General Purpose GPU Computing in High Energy Physics Event Filtering

Dorothea vom Bruch

May 22nd 2019
5th Infieri Summer School
HUST, Wuhan, China
Outline

• Why are we interested in GPUs?
• History of GPU Computing
• Parallel programming
• What is a GPU?
• How is it different to CPUs, FPGAs?
• Where to use GPUs in High Energy Physics?
• GPUs in HEP event filtering
Part 1

Why are we interested in GPUs?
Moore’s prediction

• Moore's prediction:
  From 1975 on, double transistor count every two years

• 70s - 90s:
  Increase of hardware speed was enough to speed up sequential programs

Moore’s law today

Clock speed stopped increasing due to heat limit

Multiple core processors emerge (Intel i7: 4 cores)

40 Years of Microprocessor Trend Data

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Year


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Oluwotun, L. Hammond, and C. Batten. New plot and data collected for 2010-2015 by K. Rupp
What about the cost?

- Usually assume flat budget for computing cost estimation
- Estimated improvement increase: 10-15% per year
- Smartphone market is flat
- PCs, tablets etc. sales are declining
- Little innovation incentive for companies
Computing demands: ATLAS

• Mainly okay for Run3
• Very challenging for Run4
• Especially in terms of storage
Comp. demands: ALICE, Run 3

- Major detector upgrade for Run3
- 2 orders of magnitude more data from up to 500 million detector channels
- Continuous (trigger-less) readout with up to 3.4 TB/s from detector
- New facility for data processing and compression – 1500 CPU/GPU nodes, 60 PB storage
Comp. demands: LHCb, Run 3

- Major detector & readout upgrade
- Continuous (trigger-less) readout with up to 40 Tb/s from detector
- New data center being built

LHCb Upgrade Trigger Diagram

30 MHz inelastic event rate (full rate event building)

Software High Level Trigger

Full event reconstruction, inclusive and exclusive kinematic/geometric selections

Buffer events to disk, perform online detector calibration and alignment

Add offline precision particle identification and track quality information to selections
Output full event information for inclusive triggers, trigger candidates and related primary vertices for exclusive triggers

10 GB/s to storage
How to handle the computing load?

- Improve algorithms used in event selection
- Local data processing $\rightarrow$ less network and buffers required
- Integrate hard- and software
- Make use of the many cores
- Large interest in GPU computing emerged

$\rightarrow$ Let’s dive into GPU computing!
Part 2

History of GPU Computing
GPU: Actual purpose

Display graphics on the computer screen
Graphics Programming Unit

**Vertex/index buffers:**
Description of image with vertices and their connection to triangles

**Vertex shading**
For every vertex: calculate position on screen based on original position and camera view point

**Rasterization**
Get per-pixel color values

**Pixel shading**
For every pixel: get color based on texture properties (material, light, ...)

**Rendering**
Write output to render target

http://fragmentbuffer.com/gpu-performance-for-game-artists/

D. vom Bruch
GPU requirements

- Graphics pipeline: huge amount of arithmetic on independent data:
  - Transforming positions
  - Generating pixel colors
  - Applying material properties and light situation to every pixel

**Hardware needs**

- Access memory simultaneously and contiguously
- Bandwidth more important than latency
- Floating point and fixed-function logic
First GPUs

- 3D graphics hardware evolved since 1980s
- Since then: Price dropped, performance increased
- Driven by real-time graphics in computer applications

1980s - 2000s

- **1980s and 1990s:**
  Fixed built-in functions in hardware to do rasterizing, shading etc.
- **Functions depend on rendering algorithms → make programmable**
- **Early 2000s:**
  First programmable graphics hardware

Mid 2000s: unified processors for graphics stages

→ Programmable GPU processors could be used for general purpose computing

GPGPU programming

- Early days of GPGPU: problems had to be translated to graphics language via OpenGL
  → very tedious task, no random-byte addressing
- Processors with instruction memory, cache and sequencing control were added
- Memory load / store functions for random-byte addressing
- CUDA / OpenCL as APIs
GPU performance over time

Part 3

Parallel programming
Amdahl’s law

Speedup in latency = \( \frac{1}{(S + \frac{P}{N})} \)

- \( S \): sequential part of program
- \( P \): parallel part of program
- \( N \): number of processors

- Parallel part: identical, but independent work
- Consider how much of the problem can actually be parallelized!
### SISD vs. SIMD

<table>
<thead>
<tr>
<th>SISD</th>
<th>MIMD</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction</td>
<td>Multiple Instruction</td>
<td>Single Instruction</td>
</tr>
<tr>
<td>Single Data</td>
<td>Multiple Data</td>
<td>Multiple Data</td>
</tr>
<tr>
<td><strong>Uniprocessor machines</strong></td>
<td><strong>Multi-core, grid-, cloud-</strong></td>
<td><strong>e.g. vector processors</strong></td>
</tr>
</tbody>
</table>

**Diagram:**
- **SISD:** Single Instruction, Single Data
- **MIMD:** Multiple Instruction, Multiple Data
- **SIMD:** Single Instruction, Multiple Data

**Legend:**
- **PU:** Processing Unit
- **Data Pool:** Source of data
- **Instruction Pool:** Source of instructions
## SISD vs. SIMT

<table>
<thead>
<tr>
<th>SISD</th>
<th>MIMD</th>
<th>SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction</td>
<td>Multiple Instruction</td>
<td>Single Instruction</td>
</tr>
<tr>
<td>Single Data</td>
<td>Multiple Data</td>
<td>Multiple Threads</td>
</tr>
</tbody>
</table>

**Uniprocessor machines**

- SISD
- MIMD
- SIMT

- GPUs

**Diagram**

- SISD: Single Instruction Single Data
- MIMD: Multiple Instruction Multiple Data
- SIMT: Single Instruction Multiple Threads

**Uniprocessor machines**

- Single Instruction Single Data
- Multiple Instruction Multiple Data
- Single Instruction Multiple Threads

**Multi-core, grid-, cloud-computing**

**GPUs**
Single Instruction Multiple Threads

**SIMT**

- Similar to programming a vector processor
- Use threads instead of vectors
- No need to read data into vector register
- Only one instruction decoder available
  \[\rightarrow\text{ all threads have to execute the same instruction}\]
- Abstraction of vectorization:
  - Each element of vector is processed by an independent thread
  - One instruction fills entire vector
  - \# of threads = vector size
SIMT vector addition

- Same instruction on independent data
- One thread calculates one element of the output vector
What is a GPU?
What is a GPU?

- Multiple SIMT threads grouped together: GPU core
- Multiple cores grouped together: GPU
### Terminology

<table>
<thead>
<tr>
<th>CUDA (Nvidia)</th>
<th>OpenCL (AMD / Nvidia)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Processing Element</td>
</tr>
<tr>
<td>CUDA Core</td>
<td>SIMD Unit</td>
</tr>
<tr>
<td>Streaming Multiprocessor</td>
<td>Compute Unit</td>
</tr>
<tr>
<td>GPU Device</td>
<td>GPU Device</td>
</tr>
</tbody>
</table>
GPU Architecture: Nvidia Pascal
Pascal: Streaming Multiprocessor

- Scheduler
- Dispatch unit
- 64 single precision cores (FP32)
- 32 double precision cores (FP64)
- Load / store units
- Special function units
GPU Architecture: AMD Polaris

A GPU’s natural habitat

Host

DRAM

CPU Chip

PCIe

Device

DRAM

GPU Chip
A GPU’s natural habitat

The PCIe connection can be a bottleneck
## Talk to a GPU: PCIe

<table>
<thead>
<tr>
<th>PCIe generation</th>
<th>1 lane</th>
<th>16 lanes</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>500 MB/s</td>
<td>8 GB/s</td>
<td>2007</td>
</tr>
<tr>
<td>3.0</td>
<td>985 MB/s</td>
<td>15.75 GB/s</td>
<td>2010</td>
</tr>
<tr>
<td>4.0</td>
<td>1.97 GB/s</td>
<td>31.5 GB/s</td>
<td>2017</td>
</tr>
</tbody>
</table>
Talk to a GPU: NVLink, GPUDirect

**GPUDirect:**
- Direct memory access (DMA) transfer directly over PCIe switch
- Only available for scientific Nvidia GPUs

**NVLink:**
- Communications protocol developed by Nvidia
- Can be used between CPUs and GPUs and among multiple GPUs
- 160 / 300 GB/s data rate
Part 5

How does a GPU compare to its peers?
What do you see?
What do you see?

**CPU**


**GPU**

https://www.flickr.com/photos/130561288@N04/29111683364/in/album-72157650403404920/
GPU vs. CPU

http://images.anandtech.com/reviews/cpu/intel/SNBE/Core_I7_LGA_2011_Die.jpg
http://wccftech.com/nvidia-gtx-1080-gp104-die-shot/
## GPU vs. CPU: Specifications

<table>
<thead>
<tr>
<th></th>
<th>Intel Core E7-8890 v3</th>
<th>GeForce GTX 1080</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core count</td>
<td>18 cores / 36 threads</td>
<td>20 SMs / 2560 cores</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.5 GHz</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>Peak Compute Performance</td>
<td>1.8 GFLOPs</td>
<td>8873 GFLOPs</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>Max. 102 GB/s</td>
<td>320 GB/s</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>Max. 1.54 TB</td>
<td>8 GB</td>
</tr>
<tr>
<td>Technology</td>
<td>22 nm</td>
<td>16 nm</td>
</tr>
<tr>
<td>Die size</td>
<td>662 mm²</td>
<td>314 mm²</td>
</tr>
<tr>
<td>Transistor count</td>
<td>5.6 billion</td>
<td>7.2 billion</td>
</tr>
<tr>
<td>Model</td>
<td>Minimize latency</td>
<td>Hide latency through parallelism</td>
</tr>
</tbody>
</table>
GPU vs. CPU: Memory

CPU: Minimize latency
- Large, low latency cache
- High frequencies
- Speculative executions

Optimal serial performance

GPU: Hide latency
- Small cache with higher latency
- Lower frequencies
- No speculative executions
- Thousands of threads
  → always have work to do

Optimal parallel performance
And what about FPGAs?

- Field programmable gate array
- Configure a circuit to do the task it is programmed for
  → Hardware implementation of an algorithm
- Very good at integer computations
- Does not require a computer to run (has its own I/O)
- Traditionally, programmed with hardware description languages (Verilog, VHDL)
  → long development time
GPU vs. FPGA

- Higher latency
- Connection via PCIe (or NVLink)
- Bandwidth limited by PCIe
- Very good floating point operation performance
- Lower engineering cost
- Backward compatibility

- Low & deterministic latency
- Connectivity to any data source
- High bandwidth
- Intermediate floating point performance
- High engineering cost
- Not so easy backward compatibility
Where to use GPUs in experimental HEP?
Computing needs in HEP

Detector → Selection → Storage

Simulation

Data analysis

D. vom Bruch
GPUs for simulation

Example

- 3D convolutional GAN for fast simulation
- Train GAN to realistically simulate shower
  → speed up simulation processing time
- Use GPU for training
- Future plans: possibly use GPU for inference

S. Vallecorsa et al.

<table>
<thead>
<tr>
<th>Method</th>
<th>Machine</th>
<th>Time/Shower (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Simulation</td>
<td>Intel Xeon Platinum 8180</td>
<td>17000</td>
</tr>
<tr>
<td>(geant4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3d GAN (batch size 128)</td>
<td>Intel Xeon Platinum 8180 (TF 1.12)</td>
<td>1</td>
</tr>
</tbody>
</table>
GPUs for data analysis

Example

- Partial wave analysis using GooFit
- GooFit: GPU-friendly framework for maximum-likelihood fits
- Backend for Nvidia GPUs (using Thrust library) and for CPUs using OpenMP

<table>
<thead>
<tr>
<th>Platform</th>
<th>GPU Model</th>
<th>Chip</th>
<th>CUDA cores</th>
<th>Run time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWorkstation</td>
<td>Tesla K40c</td>
<td>GK110BGL</td>
<td>2880</td>
<td>76</td>
</tr>
<tr>
<td>Desktop PC</td>
<td>GeForce GTX 980</td>
<td>2nd gen. Maxwell (GM204)</td>
<td>2048</td>
<td>67</td>
</tr>
<tr>
<td>Laptop ASUS N56V</td>
<td>GeForce GT 650M</td>
<td>GK107</td>
<td>384</td>
<td>179</td>
</tr>
</tbody>
</table>

Running on one CPU core: 8 hours

arxiv.org 1703.03284
GPUs in event filtering
Event filtering

Detector readout

- High bandwidth
- Fixed latency

→ optimal task for FPGAs / ASICS
Event filtering

Detector

Local characteristic signature,
For example high energy / pt particle

Selection

Analysis of whole event required to select events → reconstruct all trajectories

Storage

D. vom Bruch
Event selection locally in hardware

- Necessary if not the whole data stream can be read out
- Possible if decision is based on local objects
- For example: part of a calorimeter
- Need low latency and high bandwidth $\rightarrow$ optimal for FPGAs / ASICS
  $\rightarrow$ hardware level triggers
Event selection globally in software

- Only possible if the whole data stream can be read out (or has been reduced before by a hardware level trigger)
- Necessary if the information from all (or several) detectors is needed for the selection decision
- No hard latency requirement
- Input data usually provided from CPUs that build the event
- Optimal task for CPUs and / or GPUs → software level triggers
Track reconstruction

- The most compute intensive part of event selection
- Consists of two steps:
  - Pattern recognition: Which hits belong to which track? → Huge combinatorics
  - Track fitting: Done for every track
- Many computations on independent data
  - Can process many events in parallel
  - In pattern recognition: can check many combinations of hits in parallel
  - In track fitting: can fit many tracks in parallel

→ Very well suited for GPUs!

GPUs in real-time reconstruction
ALICE event types

- One of the four large experiments at the LHC
- Take data with proton and heavy ion collisions
- Study quark-gluon plasma
- What happened just after the big bang?
- Main detector: large time projection chamber (TPC)
ALICE track reconstruction

Tracking Algorithm

On the GPU

• TPC Volume is split in 36 sectors.
  – The tracker processes each sector individually.
  – Increases data locality, reduce network bandwidth, but reduces parallelism.
  – Each sector has 160 read out rows in radial direction.

• 1. Phase: Sektor-Tracking (within a sector)
  – Heuristic, combinatorial search for track seeds using a Cellular Automaton.
    A) Looks for three hits composing a straight line (link).
    B) Concatenates links.
  – Fit of track parameters, extrapolation of track, and search for additional clusters using the Kalman Filter.

• 2. Phase: Track-Merger
  – Combines the track segments found in the individual sectors.
Cellular Automaton

- Build short track segments
- Find segments from different layers that fit together
  → build tree structure
- Track: tree with minimum number of connected segments
- Data locality and intrinsically parallel → well suited for many-core architecture
Kalman Filter

- One method for track fitting
- Subsequently iterates over all hits on a track
- For every hit, estimate the state of the track at that location:
  - First: predict it based on the previous state
  - Second: update it based on the measurement (hit)
Kalman Filter

• One method for track fitting
• Subsequently iterates over all hits on a track
• For every hit, estimate the state of the track at that location:
  • First: predict it based on the previous state
  • Second: update it based on the measurement (hit)
Kalman Filter

• One method for track fitting
• Subsequently iterates over all hits on a track
• For every hit, estimate the state of the track at that location:
  • First: predict it based on the previous state
  • Second: update it based on the measurement (hit)
Kalman Filter

- One method for track fitting
- Subsequently iterates over all hits on a track
- For every hit, estimate the state of the track at that location:
  - First: predict it based on the previous state
  - Second: update it based on the measurement (hit)
Kalman Filter

- One method for track fitting
- Subsequently iterates over all hits on a track
- For every hit, estimate the state of the track at that location:
  - First: predict it based on the previous state
  - Second: update it based on the measurement (hit)
- At last plane: best linear estimator for track state

- Only parallelizable over all tracks in one event
ALICE GPU Usage

- Run several events in parallel
- The event size is large, so not too many events fit into GPU memory at once
- Process the sectors of the TPC in parallel
- Data copied via PCIe
- Same code base for CPU and GPU code
  → can run on either architecture

<table>
<thead>
<tr>
<th>#</th>
<th>Phase</th>
<th>Task</th>
<th>Method</th>
<th>Locality</th>
<th>Time</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I</td>
<td>Seeding</td>
<td>Cellular Automaton</td>
<td>Very local</td>
<td>30%</td>
<td>CPU &amp; GPU</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Track following</td>
<td>Simple Kalman filter</td>
<td>Sector-local</td>
<td>60%</td>
<td>CPU &amp; GPU</td>
</tr>
<tr>
<td>3</td>
<td>II</td>
<td>Track Merging</td>
<td>Matching Covariance</td>
<td>Global</td>
<td>2%</td>
<td>CPU</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Final Fit</td>
<td>Kalman filter</td>
<td>Global</td>
<td>8%</td>
<td>CPU (or GPU)</td>
</tr>
</tbody>
</table>

arxiv 1712.09430
ALICE: GPUs in the Trigger

- First experiment to use GPUs in the trigger

**Runs 1 and 2:**

- Reliably running 24/7

<table>
<thead>
<tr>
<th>GPU</th>
<th>Shader</th>
<th>MHz</th>
<th>Time</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA GTX480 (Fermi)</td>
<td>448 shader</td>
<td>1215 MHz</td>
<td>174 ms</td>
<td>(used in the old HLT)</td>
</tr>
<tr>
<td>NVIDIA GTX780 (Kepler)</td>
<td>2304 shader</td>
<td>863 MHz</td>
<td>155 ms</td>
<td></td>
</tr>
<tr>
<td>NVIDIA Titan (Kepler)</td>
<td>2688 shader</td>
<td>837 MHz</td>
<td>146 ms</td>
<td></td>
</tr>
<tr>
<td><strong>AMD S9000 (Tahiti)</strong></td>
<td>1792 shader</td>
<td>900 MHz</td>
<td><strong>145 ms</strong></td>
<td>(used in the new HLT)</td>
</tr>
<tr>
<td>NVIDIA GTX980 (Maxwell)</td>
<td>2048 shader</td>
<td>1126 MHz</td>
<td>120 ms</td>
<td></td>
</tr>
</tbody>
</table>

*With both NVIDIA and AMD as possible vendors, we are no longer vendor-locked!*

**Plans for Run3 and beyond:**

- TPC is updated, time frames instead of collision events
- Adopt for these changes
- Add other parts of the event selection to the GPU, where tracks are needed
  → reduce PCIe copy overhead

D. Rohr
• One of the four large LHC experiments
• General purpose detector
• Testing GPU usage for HL-HLC: ~140 p-p collisions per bunch crossing
CMS: Pixel tracking

- PATATRACK project: create new trigger stage computing particle tracks using high performance computing
- Detector divided into sections in eta and phi
- Process sections in parallel
- Use Cellular Automaton for pattern recognition

Arxiv 1806.08975
Different options where to place GPUs for Run4

- In the event builder nodes
- In every filter farm node
- In a separate server of the filter farm
NA62

- Fixed target experiment at CERN
- Precision experiment: measure decay $K \rightarrow \pi \nu \nu$ with 10% precision
- L0 trigger: 10 MHz → 1 MHz
NA62: RICH reconstruction

- Reconstruct ring-shaped patterns for pion – muon separation
- **Method one:** Histogram of distances from every PM to measurements → in parallel, find histogram that best fits a circle
- **Method two:** Almagest algorithm, for points on circle: \[ AD*BC+AB*DC=AC*BD \] → test hypotheses of four hits in parallel
NA26: RICH L0 trigger on GPUs

- Run RICH GPU reconstruction in L0 trigger within 1 ms
- Dedicated network interface card (NaNet) to handle data transfers (I/O) and guarantee fixed latency
- Send data from NaNet via GPUDirect to the GPU over PCIe switch
- Tested in 2016 data taking → plan to integrate RICH L0 trigger in the future

![Graph showing time and events](image)

Arxiv 1606.04099
Mu3e

- Fixed target experiment at the Paul Scherrer Institute in Switzerland
- Study lepton flavor violating decay $\mu^+ \rightarrow e^+ e^- e^+$
- In Standard Model: suppressed to $\text{BR} < 10^{-54}$
- Any hint of signal $\rightarrow$ new physics
Mu3e detector readout

Use central pixel layers for event selection

- 2844 Pixel Sensors
- 3072 Fibre Readout Channels
- 6272 Tiles

- 10 Gbit/s output link per board
- 10 GB/s
- 100 MB/s

- 4 inputs each PCIe connection

- Data Collection Server
- Mass Storage

D. vom Bruch
Mu3e event reconstruction

**Track Fit**

- Find combinations of three hits in parallel
- Extrapolate to fourth layer in parallel
- Linear track fit:
  - Assume multiple scattering at middle hit
  - Minimize chi2

\[ \chi^2 = \frac{\Phi_{MS}^2}{\sigma_{MS, \Phi}^2} + \frac{\theta_{MS}^2}{\sigma_{MS, \theta}^2} \]

**Vertex estimate**

- Based on intersection of tracks projected onto plane transverse to magnetic field
- Test all track combinations in parallel
Mu3e reconstruction on GPUs

**PCle FPGA**
- Hits layer 1
- Hits layer 2
- Hits layer 3
- Hits layer 4
- Geometrical three-hit selection
- Coordinate transformation
- Recurl station hits, Timing information

**GPU**
- Vertex selection
- Positive tracks
- Negative tracks
- Propagation, four-hit fit
- Three-hit fit

**RAM buffer**
- DMA

**GPU memory**
- DMA

**Storage**
Mu3e event selection on GPUs

- Continuous beam (different to LHC experiments)
- Use time slices of 50 ns for reconstruction
- Plan for 12 filter farm PCs with one GPU each
  → Need to process at least $1.7 \times 10^6$ time slices per second per GPU
- Parallelize over events, hit combinations and tracks
- Measured $2 \times 10^6$ time slices / s on one Nvidia GTX 1080
  → Can do full event selection with 12 GPUs

- Mu3e will start taking data in the next years
LHCb

- One of the four large LHC experiments
- Mainly specialized in the physics of beauty hadrons
- Single-arm forward spectrometer
- Run 3: upgrade many detectors and full readout
LHCb Run 3 readout

- 27 GB/s of beauty hadrons
  → need refined selection decision
- Triggerless readout, full software trigger
- Huge compute challenge to reconstruct, calibrate and align all detectors in real-time
- Currently: exploring different architectures for this task
- One R&D project: run full first trigger stage (HLT1) on GPUs
LHCb HLT1 tasks

**Velo**
- Decode raw banks
- Clustering of pixel hits
- Track reconstruction
- Primary vertex reconstruction

**UT**
- Decode raw banks
- Track reconstruction

**SciFi**
- Decode raw banks
- Track reconstruction

**Muons**
- Decode raw banks
- Match hits to tracks

**Kalman filter**
- Track fitting of full track
  (Velo + UT + SciFi)

**Selections**
- 1-track selection
- 2-track selection
- Based on $p$, $p_t$, displacement and vertex criteria
Allen: LHCb HLT1 on GPUs

- Run thousands of events in parallel
- Exploit data-parallelism within events
- One GPU should process events at roughly 60 kHz
- Reduce data rate by factor 30 based on single and two-track selections

Data flow

Raw data → Selection decisions
LHCb: Velo on GPUs

- 26 planes of silicon pixel detectors
- Parallel algorithm for cluster finding using bitmasks
- Pattern recognition finding seed triplets in parallel
- Primary vertex finding using cluster finding in a histogram
LHCb: UT on GPUs

- 4 planes of silicon strip detectors
- Hits decoded into a memory pattern optimized for access in pattern recognition step
- Extrapolate Velo tracks in parallel
- Find hits of corresponding UT tracks in parallel
LHCb: SciFi on GPUs

- 12 layers of scintillating fibre detectors
- Extrapolate UT tracks to SciFi
- Find seeds from triplets of hits on x layers → extrapolate to other layers
- Obtain momentum estimate
- Process events, input UT tracks and SciFi track seeds in parallel
LHCb: Kalman filter on GPUs

- Use parametrization of magnetic field
  → don’t need to copy field map to the GPU
- Process events and tracks in parallel
- Can run Kalman filter on parts of the tracks or the complete track
LHCb: Allen project status

- Almost complete HLT1 chain implemented
- Throughput & phycis performance look promising
- Will test integration into DAQ system in more detail
- Review planned for this year

LHCb simulation, GPU R&D
Summary

- We need GPUs to handle the compute load in the future
- They are well suited for inherently parallel problems:
  run the same instructions on independent data
- Various applications in high energy physics
- Specifically event selection w/o hard latency requirement
- Some systems with GPUs running already
- Others possibly going online in the next years
- Maybe the concepts in this talk apply to your computing problem as well??
Backup
# Some Nvidia GPUs

<table>
<thead>
<tr>
<th>Feature</th>
<th>GeForce GTX 1060</th>
<th>GeForce GTX 1080 Ti</th>
<th>GeForce GTX 2080 Ti</th>
<th>Tesla T4</th>
<th>Tesla V100</th>
</tr>
</thead>
<tbody>
<tr>
<td># cores</td>
<td>1280</td>
<td>3584</td>
<td>4352</td>
<td>2560</td>
<td>5120</td>
</tr>
<tr>
<td>Max. frequency</td>
<td>1.81 GHz</td>
<td>1.67 GHz</td>
<td>1.55 GHz</td>
<td>1.59 GHz</td>
<td>1.37 GHz</td>
</tr>
<tr>
<td>Cache (L2)</td>
<td>1.5 MB</td>
<td>2.75 MB</td>
<td>6 MB</td>
<td>6 MB</td>
<td>6 MB</td>
</tr>
<tr>
<td>DRAM</td>
<td>5.94 GB GDDR5</td>
<td>10.92 GB GDDR5</td>
<td>10.92 GB GDDR5</td>
<td>16 GB GDDR6</td>
<td>32 GB HBM2</td>
</tr>
</tbody>
</table>

- **Gaming GPUs**
- **Scientific GPUs**