Profiling and data access patterns in Allen

Douglas Li

Stanford University

October 2021

Douglas Li

Profiling and data access patterns in Allen

4 October 2021 1 / 26



2 Allen and GPUs



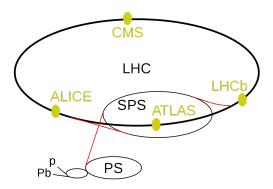
A quick introduction to LHCb

2 Allen and GPUs

3 Profiling and data access patterns

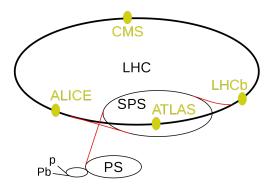
LHCb detector

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



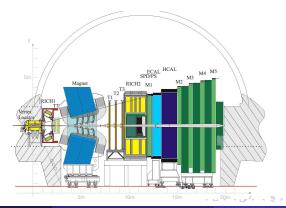
LHCb detector

- 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.



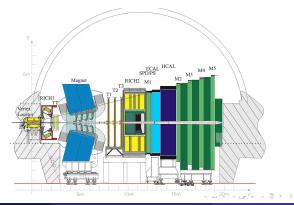
LHCb upgrades

• 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

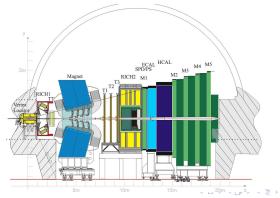
- The LHCb is currently shut down for upgrades before Run 3 of data taking. Importantly, Run 3 will not use a hardware trigger.
- Only a small fraction of the data read out from subdetectors can be kept for study: the trigger selects which events to keep.



Douglas Li

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.



A quick introduction to LHCb



3 Profiling and data access patterns

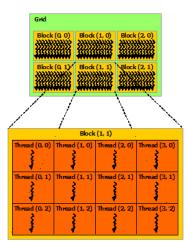
💮 LHCb 🔸 🔕 Allen		
Allen @ Project ID: 38633		☆ Star 31 ¥ Fork 18
- •- 5,717 Commits 🕑 605 Branches 🗸	🤊 16 Tags 🗈 6.3 MB Files 🕞 330.8 GB Storage 1	16 Releases
Full software HLT1 reconstruction sec	uence on GPU.	
master v Allen / +	~ Histo	ory Find file Web IDE 🖄 🗸 Clone 🗸
Merge branch 'dcampora_defa Christoph Hasse authored 10 ho	ult_to_building_single_sequence' into 'master' 🐽	a613e2f3 🛱
README T Apache License 2.0	CONTRIBUTING	
Name	Last commit	Last update
Dumpers		
E Rec/Allen		
ReleaseNotes		
🖨 backend		

3

Image: A match a ma

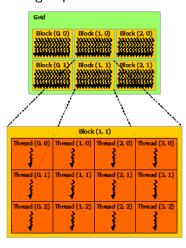
- "HLT1 reconstruction sequence": Stage 1 of software trigger. A sequence of algorithms that does the following tasks:
 - "Hits" are identified in each of the VELO, UT, and SciFi detectors.
 - (Each detector has several layers, and as particles pass through the layers, they leave signals in each layer. 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

• Grid-block-thread hierarchy.



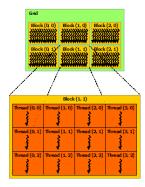
CUDA C++ Programming Guide

- 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

 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.



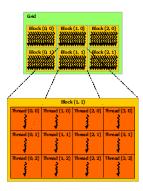
CUDA C++ Programming Guide

・ロト ・ 同 ト ・ ヨ ト ・ ヨ ト

Douglas Li

Profiling and data access patterns in Allen

- 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

< ロ > < 同 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 三 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 > < 1 >



2 Allen and GPUs



Profiling Allen

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

Page: Details 👻	Launch: 47 - 616 - fit_secondary_ve	rtices 👻	🟹 👻 Add Bas	eline 👻 A	oply <u>R</u> u	ules				Co	py as l	image	
	Launch		Time	Cycles F	tegs	GPU	SM Frequency		Process	۲	Θ	8	0
Current	616 - fit_secondary_vertices (935, 1	, 1)x(16, 16, 1)	358.94 usecond	498,246 €		GeForce RTX 2080 Ti	1.39 cycle/nsecond		[133860] Allen				
GPU Speed Of Li	ght												
High-level overview of	the utilization for compute and mem	nory resource:	s of the GPU. For e	ach unit, th	: Spee	d Of Light (SOL) report	s the achieved percer	ntage	e of utilization with respect to the theoretical maximum.				
SOL SM (%)						11.43 Duration	n [usecond]				35	58.94	
SOL Memory [%]						34.48 Elapsed	Cycles [cycle]				498	3,246	
SOL L1/TEX Cache [%						36.48 SM Acti	ve Cycles [cycle]				398,03	1.51	
SOL L2 Cache [%]						29.87 SM Freq	uency (cycle/nsecon	d]					
SOL DRAM [%]						34.48 DRAM F	requency [cycle/nsec	cond]				6.91	
A Bottleneck	This kernel exhibits low compute thr indicate latency issues. Look at <u>Sch</u>	oughput and r aduler Statisti	nemory bandwidth <u>cs</u> and <u>Warp State</u>	utilization Statistics	elative or pote	e to the peak performa ential reasons.	nce of this device. Ac	chiev	ved compute throughput and/or memory bandwidth below 60	.0% of peak typi	cally		
Compute Workle	ad Analysis												
Detailed analysis of the performance.	e compute resources of the streami	ng multiproce	ssors (SM), includ	ing the achi	eved ir	nstructions per clock (IPC) and the utilizatio		each available pipeline. Pipelines with very high utilization m	night limit the ov	erall		
Executed Ipc Elapsed	[inst/cycle]					0.19 SM Busy	y (%)					6.01	
Executed Ipc Active [i	nst/cycle]					0.24 Issue SI	ots Busy [%]						
Issued Ipc Active [ins	t/cycle]					0.24							
🔥 High Pipe Uti	lization All pipelines are under-uti	lized. Either tl	nis kernel is very s	mall or it do	iesn't i	issue enough warps pe	r scheduler. Check th		unch Statistics and Scheduler Statistics sections for further	details.			
- Memory Worklo	ad Analysis												
	e memory resources of the GPU. Me Max Bandwidth), or by reaching the m						hen fully utilizing the	invol	lved hardware units (Mem Busy), exhausting the available co	mmunication ba	ndwidt		
Memory Throughput	Gbyte/second]					228.67 Mem Bu	isy [%]				2	4.46	
L1/TEX Hit Rate [%]						72.61 Max Bar	dwidth [%]					4.48	
L2 Hit Rate [%]						93.19 Mem Pip	pes Busy [%]						
 Scheduler Statis 	tics												
every cycle each sche	duler checks the state of the allocate	ed warps in th	e pool (Active War	ps). Active	warps	that are not stalled (El	igible Warps) are rea	dy to	of warps in the pool (Theoretical Warps) is limited by the lat issue their next instruction. From the set of eligible warps to ued. Having many skipped issue slots indicates poor latency	he scheduler se	on. On ects a		

Profiling and data access patterns in Allen

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

- 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.

 Warp State Statistics 			Q
instructions. The higher the value, the more wa	rp parallelism is required to hide this latency. For each w	rarp state, the chart shows the average number of cycles spent in that us every cycle. When executing a kernel with mixed library and user c	warp cycles per instruction define the latency between two consecutive t state per issued instruction. Stalls are not always impacting the overall ode, these metrics show the combined values.
Warp Cycles Per Issued Instruction [cycle]		193.40 Avg. Active Threads Per Warp	
Warp Cycles Per Executed Instruction (cycle)		194.01 Avg. Not Predicated Off Threads Per Warp	
▲ CP1 Stall 'LG Throttle' issung two	each warp of this komel spends 194.8 cycles being stalle instructions. Typically this stall occurs only when execut ider memory operations and try interfeaving memory ope	d weiting for the local/global instruction queue to be not full. This re ing local or global memory instructions extremely frequently. If appli- rations and math instructions.	presents about 69.7% of the total average of 193.4 cycles between cable, consider combining multiple lower-width memory operations
Thread Divergence to 21.0 threads execute the inst	ion, and divergent flow control can algorificantly lower the per warp due to predication. The compiler may use predic	cation to avoid an actual branch. Instead, all instructions are achedul	use the same instruction. The chasse (auroch configuration, welly, a versing of 212 threads being active per cycle. The list for their robated et, but a per-thread constition code or predicate controls which threads perdent Thread Scheduling, which allows a warp to reconverge after a
Warp Stall Warp stall analysis			
		Warp State (All Cycles)	
Stall LG Throttle			134.81
Stall Long Scoreboard			52.52
Stall Walt			
Stall Not Selected			
Selected			
Stall IMC Miss			
Stall Drain			

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.

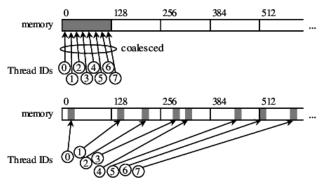
Memory Coalescing

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

_			
► S	ource Counters		~ Q
		and sampled warp stall reasons. Sampling Data metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the doc s on stalls if the schedulers fail to issue every cycle.	cumentation for
Bran	ch Instructions [inst]	121,248 Branch Efficiency [%]	98.05
Bran	ch Instructions Ratio [%]	0.03 Avg. Divergent Branches	6.91
A	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x7107c6ds205d0 at /home/doli/test/Allen/device/vertex_fit/vertex_fitter/src/VertexFitter.ou/70	
A	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC (http://doi.org/101/est/s0150) at /home/doil/test/Allen/device/vertex_fit/vertex_fitter/src/VertexFitter.cu/71	
▲	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x107te555a020 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.cc:39	
▲	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0/07/e5/da(80) at /home/doll/test/Allen/device/vertex_fit/vertex_fit/enter/include/VertexFitter.icc:38	
•	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x2107e5ds0cc0 at /home/doll/test/Allen/device/vertex_fit/vertex_fit/enter/include/VertexFitter.icc:37	
▲	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x7107e6dda120 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.icc:50	
▲	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0/7/07/e6dda3a0 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.icc/84	
▲	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0//107e5/sc/30 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.icc:170	
A	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x7107ze5ds/440 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.icc:170	
A	Uncoalesced Global Accesses	Uncoalesced global access, expected 27896 sectors, got 101552 (3.64x) at PC 0x7107z6565050 at /home/doll/test/Allen/device/vertex_fit/vertex_fitter/include/VertexFitter.icc:170	
	PC sampling data PC sampling	data	Apply

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.

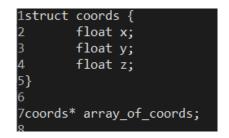
Douglas Li

Profiling and data access patterns in Allen

4 October 2021 16 / 26

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:



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:

.struct CoordsArray {
private:
<pre>const float * coords;</pre>
const unsigned num_hits;
public:
5 // Constructor
7host_device_ CoordsArray(float * base_pointer, const unsigned offset, const unsigned total_number_of_hits)
<pre>m_base_pointer(base_pointer + offset), num_hits(total_number_of_hits) {}</pre>
) // Getter functions.
hostdevice float x(unsigned index) const { return coords[index]; }
<pre>2hostdevice float y(unsigned index) const { return coords[index + num_hits]; }</pre>
hostdevice float z(unsigned index) const { return coords[index + 2 * num_hits]; }
6 // Setter functions.
ihostdevice void set_x(unsigned index, float value) { coords[index] = value; }
<pre>/hostdevice void set_y(unsigned index, float value) { coords[index + num_hits] = value; }</pre>
<pre>Bhostdevice void set_z(unsigned index, float value) { coords[index + 2 * num_hits] = value; }</pre>
1}

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

< /□ > < ∃

- 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.

- 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.

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

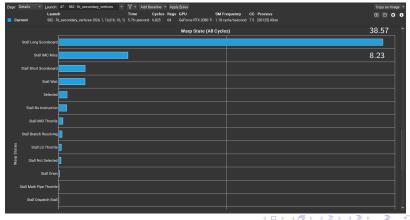
VertexFit::TrackMVAVertex* event secondary vertices = parameters.dev consolidated sys + sy offset; for (unsigned i sv = threadIdx.x; i sv < n svs; i sv += blockDim.x) {</pre> event secondary vertices[i sy].chi2 = -1; event secondary vertices[i sv].minipchi2 = 0; auto i track = event svs trk1 idx[i sv]; auto i track = event svs trk2 idx[i sv]; const ParKalmanFilter::FittedTrack trackA = event tracks[i track]; const ParKalmanFilter::FittedTrack trackB = event tracks[j track]; if (doFit(trackA, trackB, event secondary vertices[i sv])) { event secondary vertices[i sv].trk1 = i track; event secondary vertices[i sy].trk2 = i track:

< ロト < 同ト < ヨト < ヨ

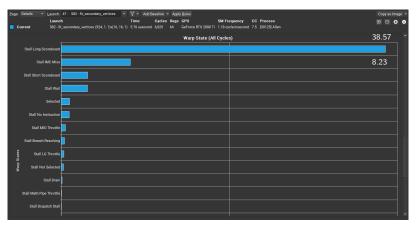
• We change this algorithm and the Kalman filtering algorithm to use multiple SOA's. We also cut down on global memory accesses by declaring data members in register memory.

- 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?

- 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?



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



< □ > < 同 > < 三 > <

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

- 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.

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

- 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.

- 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.

• Tom Boettcher and Daniel Campora

- (日)

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

• Any questions?

< 1 k