A new approach to Track Finding in the TPC - Study of a Fast Hough Transform on a GPU

Felix Böhmer, Sebastian Neubert

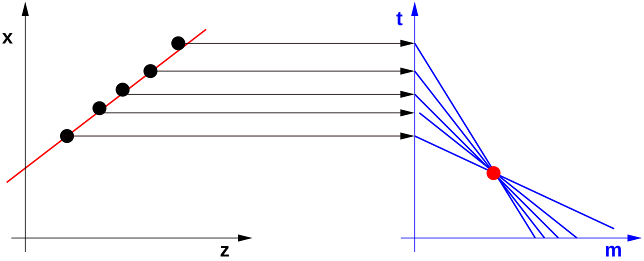
Physik Department E18
Technische Universität München

$\overline{\text{P}}$ANDA collaboration meeting Juelich,
September 2009

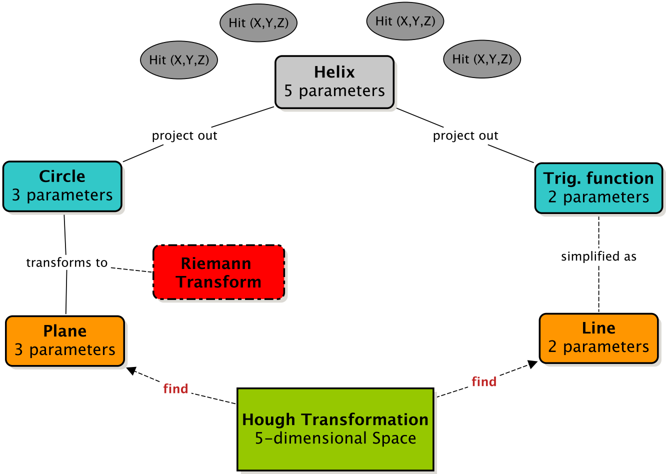
Goal: Hough Transform for Track Finding

- Wanted: Alternative to track-following
- One direct method: **Hough Transform**

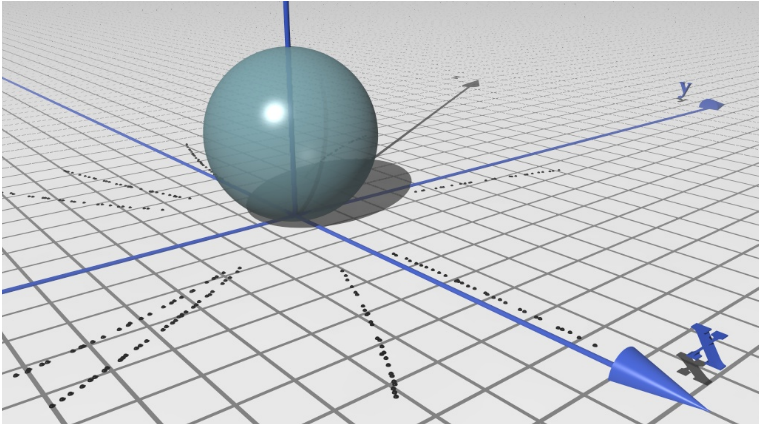
- Working principle:



Our Implementation: Hough Transform Topology

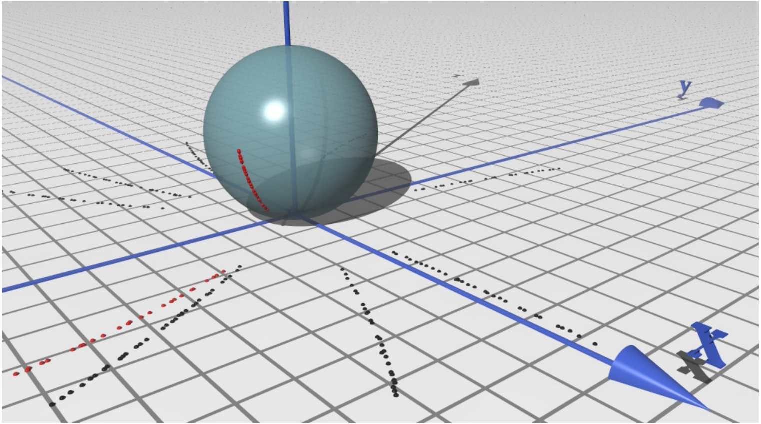


Riemann Transform Example



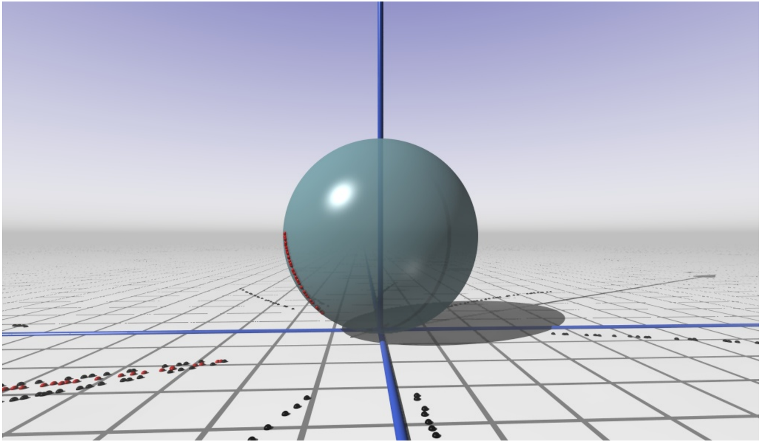
Schematic: Riemann Sphere on top of the Readout Plane and some example tracks

Riemann Transform Example



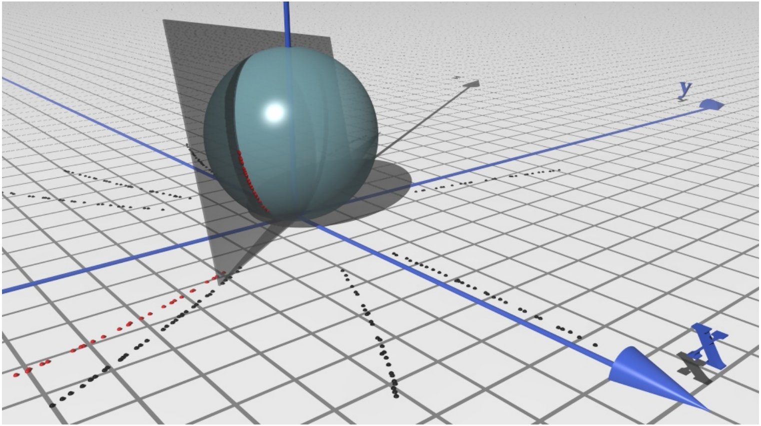
Schematic: Riemann Sphere on top of the Readout Plane and some example tracks

Riemann Transform Example



Schematic: Riemann Sphere on top of the Readout Plane and some example tracks

Riemann Transform Example



Schematic: Riemann Sphere is cut by the plane connected to the hits of one track

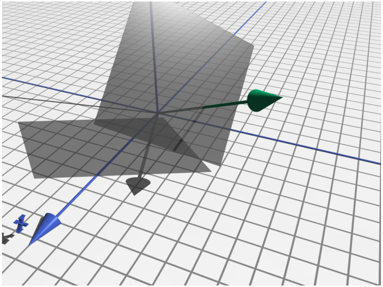
Data & Hough Space of X-Y data

Data Space (2-dimensional):

- Riemann transformation:
  $\mathbf{(X, Y) \longrightarrow (x', y', z')}$
  $(x'^2 + y'^2 + z'^2 = 1)$
- All $\vec{x}'$ fullfill condition:

$$\vec{n} \cdot \vec{x}' = c \qquad (1)$$

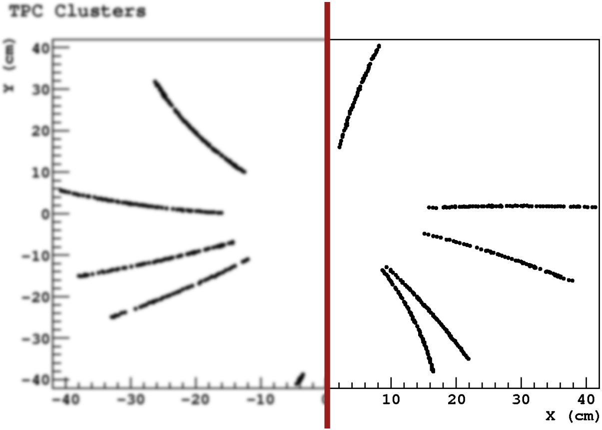
- Plane fully param. by $\vec{n}$ and c
- Since $|\vec{n}| = 1$, there are **3 parameters** in polar coordinates: $\mathbf{\Theta_n, \Phi_n, c}$



→ Hough Space (Θ_n, Φ_n, c):

- $\vec{x}'$ **fixed**
- Equ. (1) describes a **hypersurface** $c(\Theta, \Phi)$ in this space
- This corresponds to the set of all possible planes through point $\vec{x}'$

Hit Selection

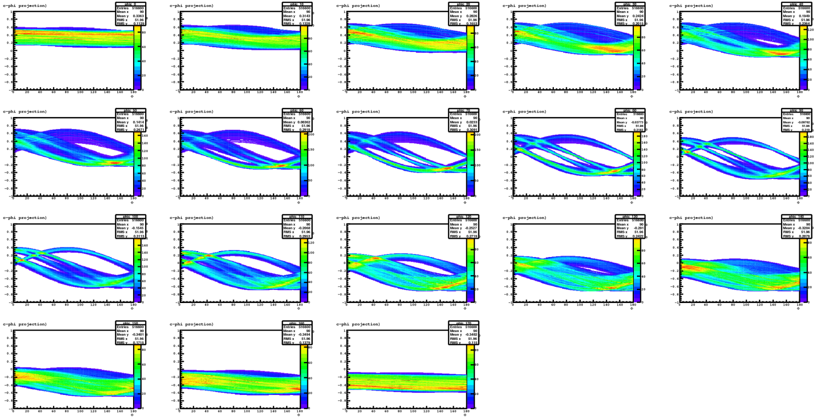


Example event: Clusters on the pad-plane

- To avoid ambiguities in Φ in the Hough Space all clusters with $X < 0$ are ignored.

X-Y / (Φ, Θ, c) Hough-Space of 5 Primary Tracks

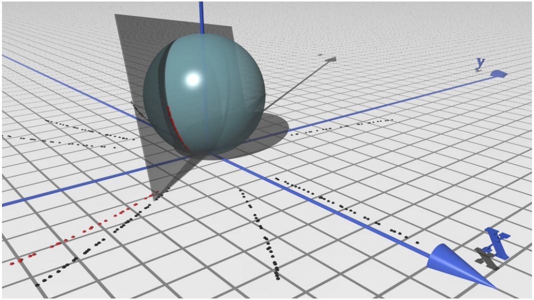
c (Θ, Φ) - density:



10 deg Θ-slices

Where to look for the Maxima?

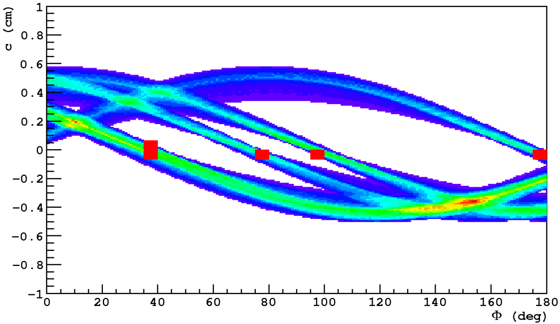
- In this example event all track projections (X,Y) are bent the same way
- We can expect their planes to be equally tilted with respect to Z
- Expect: $\mathbf{\Theta \approx 80\,deg}$



- Can be confirmed with MC data

Cross-check in $c\,(\Phi\,|\,\Theta \in [80, 90])$

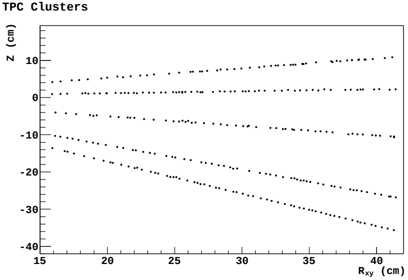
- Compare 3D Hough Space in the $\Theta \in [80, 90]$ deg - projection with calculated MC values $\mathbf{c\,(\Phi)}$



Calculated plane normal vector parameters from Monte Carlo data superimposed with Hough Space

What about the Time Information ?

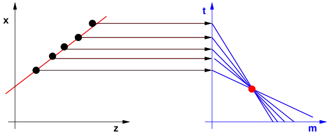
- So far the time (Z) information has not been taken into account
- Use relation $\mathbf{Z(R_{xy})}$ of the hits, where R is the **projected** radius in the readout-plane
- **Simplification:** $Z(R_{xy})$ is a straight line
- To avoid confusion: No Riemann transformation here!

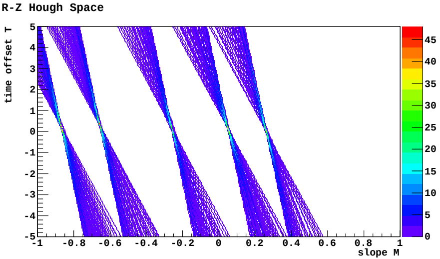


R-Z dependence of TPC clusters from primary tracks ($p_t \approx 1.0 GeV$ MeV)

Hough Space of R-Z dependence

- Simple 2D Hough Transform in this subspace
- **Z (R) ⟶ m (t)**





R-Z Hough Space

- Maxima of the 5 primary tracks visible

Summary

- Original TPC data consists of **3-dimensional** space-points (X,Y,Z)
- Track (helix): 5 parameters

- $\rightarrow$ **Complete Hough Space is 5-dimensional**
  1. Circle search in X-Y-plane
     $\xrightarrow{\textit{Riemann}}$ Plane search in the 3-dimensional space (c, Θ, Φ)
  2. Line search (for now!) in the 2-dimensional space (R, Z)
- Two adjoined Hough Spaces

The Problem of Maxima Detection

Task: Find Maxima in a **5-dimensional** space

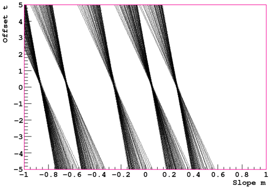
- Conventional method: Create a histogram and extract maxima
- High granularity needed: Say 500 bins for each dimension
  1 `int` (4 bytes) for each bin ⟶ **10 TB** size !!!
- Even with 100 bins the histogram size would be 40 GB
- Sparse histogram methods would reduce the size significantly, but filling alone takes far too much time

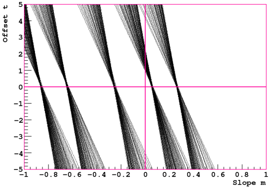
⟶ **Another method is needed**

Fast Hough Transform - a k-tree search

- Interpret parameter space as N-dimensional unit cube

- Iteratitive algorithm;
  in each step do:
  - Divide each cube (*node*) in the parameter space in 2^N subcubes with side length $\frac{1}{2^N}$
  - Count the number of **Hypersurface intersections** in each node and set as this node's *vote*
  - Each node with votes exceeding a certain *threshold* qualifies for further division in the next step
  - Discard all other nodes

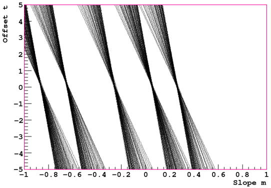




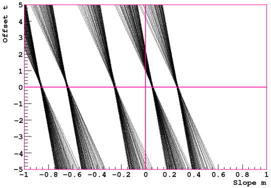
- Continue until a certain iteration depth is reached

Fast Hough Transform - a k-tree search

- Interpret parameter space as N-dimensional unit cube



- Each node knows:
  - Its **vote**
  - Its mother's vote
  - Its **hitlist**, a list of all hits which lie inside this node

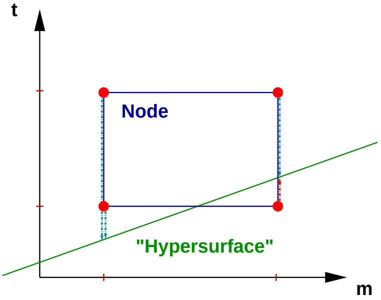


⟶ **A node is a track candidate**

- Continue until a certain iteration depth is reached

The Intersection Test

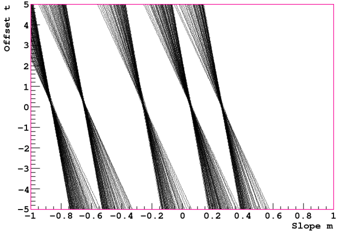
- One 5-dimensional Hypersurface consists of
  - One 3-dimensional Hypersurface $\mathbf{c} = \mathbf{c}(\Theta, \Phi)$
  - One 2-dimensional Hypersurface $\mathbf{Z} = \mathbf{Z}(\mathbf{R})$
- The two tests are **independent**
- The check itself:
  - Evaluate difference of node corners and hypersurface values at the corresponding coordinates
  - Scan for **sign changes**
  - **Sign change** → intersection



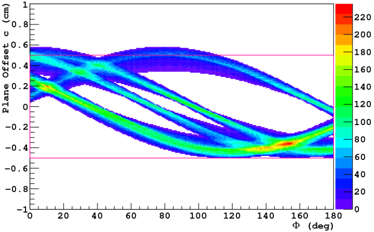
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

1 iteration



(m,t) Space

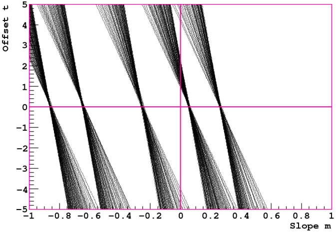


(c, Θ, Φ) Space
Projection

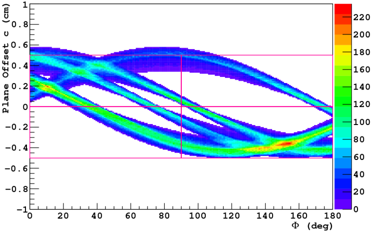
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

2 iterations



(m,t) Space



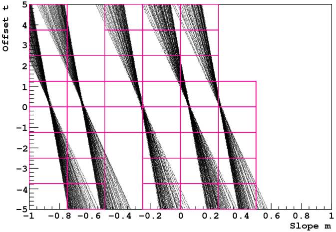
(c, Θ, Φ) Space

Projection

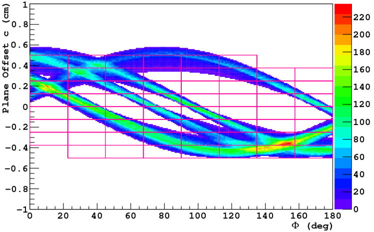
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

4 iterations



(m,t) Space

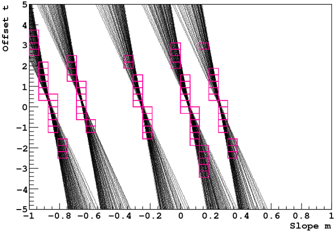


(c, Θ, Φ) Space
Projection

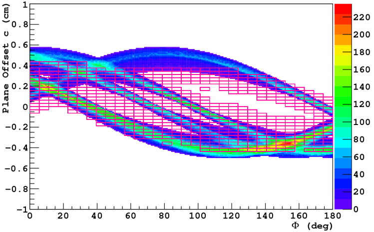
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

6 iterations



(m,t) Space

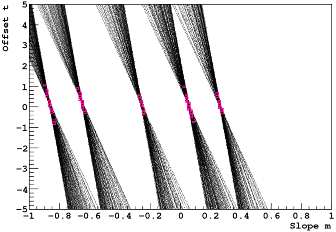


(c, Θ, Φ) Space
Projection

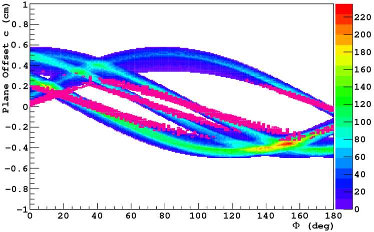
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

8 iterations



(m,t) Space

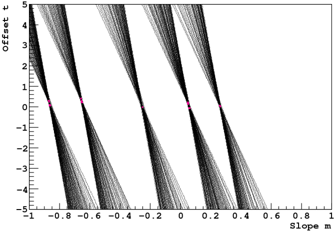


(c, Θ, Φ) Space
Projection

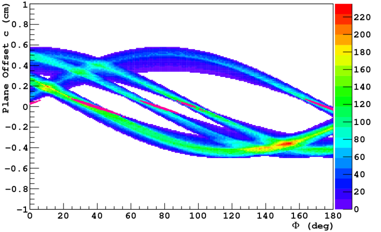
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

10 iterations



(m,t) Space

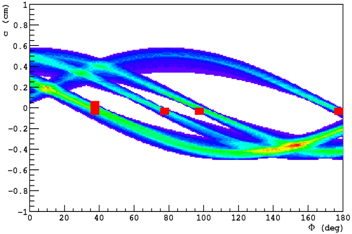


(c, Θ, Φ) Space
Projection

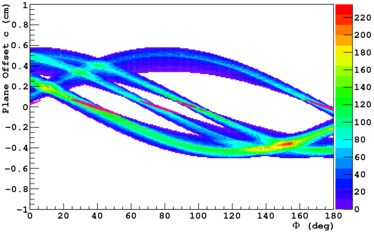
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

10 iterations



Monte Carlo Truth

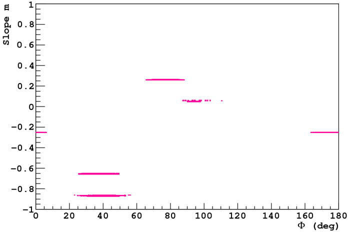


(c, Θ, Φ) Space
Projection

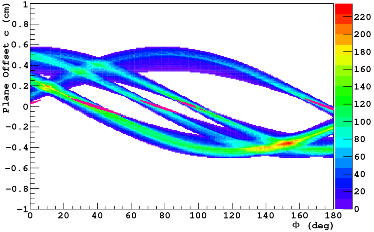
Fast Hough Transform - some nice plots

Evolution of nodes in two projections

10 iterations



(m, Φ) Space



(c, Θ, Φ) Space
Projection

Fast Hough Transform - Settings & CPU Performance

Parameter	Minimum	Maximum
Pl. Normal **Φ**	0	180
Pl. Normal **Θ**	20	160
Pl. Offset **c**	-0.5	0.5
RZ - Slope **m**	-1	1
T. Offset **t**	-5	-5

- Hardware: Intel Core2 Quad Q8400 (2.66 GHz), 4GB RAM
- Software: Ubuntu 8.10

- Memory usage: **20 MB**
- Execution time (8 iterations): **55 seconds!!!!** (1 core)

We have to do better!

Parallelism of the Problem

The problem is highly parallel on the computation level:

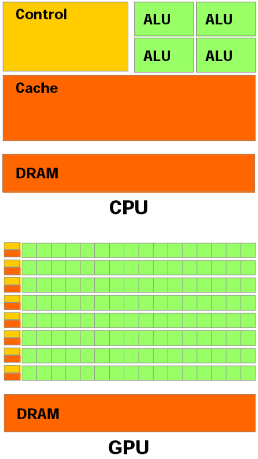
- Each cluster is completely independent
  $\longrightarrow$ each **Riemann-Transform** is completely independent
  $\longrightarrow$ each **hypersurface** is completely independent
- Each **node** is completely independent
  $\longrightarrow$ each **intersection test** is completely independent

- Level of parallelism is very high: Typically more than **10000** nodes are active $\longrightarrow$ nunber of potential threads is very high

$\longrightarrow$ Very well suited for an implementation on a GPU

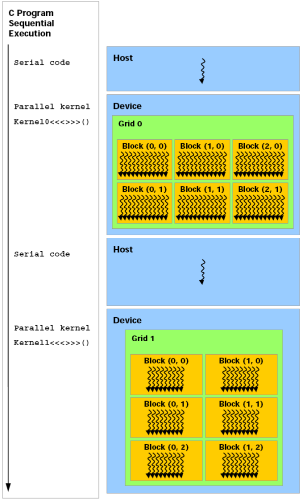
The CUDA Programming Model - an Extension to C

- Hardware optimized on calculations rather than caching and flow control
- Minimal set of instructions extending C
- GPU code is compiled by nVidia's `nvcc`
- **"Heterogenous" computing:** Both CPU and GPU do what they do best



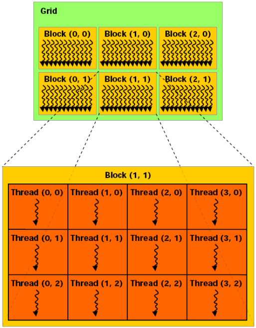
The CUDA Programming Model - an Extension to C

- Hardware optimized on calculations rather than caching and flow control
- Minimal set of instructions extending C
- GPU code is compiled by nVidia's `nvcc`
- **"Heterogenous" computing:** Both CPU and GPU do what they do best



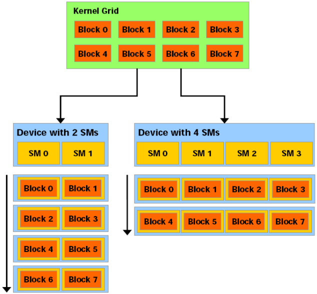
The CUDA Programming Model - an Extension to C

- GPU code runs in so called kernels:
  - A *Grid* of
  - *Blocks* , consisting of
  - a (large) number of *threads*
- **SIMT** model (single instruction multiple threads)
- **No thread-to-thread communication**
- Distributed to the device's *Streaming Multiprocessors*
- Smallest unit of execution: **32 threads** (one *warp*)
  - **AVOID BRANCHING INSIDE A WARP**
- Rather complicated memory structure



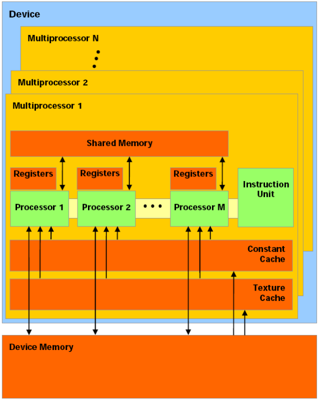
The CUDA Programming Model - an Extension to C

- GPU code runs in so called kernels:
  - A *Grid* of
  - *Blocks* , consisting of
  - a (large) number of *threads*
- **SIMT** model (single instruction multiple threads)
- **No thread-to-thread communication**
- Distributed to the device's *Streaming Multiprocessors*
- Smallest unit of execution: **32 threads** (one *warp*)
  - **AVOID BRANCHING INSIDE A WARP**
- Rather complicated memory structure

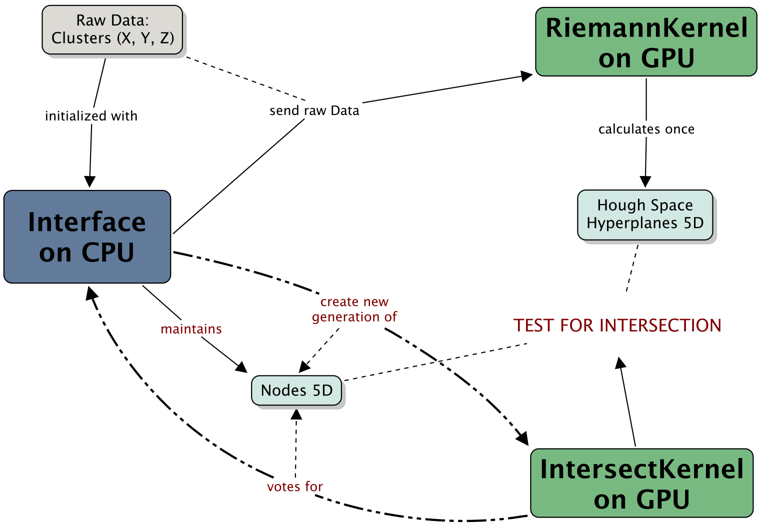


The CUDA Programming Model - an Extension to C

- GPU code runs in so called kernels:
  - A *Grid* of
  - *Blocks* , consisting of
  - a (large) number of *threads*
- **SIMT** model (single instruction multiple threads)
- **No thread-to-thread communication**
- Distributed to the device's *Streaming Multiprocessors*
- Smallest unit of execution: **32 threads** (one *warp*)
  - **AVOID BRANCHING INSIDE A WARP**
- Rather complicated memory structure



Current GPU-Project Design



Implementation of the FHT on the GPU

- GPU kernel code defines one thread's work: E.g. intersection tests

```
__global__ void
testIntersect(int nNodes, int level, int nClusters,
              ..., uint* votes) {

    int tID = blockIdx.x * blockDim.x + threadIdx.x;

   /* Loop over all clusters / hypersurfaces */
    for(int n=0; n < nClusters; ++n) {
      /* MAIN WORK HERE: Intersection tests */
      setBit(hitlist); vote();
    }

}
```

- Interface class `fastHoughGPU_IFC` takes pandaroot data and performs GPU CUDA calls

Implementation of the FHT on the GPU

- CPU still maintains **node objects** for easy use
- Each node knows its vote and its **hitlist**
- A hitlist is a bit-field with a "1" for every Hypersurface / cluster that hit this node
- The GPU calculates the global hitlist (for all active nodes) in each step

- In each loop iteration the list of active nodes can be processed conveniently in the CPU code
- This convenience comes at the price of a lot of memory copies

Track extraction

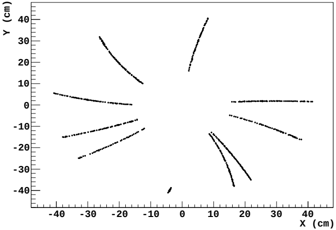
- Very nice feature of this algorithm: **Easy track extraction**
- After FHT is finished, just do iteration:

  while (vote $>$ some minimal number of hits per track **T**)

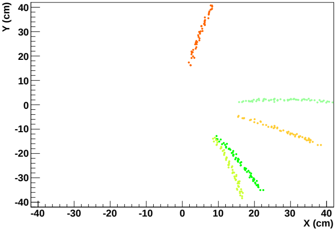
  - Sort all surviving nodes by their votes
  - Get the node with the **highest vote** of all survivors
  - This node's *hitlist* defines the best track
  - Remove these hits from *every node*
- This automatically gives back a list of all track candidates with at least **T** hits
- Next slides: 2 examples with T $=$ 5 (!)

Track Extraction Examples

- **Remember:** I only use clusters with X $>$ 0!



Original clusters

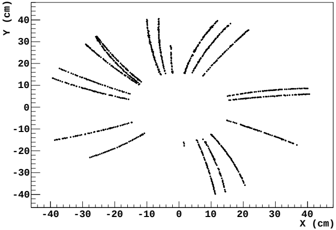


Reconst. tracks **in chamber 1**

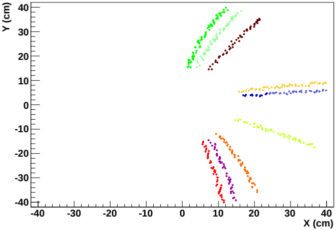
- Automatic extraction of n best nodes with more than $\mathbf{T = 5}$ votes
- Very nice result

Track Extraction Examples

- **Remember:** I only use clusters with X $>$ 0!



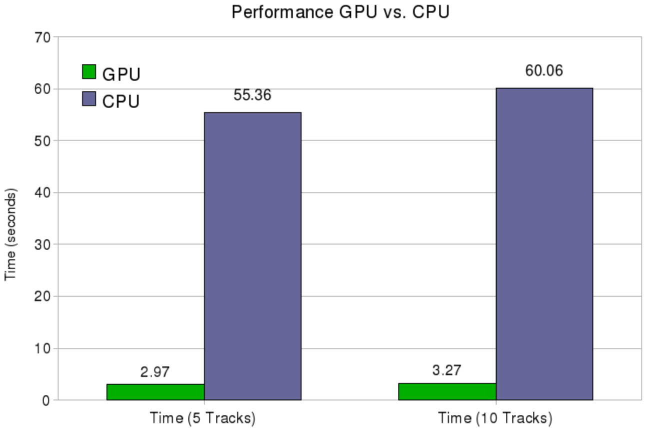
Original clusters



Reconst. tracks **in chamber 1**

- Automatic extraction of n best nodes with more than $\mathbf{T = 5}$ votes
- Very nice result

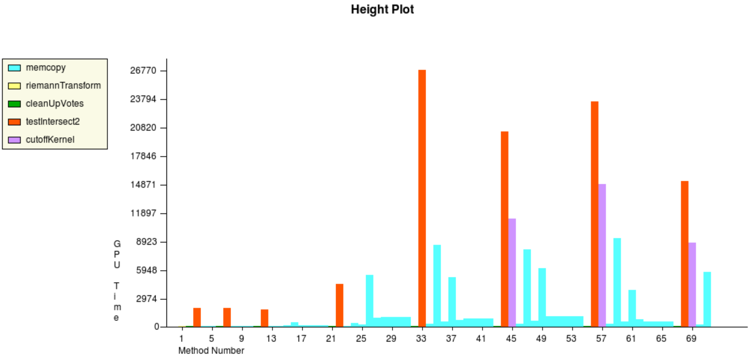
CPU & GPU compared



Hardware:

- Intel Core 2 Q8400 (2.66 GHz, single core mode), 4GB RAM
- nVidia GTX 285: 30 SM (8 ALUs each), 2GB RAM, clock 1.48 GHz

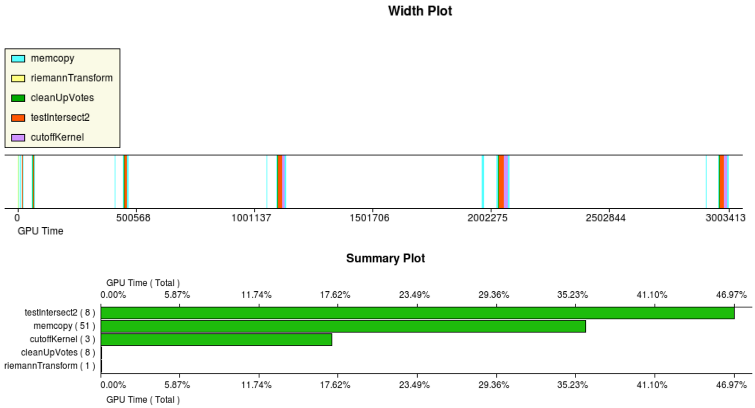
Some Profiler Output



Kernel Time Evolution:

- Exp. growth, saturation and decline of the number of **active nodes** visible
- IntersectionKernel consumes biggest fraction of the GPU time

Some Profiler Output



- Over 90 % of the *total time* is still spent on the CPU!
- CPU data maintainance & copying is a considerable overhead the limiting factor
- **This can be moved to the GPU for a significant speed boost**

Some Profiler Output II

Profiler Output

	Method	GPU Time	CPU Time	Occupancy	registers per thread	branch	divergent branch	instructions	cta launc
10	memcopy	28.704	49						
11	cleanUpVotes	3.552	21	1	2	0	0	0	1
12	testIntersect2	25509.9	25525	1	8	0	0	0	1
13	memcopy	5.024	18						
14	memcopy	23.2	54						
15	memcopy	82.528	640						
16	memcopy	438.752	110						
17	memcopy	111.552	101						
18	memcopy	111.616	104						
19	memcopy	111.552	103						
20	memcopy	111.488	105						
21	cleanUpVotes	6.336	23	1	2	64	0	496	4
22	testIntersect2	74196	74211	1	8	116268	0	1834551	14
23	memcopy	25.376	1039						
24	memcopy	351.872	682						

- **No divergent branching**
- In terms of algorithm logic we reach **100 % efficiency**

Present GPU code Optimizations

1. Node Cutoff (GPU only)
    - Only a certain fraction of the 32 sons is kept to reduce # Nodes
2. Relative Thresholding (CPU & GPU)
    - Instead of global threshold use a fraction of **mother's threshold** for each son
    - Converge faster on the maxima
3. **Hitlist Intersection Testing** (CPU & GPU)
    - Only test for intersection if mother node was hit by this hypersurface
    - Under optimal conditions only $\frac{1}{\# Tracks}$ of the intersection tests have to be done
      $\rightarrow$ **SCALING!!!**
    - 32 sons of a mother read the *same* hitlist
      $\rightarrow$ **No branching!** (*1 warp* = **32** *threads*)

Optimization Wishlist

1. **Most important:**
   - Control the strong dependence on algorithm parameters!
2. **Improved Dynamic Thresholding**
   - Find a way to reliably switch on dynamic thresholding track-wise
3. Find more effective ways of node-purging between iteration steps
   - Further reduce # of active nodes (esp in (c, Φ, Θ)-space)
4. General:
   - Start moving tasks from CPU to GPU
   - Reduce maintainance overhead

Conclusion

Summary:

- Implemented 5-dimensional Hough-Transform for helix track-finding
- General track parametrisation $\longrightarrow$ no constraints
- Fast Hough Search for maximum detection
- Massively parallel implementation on GPU (nVidia CUDA)
    - **Performance gain at least factor of 20**
      (Benchmark of first version **including** CPU & memcopy overhead: 3 seconds for 10 tracks)
    - Very nice scaling behaviour (# of tracks, # of hits)

- General remarks on GPU implementation
    - It's **just C!**
    - CPU & GPU code almost identical

Outlook

- Long term goal: Move everything to the GPU
- Characterize algorithm parameter dependencies
  - Efficiency, stability, performance
- Further optimize GPU code:
  - Exploit **shared / texture memory**
  - More aggressive node management
- Find strategies to deal with mixed events
  - Example: Multiple passes
    - Primary cut: tight cut on c & Θ
- Apply this to
  - Physics channel
  - Mixed events (TPC)

BACKUP SLIDES

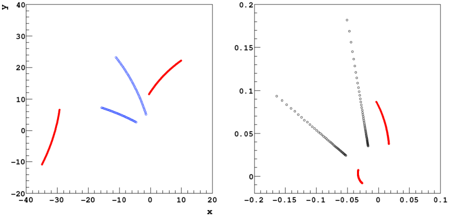
Conformal vs. Riemann Mapping

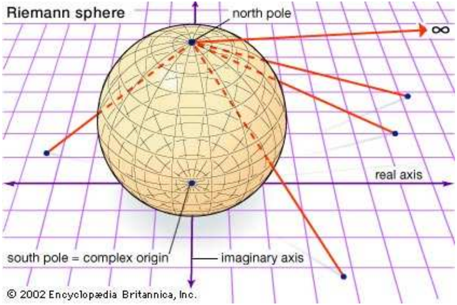
Conformal Mapping

NIM A380(1996)582

$$x' = \frac{x-x_0}{r^2}, \quad y' = \frac{y-y_0}{r^2}$$

- Reference point (x_0, y_0)
- Different transformations





Stereographic Projection

$$x' = R \cdot \frac{\cos\Phi}{1+R^2} \quad y' = R \cdot \frac{\sin\Phi}{1+R^2} \quad z' = \frac{R^2}{1+R^2}$$

- No reference point needed
- Primaries and secondaries treated equally
- Circles in XY-Plane → planes on sphere