Algorithm Acceleration from GPGPUs for the ATLAS Upgrade
Computing in High Energy and Nuclear Physics 2010

Andrew Washbrook
on behalf of the ATLAS Collaboration

University of Edinburgh

21st October 2010
General Purpose GPUs

- GPU architectures are designed for running thousands of threads in parallel.
- No additional overhead from running many threads.
- Suited to problems which can be performed in a data parallel manner.
- APIs allow the host to manage the GPU device.
- Several APIs and SDKs can be used for GPGPU programming: Nvidia CUDA, OpenCL, AMD/ATI stream SDK.

Where to start?
Nvidia CUDA zone
GPU Projects at Edinburgh

- Number of GPU related projects at Edinburgh over the summer:

- **Chris Jones** - "Porting the Z finder algorithm to GPU" (MSc in High Performance Computing)

- **Maria Rovatsou** - "SIMT design of the High Level Trigger Kalman Fitter" (MSc School of Informatics)

- **James Henderson** - "An Investigation Into Particles Tracking and Simulation Algorithms using GPUs"

- Project reports and source code available at: [ATLAS Edinburgh GPU Computing](#)
Project Resources

- Access to a number of dedicated GPUs with different architectures (Tesla and Fermi).
- CUDA code based on CUDA version 3.1.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Tesla C1060</th>
<th>GeForce GTX 470</th>
<th>Tesla C2050 (x4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Capability</td>
<td>1.3</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Global Memory</td>
<td>4.3GB</td>
<td>1.3GB</td>
<td>2.8GB</td>
</tr>
<tr>
<td>Multiprocessors</td>
<td>30</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Cores</td>
<td>240</td>
<td>448</td>
<td>448</td>
</tr>
<tr>
<td>Threads/block</td>
<td>512</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>
The ATLAS Trigger

- **Level 1**: Custom built hardware with special processor units.
- **Level 2**: Software trigger operating independently on detector regions of interest (RoIs).
- **Event filter (Level 3)**: Software trigger analysing whole event signatures.
The ATLAS Trigger

- **Level 1**: Custom built hardware with special processor units.
- **Level 2**: Software trigger operating independently on detector regions of interest (RoIs). **Ideal for GPGPUs**
- Event filter (Level 3): Software trigger analysing whole event signatures.
Z Finder GPU Motivation

- Already break an event up into regions of interest (ROIs) for distributed processing.
- Break ROIs into slices of $\phi$ and process independently.
Z Finder GPU Motivation

- Already break an event up into regions of interest (ROIs) for distributed processing.
- Break ROIs into slices of $\phi$ and process independently.
- Candidate for parallelisation using GPUs.

Cross section view of the ATLAS detector
The Z Finder Algorithm

\[ z_V = \frac{Z_2 \cdot \rho_1 - Z_1 \cdot \rho_2}{\rho_1 - \rho_2} \]

- Process each combination of spacepoints and extrapolate back to the beam line.
- The histogram peak is the chosen interaction point.
Z Finder Test Case

- Standalone version of Z finder code used for feasibility studies with CUDA.
- Initially optimised for calculating $z_V$ using pairs of spacepoints.
- Timing performance measured using two samples of simulated events.

<table>
<thead>
<tr>
<th>Luminosity ($cm^{-2}s^{-2}$)</th>
<th>lowlum</th>
<th>highlum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(10^{32})$</td>
<td>333</td>
<td>8104</td>
</tr>
</tbody>
</table>

Total Execution Time - CPU (ms)

- lowlum: 0.11
- highlum: 7.13
Z Finder Kernel: Histogram Summation

**Code Iterations**

- Single thread per $\phi$ slice.
- Thread block per $\phi$ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

![Diagram of Z Finder Kernel: Histogram Summation](image)
Z Finder Kernel: Histogram Summation

**Code Iterations**

- Single thread per $\phi$ slice.
- Thread block per $\phi$ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

![Diagram showing Z Finder Kernel: Histogram Summation](image)
Z Finder Kernel: Histogram Summation

**Code Iterations**
- Single thread per $\phi$ slice.
- Thread block per $\phi$ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

![Diagram of Z Finder Kernel: Histogram Summation](image)

- **spacepoint**
- **layer separator**
Z Finder Kernel: Histogram Summation

**Code Iterations**

- Single thread per $\phi$ slice.
- Thread block per $\phi$ slice.
- Histogram per thread block in shared memory.
- Improve spacepoint pair allocation method.

[Diagram of Code Iterations]

[Diagram of Algorithm Flow]

**Global Memory**

- Histories
- Rol Data

**Copy Data to Device**

**Summation Kernel**

**Reduction Kernel**

**Get Histogram**

**Find Z peak**

- Spacepoint
- Layer separator
ZFinder Kernel: Histogram Combination

**Code Iterations**

- Combine histograms on the GPU ⇒ reduce device to host data transfer by $\sim 500x$.
- Reduce the data to a single histogram in multiple steps.

![Diagram showing stages of histogram combination on the GPU](image-url)
ZFinder Kernel: Histogram Combination

**Code Iterations**

- Combine histograms on the GPU ⇒ *reduce device to host data transfer by ∼500x*.
- Reduce the data to a single histogram in multiple steps.
Each RoI calculation independent $\Rightarrow$ use CUDA streams.

Successful in disguising any host to device transfer latency.
Results for spacepoint pairs show up to 35x speed-up (Fermi).
Initial results for spacepoint *triplets* also show speed-up.
Kalman Filter GPU Motivation

- Potentially *thousands* of tracks to reconstruct for every event in the trigger.

- Significant acceleration possible by reconstructing one track per GPU thread.

**GPU benefits at other experiments**

- Kalman Filter port to CUDA (GSI Scientific Report 2008, FAIR-EXPERIMENTS-38)
- ALICE TPC HLT code GPU based / future PANDA TPC code
- GPUs to be used for STS (Silicon Tracking System) within CBM (Compressed Baryonic Matter) experiment at FAIR/GSI.
Tracks reconstructed using the Kalman filter method.

The trajectory of a track is predicted using detector hits as input.

Backward smoothing filter applied after final Kalman Filter estimation.

C++ Class Hierarchy of Track Objects
Kalman Filter for CUDA

Initial Complications

- Class inheritance structure captures filter specialism for each sub-detector.
- Dynamic creation of objects in the main routine.
- Track state retention at each filtering step.
- Break down main routine for a smaller kernel.

Feasibility Studies (Maria Rovatsou)

- Standalone version successfully ported to C.
- Pre-allocated memory needed for track objects.
- Promising results ⇒ memory footprint per track needs to be reduced.
Kalman Filter for CUDA

Initial Complications
- Class inheritance structure captures filter specialism for each sub-detector.
- Dynamic creation of objects in the main routine.
- Track state retention at each filtering step.
- Break down main routine for a smaller kernel.

Feasibility Studies (Maria Rovatsou)
- Standalone version successfully ported to C.
- Pre-allocated memory needed for track objects.
- Promising results ⇒ memory footprint per track needs to be reduced.
Standalone version successfully ported to C.

Structs of arrays used to store track data.

Vector data types (e.g. float4) for compact representation of data.

One GPU thread per track.

Modification of smoothing algorithm required for single precision arithmetic.

Muon tracks, $p_T=10\text{GeV}$, full MC simulation

Over 5x speed-up seen at 3000 tracks.
The ATLAS trigger, particle tracking & simulation algorithms are key areas where GPUs can be used to improve performance.

Significant enhancements to the trigger and reconstruction algorithms will prove invaluable for dealing with the rates from the LHC upgrade.

Also observed an initial 32x speed-up for parallel Runge-Kutta integration *(see supplementary material)*.

Best case optimisation of 35x speed-up for the Z Finder routine.

Port of OO-based Kalman Filter algorithm showed GPU acceleration is feasible and scales to thousands of tracks.
"Fermi" GPU
Images from Gernot Ziegler, Nvidia
A CUDA kernel is a function which is executed in parallel by a number of threads on the GPU device.

A *thread block* is a set of threads which execute together on a single multiprocessor.

Thread blocks can be arranged into a one or two dimensional grids.
CUDA devices contain different types of memory, each with their own properties.

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Size</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>1GB+</td>
<td>Main memory storage on the GPU.</td>
</tr>
<tr>
<td>Shared</td>
<td>16/48KB (block)</td>
<td>Allows data to be shared between threads in the same block.</td>
</tr>
<tr>
<td>Registers</td>
<td>16/32KB (MP)</td>
<td>Stores kernel variable data (for each thread).</td>
</tr>
<tr>
<td>Local</td>
<td>16/512KB (thread)</td>
<td>Overflow for thread variable storage.</td>
</tr>
<tr>
<td>Constant</td>
<td>64KB</td>
<td>Automatically cached, read only.</td>
</tr>
<tr>
<td>Texture Memory</td>
<td>6-8KB (MP)</td>
<td>Streaming fetches with a constant latency.</td>
</tr>
</tbody>
</table>
Particle tracking in a magnetic field

Preliminary GPGPU test case study

- Charged particles bend in the magnetic field
- Lorentz force (perpendicular to plane of magnetic field)

\[
\mathbf{F} = m \mathbf{a} = q \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})
\]

\[
\frac{d\mathbf{v}}{dt} = \mathbf{a} = \frac{q}{m} \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})
\]

- Solve the differential equation with 4th order Runge Kutta Integration
Magnetic Field Integration results

- Rapidly achieved a factor 32 speedup (more in progress).