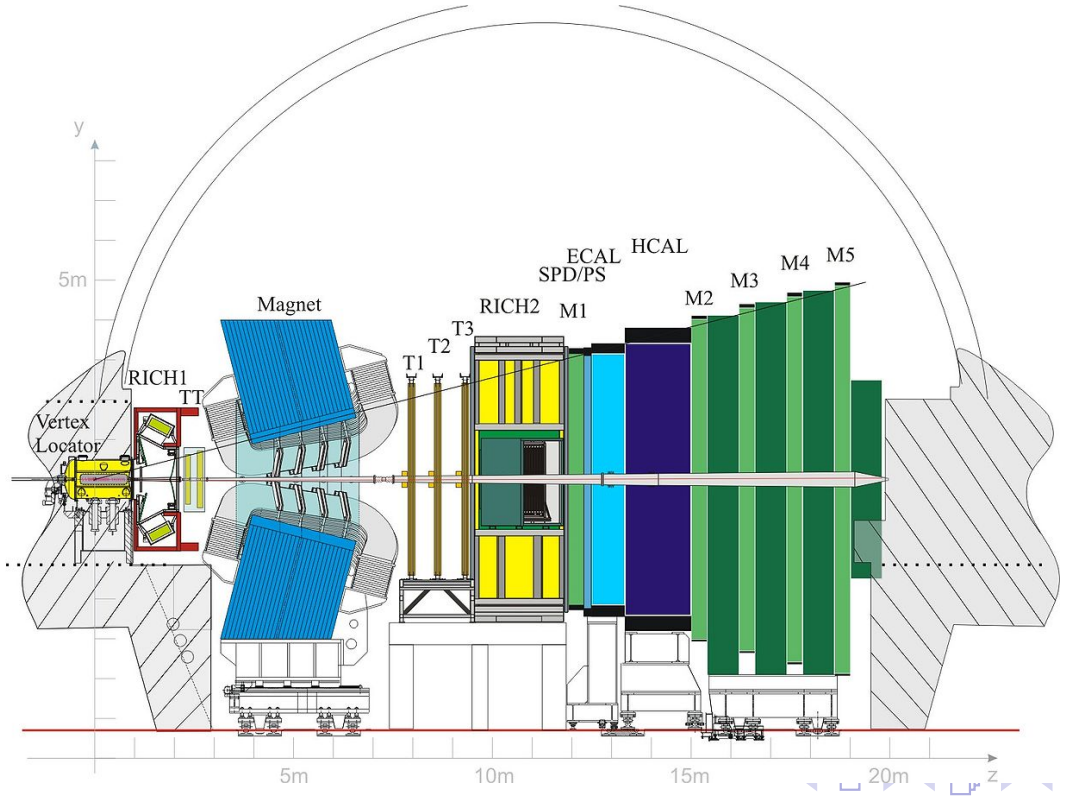
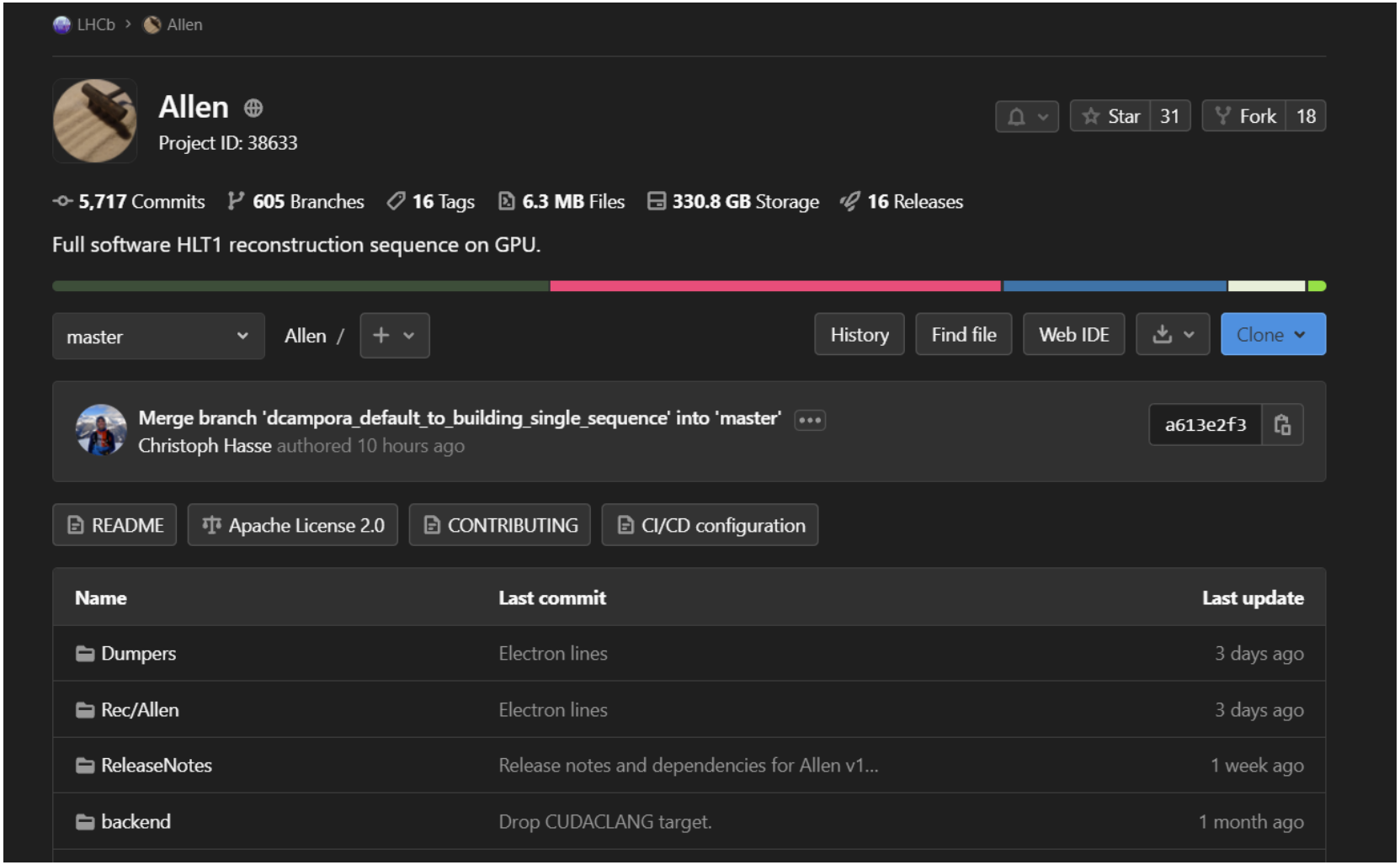


Table of Contents

What is Allen?

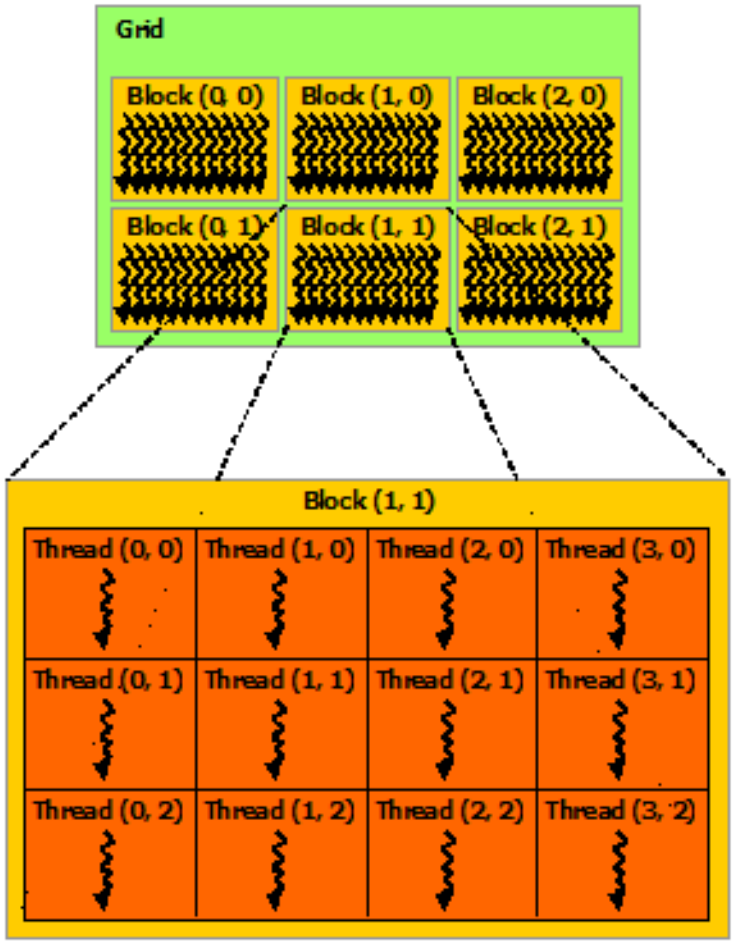


What is Allen?

- "HLT1 reconstruction sequence": Stage 1 of software trigger. A sequence of algorithms that does the following tasks:
  - "Hits" are identified in each of the VELO, UT, and SciFi detectors.
  - (Each detector has several layers, and as particles pass through the layers, they leave signals in each layer. These signals are hits.)
  - These hits are combined to form tracks (trajectories of a particle).
  - The tracks are then used to reconstruct *primary vertices* (p-p collision points) and *secondary vertices* (decay points)
- "GPUs": Graphical Processing Units

The layout of a GPU

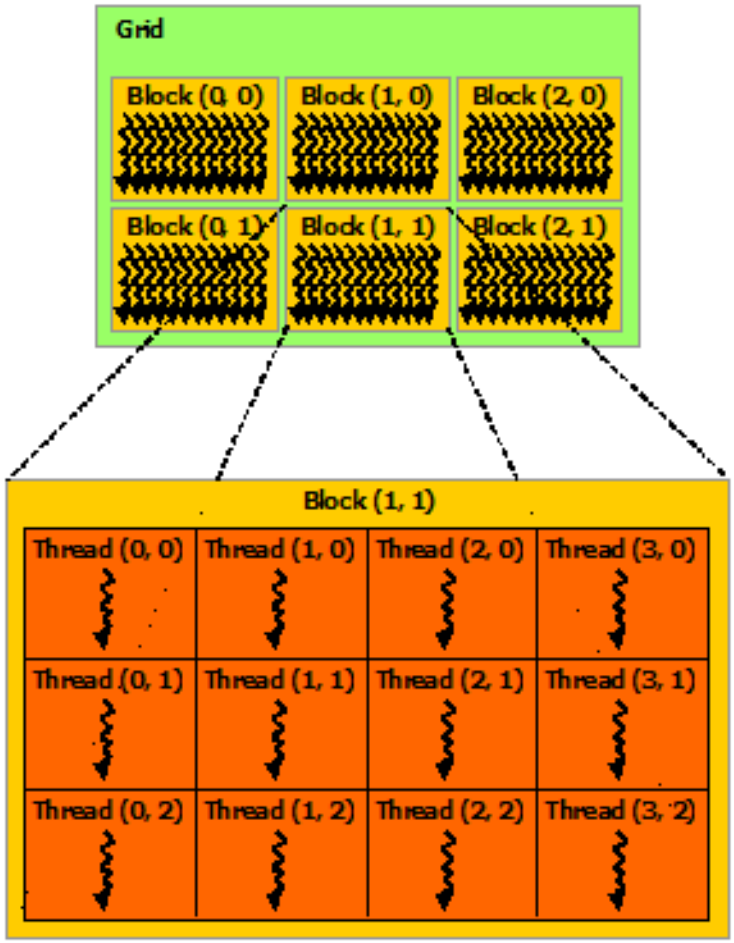
- Grid-block-thread hierarchy.



CUDA C++ Programming Guide

The layout of a GPU

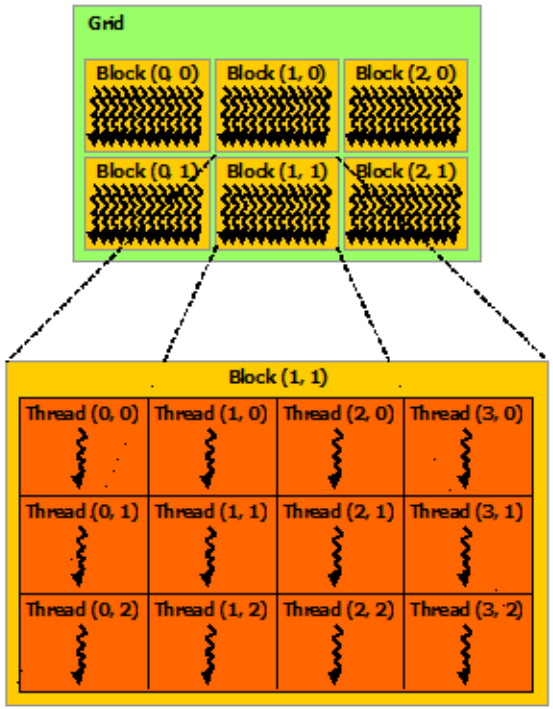
- Grid-block-thread hierarchy.
- In CUDA, functions are executed N times by N different threads. Threads are grouped into thread blocks which are grouped into grids.



CUDA C++ Programming Guide

The layout of a GPU

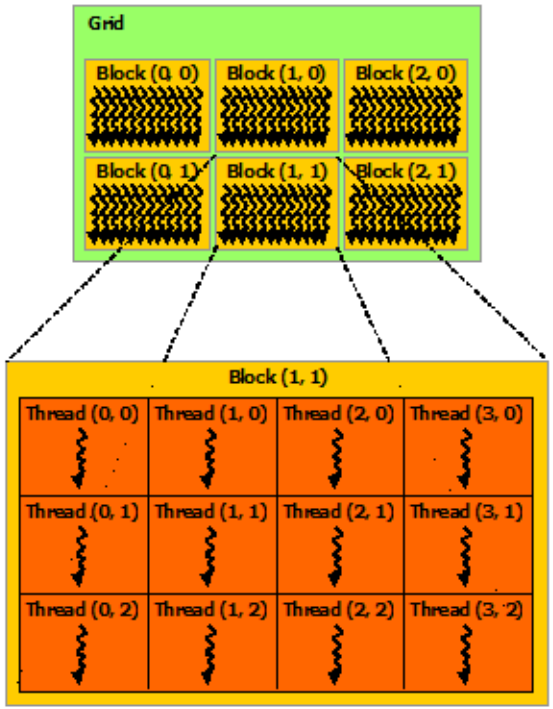
- We can take advantage of this hierarchy as follows. Events can be handled by thread blocks, and individual data in each event (hits/tracks) can be handled by threads.



CUDA C++ Programming Guide

The layout of a GPU

- We can take advantage of this hierarchy as follows. Events can be handled by thread blocks, and individual data in each event (hits/tracks) can be handled by threads.
- So each event can be processed independently, and the hits/tracks within that event can also be processed in parallel.

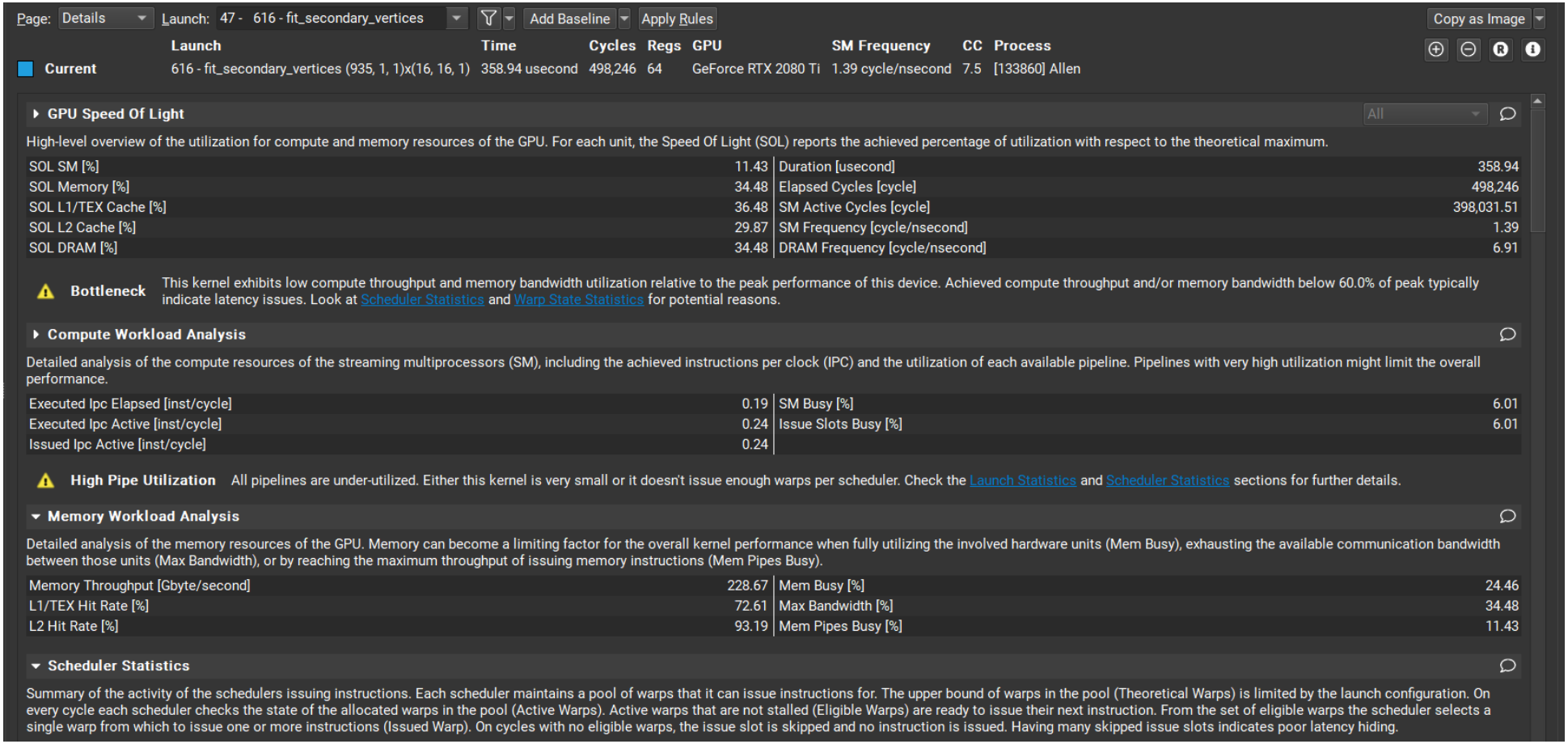


CUDA C++ Programming Guide

Table of Contents

Profiling Allen

- We can measure the performance of each algorithm using the profiling tool NVIDIA Nsight Compute.



Page: Details | Launch: 47 - 616 - fit_secondary_vertices | Add Baseline | Apply Rules | Copy as Image

	Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	616 - fit_secondary_vertices (935, 1, 1)x(16, 16, 1)	358.94 usecond	498,246	64	GeForce RTX 2080 Ti	1.39 cycle/nsecond	7.5	[133860] Allen

GPU Speed Of Light

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum.

SOL SM [%]	11.43	Duration [usecond]	358.94
SOL Memory [%]	34.48	Elapsed Cycles [cycle]	498,246
SOL L1/TEX Cache [%]	36.48	SM Active Cycles [cycle]	398,031.51
SOL L2 Cache [%]	29.87	SM Frequency [cycle/nsecond]	1.39
SOL DRAM [%]	34.48	DRAM Frequency [cycle/nsecond]	6.91

Bottleneck This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at Scheduler Statistics and Warp State Statistics for potential reasons.

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	0.19	SM Busy [%]	6.01
Executed Ipc Active [inst/cycle]	0.24	Issue Slots Busy [%]	6.01
Issued Ipc Active [inst/cycle]	0.24		

High Pipe Utilization All pipelines are under-utilized. Either this kernel is very small or it doesn't issue enough warps per scheduler. Check the Launch Statistics and Scheduler Statistics sections for further details.

Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy).

Memory Throughput [Gbyte/second]	228.67	Mem Busy [%]	24.46
L1/TEX Hit Rate [%]	72.61	Max Bandwidth [%]	34.48
L2 Hit Rate [%]	93.19	Mem Pipes Busy [%]	11.43

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Warp Stalls

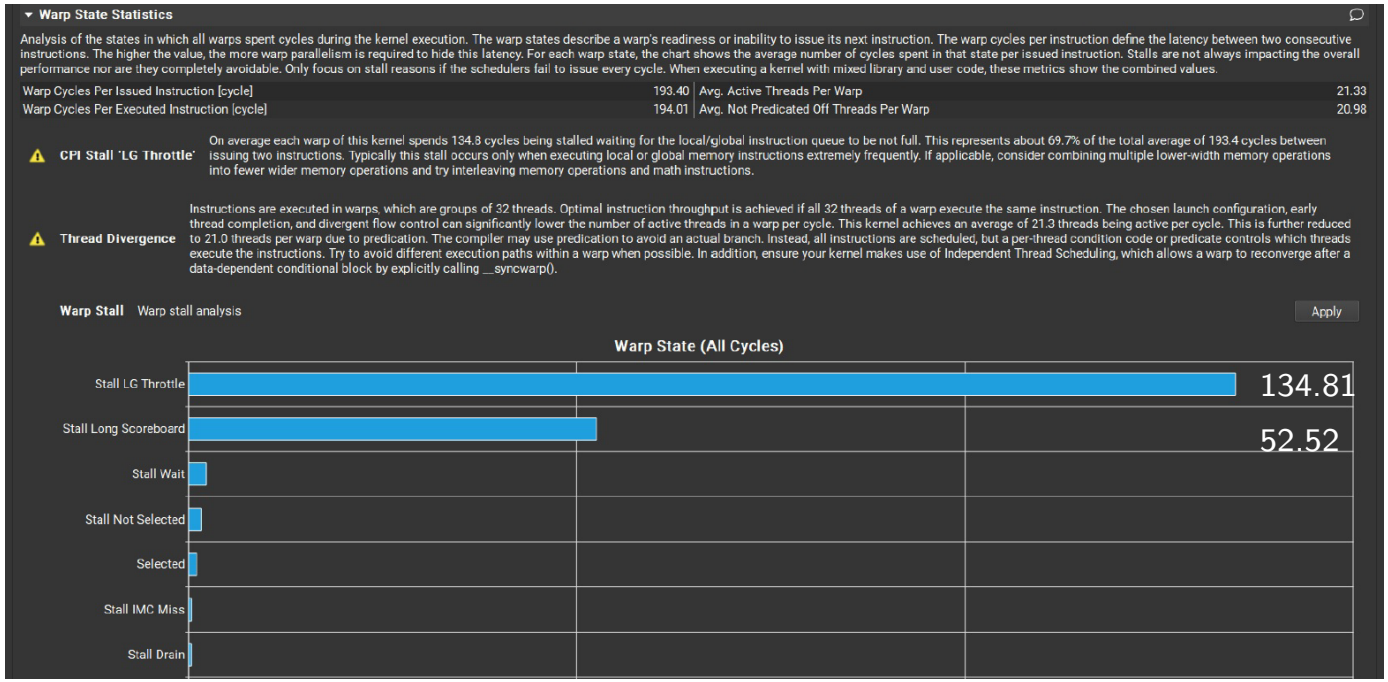
- The hardware partitions each thread block into *warps*, which are groups of 32 threads.

Warp Stalls

- The hardware partitions each thread block into *warps*, which are groups of 32 threads.
- A *warp scheduler* schedules warps for execution, but can stall for a variety of reasons.

Warp Stalls

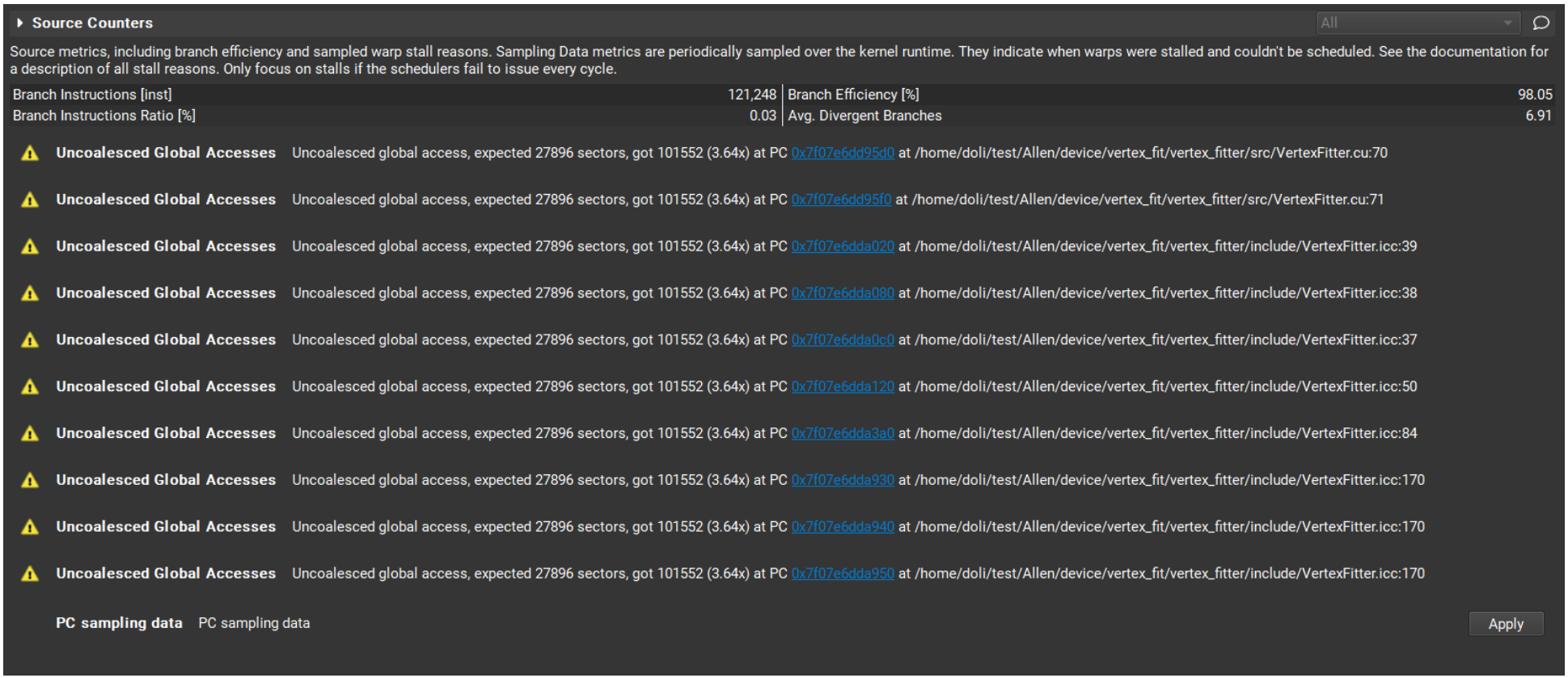
Profiler indicates that latency is a limiting factor to performance in e.g. fit_secondary_vertices algorithm.



The x-axis measures the number of cycles per instruction a warp spends in a given stall state. We see that the LG Throttle stall, with 134.81 cycles/instruction, is the bottleneck, and that this is related to frequent accesses to local or global memory. So this hints that we may want to look at how our code accesses memory.

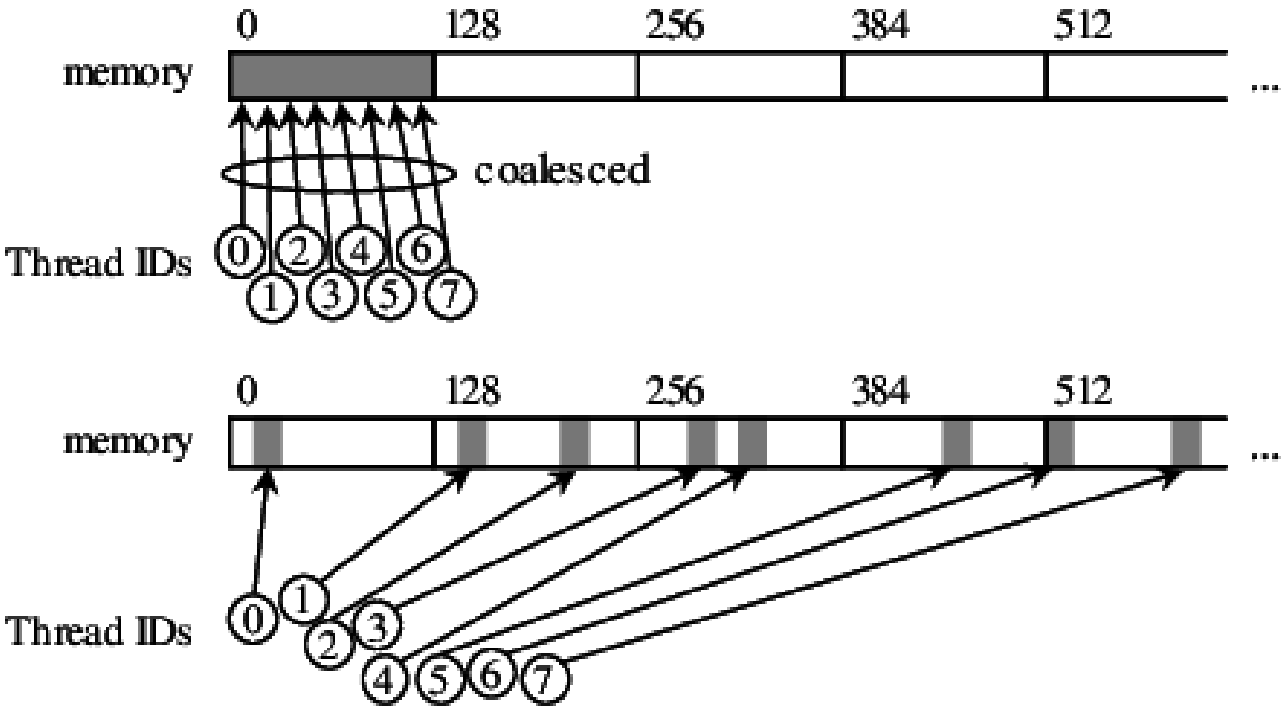
Memory Coalescing

- The Source Counters section also indicates uncoalesced global accesses, again suggesting memory accesses as a problem.



Memory Coalescing

- Threads can only access global memory in *fixed-size transactions*, which are 32 bytes in size.



Keskin, Cetin, Kocak 2015

Here we can think of each threadID as getting the data for some track or hit. Then if the data for tracks 1-8 is coalesced, only one global access is needed. If not, then multiple global accesses (in this case 5) are needed to retrieve the data. This leads to more accesses so warps will stall waiting in the global instruction queue.

Struct of Arrays (SOA) vs. Array of Structs (AOS)

- The question of whether to define data structures as structs of arrays or arrays of structs appears frequently in parallel programming.
- Array of structs:

```
1struct coords {
2        float x;
3        float y;
4        float z;
5}
6
7coords* array_of_coords;
8
```

Struct of Arrays vs. Array of Structs

- The question of whether to define data structures as structs of arrays or arrays of structs appears frequently in parallel programming.
- Struct of arrays:

```
1struct CoordsArray {
2private:
3    const float * coords;
4    const unsigned num_hits;
5public:
6    // Constructor
7    __host__device__ CoordsArray(float * base_pointer, const unsigned offset, const unsigned total_number_of_hits) :
8    m_base_pointer(base_pointer + offset), num_hits(total_number_of_hits) {}
9
10    // Getter functions.
11    __host__ __device__ float x(unsigned index) const { return coords[index]; }
12    __host__ __device__ float y(unsigned index) const { return coords[index + num_hits]; }
13    __host__ __device__ float z(unsigned index) const { return coords[index + 2 * num_hits]; }
14
15    // Setter functions.
16    __host__ __device__ void set_x(unsigned index, float value) { coords[index] = value; }
17    __host__ __device__ void set_y(unsigned index, float value) { coords[index + num_hits] = value; }
18    __host__ __device__ void set_z(unsigned index, float value) { coords[index + 2 * num_hits] = value; }
19}
20
```

SOA vs. AOS

- A general heuristic is to use SOA to keep global memory accesses coalesced.

SOA vs. AOS

- A general heuristic is to use SOA to keep global memory accesses coalesced.
- However, this intuition is often wrong, and there is no substitute for trying out both structures and comparing performance.

SOA vs. AOS

- A general heuristic is to use SOA to keep global memory accesses coalesced.
- However, this intuition is often wrong, and there is no substitute for trying out both structures and comparing performance.
- Also possible to use combinations of SOA and AOS. And different variables can be split up into different SOA's depending on where they are used.

fit_secondary_vertices

- The fit_secondary_vertices algorithm uses an array of TrackMVAVertex structs (AOS) to store all the information needed for a secondary vertex fit.

```
// Secondary vertices.
VertexFit::TrackMVAVertex* event_secondary_vertices = parameters.dev_consolidated_svs + sv_offset;

// Loop over svs.
for (unsigned i_sv = threadIdx.x; i_sv < n_svs; i_sv += blockDim.x) {
  event_secondary_vertices[i_sv].chi2 = -1;
  event_secondary_vertices[i_sv].minipchi2 = 0;
  auto i_track = event_svs_trk1_idx[i_sv];
  auto j_track = event_svs_trk2_idx[i_sv];
  const ParKalmanFilter::FittedTrack trackA = event_tracks[i_track];
  const ParKalmanFilter::FittedTrack trackB = event_tracks[j_track];

  // Do the fit.
  // TODO: In case doFit returns false, what should happen?
  if (doFit(trackA, trackB, event_secondary_vertices[i_sv])) {
    event_secondary_vertices[i_sv].trk1 = i_track;
    event_secondary_vertices[i_sv].trk2 = j_track;
```

fit_secondary_vertices

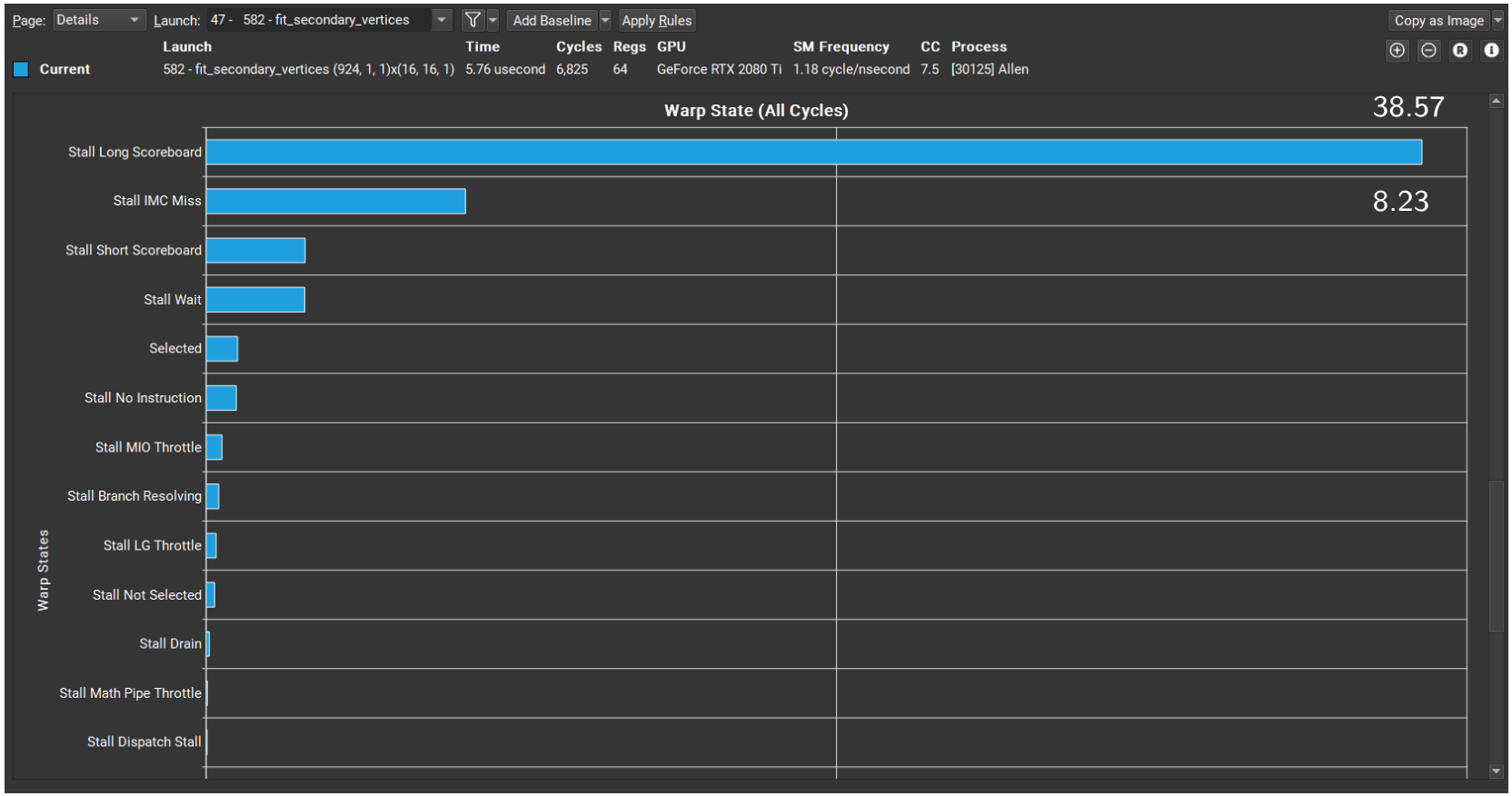
- We change this algorithm and the Kalman filtering algorithm to use multiple SOA's. We also cut down on global memory accesses by declaring data members in register memory.

fit_secondary_vertices

- We change this algorithm and the Kalman filtering algorithm to use multiple SOA's. We also cut down on global memory accesses by declaring data members in register memory.
- Does the number of warp stalls decrease?

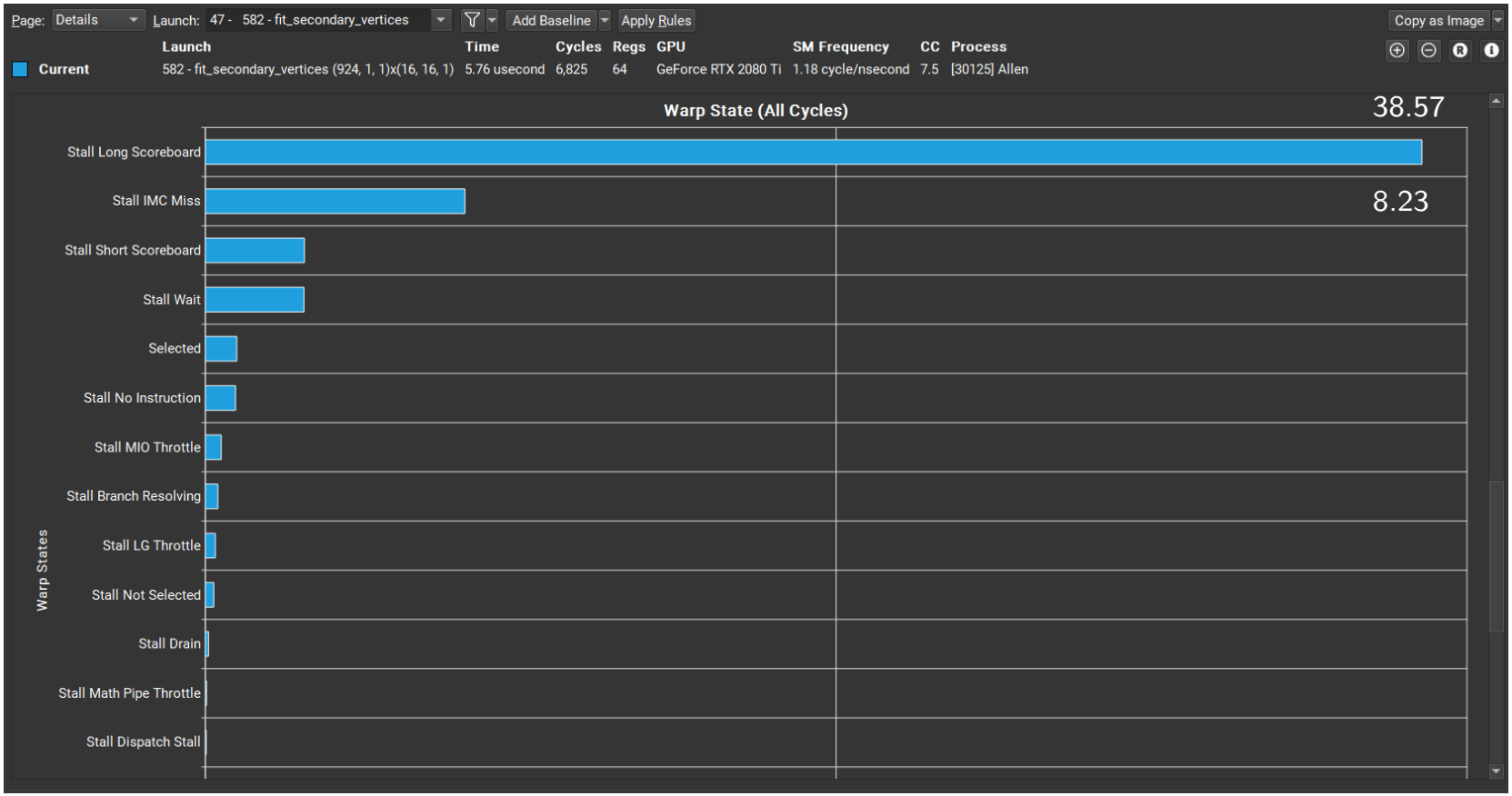
fit_secondary_vertices

- We change this algorithm and the Kalman filtering algorithm to use multiple SOA's. We also cut down on global memory accesses by declaring data members in register memory.
- Does the number of warp stalls decrease?



fit_secondary_vertices

- The LG Throttle stall is reduced to 0.32 cycles per instruction. The largest stall is Long Scoreboard, with 38.57 cycles per instruction.



Outlook

- In the long run, this view-based model can be integrated into the entire pipeline. This is being worked on by Tom and Daniel.

Outlook

- In the long run, this view-based model can be integrated into the entire pipeline. This is being worked on by Tom and Daniel.
- Ultimate test will be the effect on throughput.

Takeaways

- There are many optimizations that can be made to Allen.

Takeaways

- There are many optimizations that can be made to Allen.
- Profiling tools give a lot of information and can offer a guided look into speedup opportunities. However it is up to the programmer to decide which information is relevant and how this information can inform how we design the code.

Takeaways

- There are many optimizations that can be made to Allen.
- Profiling tools give a lot of information and can offer a guided look into speedup opportunities. However it is up to the programmer to decide which information is relevant and how this information can inform how we design the code.
- Getting maximum performance out of Allen will require carefully integrating the software with constraints of the hardware, specifically the CUDA programming model.

Acknowledgments

- Tom Boettcher and Daniel Campora

Acknowledgments

- Tom Boettcher and Daniel Campora
- University of Cincinnati and DIANA fellows program

- Any questions?