Introduction

- LHCb is a precision beauty and charm experiment at LHC
- Precision measurements are compute and I/O intensive:
  - large datasets
  - statistical tests and systematic studies with toy monte-carlo
  - complicated fits
- ROOT is an **integral component** of every analysis at LHCb
- This talk will attempt to describe how analysts at LHCb use ROOT
A typical LHCb analysis

- This title is an oxymoron - no analysis is typical.
- There are as many ways to obtain and analyse data at LHCb as there are analysts.
- There are a few common themes however:
  - Generally LHCb deals with precision measurements of parameters accessible though known decays, eg:
    - "Cut-and-count" direct-$CP$ measurements
    - Time-dependent $CP$ measurements
    - Rare decay searches
  - In nearly all cases the final results are obtained from studying an offline TNtuple...
A typical LHCb analysis: Getting to ntuple

- The LHCb software consists of several projects built on the GAUDI framework
- GAUDI depends on ROOT (See Ben’s talk this afternoon), allowing analysts to make use of ROOT objects
- Each project serves a specific task:

  Collision data

  - Trigger
  - MOORE
  - Analysis
  - DAVINCI

  Raw hits
  Tracks/Protoparticles
  Trigger decisions

  Composite particles, Vertices, lifetimes

  To TFile

  Monte-Carlo

  - Simulation
    - GAUSS
  - Digitisation
    - BOOLE

  Analysts use DaVinci to create composite particle decays, apply selections and choose what quantities they write to ntuple

  DaVinci allows users to run arbitrarily complicated algorithms on reduced/skimmed/stripped datasets, eg: TMVA-based selections are common

  Most of the ‘hard work’ is done here reducing event information into a few specific properties of selected particle decays

  This means the offline ntuple is very simple. Typical information stored in the ntuple:
    - 4-vectors, masses, vertex positions + errors, fitted lifetimes etc saved as float branches for each particle in the decay
$\phi_s$ in $B^0_s \rightarrow J/\psi \phi$

- One of the flagship analyses at LHCb \cite{LHCb-CONF-2012-002}
- $\phi_s$ is a $C\mathcal{P}$-violating phase extracted from a fit to the lifetime of $B^0_s$ mesons to the $J/\psi \{\mu^- \mu^+\} \phi \{K^+K^-\}$ final state.
- Simultaneous, multidimensional fit to three angles, lifetime over 3 tagging categories and different tagger and trigger types
- Approx 28000 signal candidates in $1\text{fb}^{-1}$ of 2011 LHCb data, but toy studies run up into the millions
- Ntuple contents: $B^0_s$ mass, $\phi$ mass, lifetime, flavor tag, per-event mistags for several tagging categories, 3x angles
Three groups work on the $\phi_s$ analysis, each using different fitting technologies:

- Two groups write their own compiled C++ fitters that interface to MINUIT and make heavy use of ROOT.
- One group uses RooFit with a custom-written PDF.

The three groups operate independently and allow for a powerful cross-check of the code implementation.

In the end, all three groups agree across all the physics parameters at the permille level.

I’ll focus on one particular fitter and their experiences with ROOT:
RapidFit

- RapidFit started out as a way to perform several classes of similar fit in a straightforward manner:
  - C++ binary compiled against ROOT for local use, ROOT library used for grid submission: Run anywhere ROOT is installed.
  - XML control files to steer the fit containing Parameter ranges, names, fit type, etc.
  - PyROOT used to pass runtime arguments when sent to the grid.
  - Simple classes containing useful, speed-optimised functions commonly used in a PDF: Commonly used integrals, etc.

- Main libraries used from ROOT:
  - TFoam used heavily for toy dataset generation
  - TMinuit, TMinuit2 for the minimisation
  - Math::AdaptiveIntegratorMultiDim, Math::IntegratorOneDim for normalisation
  - TTree for input data and to store toys
  - TH1, TGraph, TGraphErrors to plot results

- a handful of other Math:: functions used as well.
Profiling/Optimisation

- RapidFit was extensively profiled and optimised:
  - Single fit to data to get central value takes approx 8 hours on a single core
    - 28000 datapoints: PDF evaluated once per MINUIT call per datapoint
    - Several calls to normalisation method required when acceptance functions are included.
    - PDF is cached extensively as little changes within each call.
    - 600-1000 Minuit calls per fit
    - $5 \times 10^7$ calls to the PDF normalise/evaluate methods for a single fit
  - pthreads now used for single fit, speedup is 8 on an 8-core machine. Lack of thread-safety in the original code required heavy rewriting
Feldman-Cousins

- Rapdfit is also used to make Feldman-Cousins 1D and 2D contour scans for the $\phi_s$ analysis:
  - thousands of toy datasets generated of equivalent size to the real data at thousands of discrete points on a 1D or 2D plane
  - Each toy dataset fit to twice
  - Confidence belts determined from likelihood ratio of toys at each datapoint.

- This is extremely compute intensive, but "Embarassingly parallel":
  - Toys generated at one point in phase space are independent of those elsewhere, so jobs can be divided up and sent to the grid
  - Toy fit results stored and saved in ntuple format, merged and processed locally later.

- The full FC scan for the last publication consumed about 2.5 CPU years on the grid.
RapidFit summary

- The analysts who worked on RapidFit found a lot of the built-in ROOT functionality invaluable:
  - I/O: The speed, portability and merging functionality of ROOT’s data formats
  - TFoam and integrator methods were fast and accurate
  - Ability to ship Rapidfit as a ROOT library made grid deployment straightforward

- They encountered one or two issues along the way too:
  - Memory leak checking turned up a lot of issues "upstream" in the ROOT libraries
  - The analysts were unaware of the existence of PROOF or PROOF-lite and went the way of pthreads
  - Persistency with TFoam caused some problems: Calling delete didn’t free memory, re-using an existing instance required heavy code modification
  - Customising plots took a significant amount of time
The LHCb ROOT User survey

- Hopefully the last slides gave you an idea of how ROOT was used in a specific analysis
- To get a broader feel I’ve conducted a surveymonkey poll of LHCb analysts
- I got 55 respondents, which covers a reasonable slice of the collaboration involved in analysis tasks
- The survey consists of 10 questions, of which the earlier ones gauge experience and the last two are open comments
- I’ve loosely summarised the comments by picking out common themes
**Question 1: Experience**

What is your level of programming experience? Select the answer that best describes you:

- Answered: 55  Skipped: 0

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>I am very experienced: I have more than 5 years experience in at least 3 programming languages, I know what Object-Oriented code is, and how to write it. I profile my code. I write code that can be reused by myself and others easily, and I keep it maintained.</td>
<td>21.82%</td>
</tr>
<tr>
<td>I am experienced: I know a couple of programming languages well, I know what object oriented code is. I frequently re-use my own code with minor modifications.</td>
<td>58.18%</td>
</tr>
<tr>
<td>I am less experienced: I have taken some basic programming courses, but most of my understanding comes from ROOT tutorials. I write code/scripts to get results and copy and paste code snippets in order to reuse them.</td>
<td>20%</td>
</tr>
</tbody>
</table>

Total: 55
**Question 2: PAW**

**Did you or do you ever work with PAW?**

- Answered: 55  
  Skipped: 0

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes, and I prefer PAW</td>
<td>10.91%</td>
</tr>
<tr>
<td>Yes, and I prefer ROOT</td>
<td>14.55%</td>
</tr>
<tr>
<td>Yes, and ROOT is about the same as PAW</td>
<td>5.45%</td>
</tr>
<tr>
<td>No, I've never used PAW</td>
<td>69.09%</td>
</tr>
</tbody>
</table>

Total: 55
Question 3: Hardware

Where do you most commonly use ROOT when performing intensive analysis tasks, such as toy studies or fitting?

Answered: 48   Skipped: 7

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>On lxplus</td>
<td>27.08%</td>
</tr>
<tr>
<td>On my laptop/desktop</td>
<td>54.17%</td>
</tr>
<tr>
<td>On a batch system</td>
<td>16.67%</td>
</tr>
<tr>
<td>On the grid</td>
<td>2.08%</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
### Question 4: Usage preference

What ways do you commonly interact with ROOT? Rank these options according to highest usage:

<table>
<thead>
<tr>
<th>Usage</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
<th>Rank 4</th>
<th>Rank 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiling C++ binaries</td>
<td>39.62%</td>
<td>18.87%</td>
<td>26.42%</td>
<td>9.43%</td>
<td>20.75%</td>
</tr>
<tr>
<td>Using the TBrowser</td>
<td>5.66%</td>
<td>35.85%</td>
<td>26.42%</td>
<td>18.87%</td>
<td>13.21%</td>
</tr>
<tr>
<td>Writing ROOT macros</td>
<td>24.53%</td>
<td>18.87%</td>
<td>26.42%</td>
<td>9.43%</td>
<td>20.75%</td>
</tr>
<tr>
<td>Using CINT</td>
<td>5.66%</td>
<td>13.21%</td>
<td>22.64%</td>
<td>43.40%</td>
<td>15.09%</td>
</tr>
<tr>
<td>Writing PyROOT scripts</td>
<td>24.53%</td>
<td>13.21%</td>
<td>15.09%</td>
<td>9.43%</td>
<td>37.74%</td>
</tr>
</tbody>
</table>

- Interpretation of this table is tricky, as correlations aren’t included
- Generally, LHCb users prefer to compile their analysis code to get better debugging information from GCC
- While PyROOT comes last, it’s the more experienced analysts that tend to use it
### Question 5: Parallelism

Do you make use of multiple threads or parallel processing when writing code? Select any of the following that apply to you.

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>I use PROOF to write parallel code for batch submission</td>
<td>7.69%</td>
</tr>
<tr>
<td>I use PROOF-lite to write multithreaded code to run on my multi-core desktop/laptop</td>
<td>5.13%</td>
</tr>
<tr>
<td>I use the &quot;nCPU&quot; option in ROOFIT to speed up my fits</td>
<td>74.36%</td>
</tr>
<tr>
<td>I write my own multithreaded desktop/laptop code without the use of PROOF/PROOF-lite/nCPU</td>
<td>17.95%</td>
</tr>
<tr>
<td>I write my own batch-submitted, single-threaded analysis code and combine the results myself</td>
<td>41.03%</td>
</tr>
</tbody>
</table>

Total Respondents: 39
Question 6: Dataset size

When fitting to data, toys or MC, what size of ntuple do you generally work with?

Answered: 53  Skipped: 2

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1GB</td>
<td>18.87%</td>
</tr>
<tr>
<td>&lt;5GB</td>
<td>47.17%</td>
</tr>
<tr>
<td>&lt;10GB</td>
<td>18.87%</td>
</tr>
<tr>
<td>&gt;10GB</td>
<td>15.09%</td>
</tr>
</tbody>
</table>

Total 53
Question 7: Basic fits

How do you perform simple fits? (Example: Fitting a gaussian to a peak)

Answered: 54  Skipped: 1

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>I use the FitPanel in the ROOT TBrowser</td>
<td>11.11%</td>
</tr>
<tr>
<td>I use ROOFIT with the built-in basic PDFs</td>
<td>66.67%</td>
</tr>
<tr>
<td>I use ROOFIT with a custom-written PDF</td>
<td>7.41%</td>
</tr>
<tr>
<td>I use MINUIT and write the rest myself</td>
<td>14.81%</td>
</tr>
<tr>
<td>I use neither ROOFIT nor MINUIT</td>
<td>0%</td>
</tr>
<tr>
<td>Total</td>
<td>54</td>
</tr>
</tbody>
</table>
How do you perform complex fits? (Example: Multidimensional fits with acceptance functions, convolutions, several signal/background components)

Answered: 53  Skipped: 2

<table>
<thead>
<tr>
<th>Answer Choices</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>I use the FitPanel in the ROOT browser</td>
<td>0%</td>
</tr>
<tr>
<td>I use RooFit with the built-in...</td>
<td>24.53%</td>
</tr>
<tr>
<td>I use RooFit with a custom-written PDF</td>
<td>56.60%</td>
</tr>
<tr>
<td>I use Minuit and write the rest myself</td>
<td>16.98%</td>
</tr>
<tr>
<td>I use neither RooFit nor Minuit</td>
<td>1.89%</td>
</tr>
<tr>
<td>Total</td>
<td>53</td>
</tr>
</tbody>
</table>
Question 9: Dislikes

What do you dislike about ROOT? Please list any problems or issues you’ve encountered.

Common responses:

- Inheritance structure/OO is unnatural, eg: TH1 has 3 dimensional methods
- Overuse of Run-time type information and use of raw pointers makes debugging difficult
- Objects require unique names, lots of stuff happens "under the hood" with global pointers killing encapsulation
- Passing options as strings causes many hard to locate bugs
- CINT prevents easy debugging/profiling, often randomly (and recursively!) segfaults, isn’t fully compliant
- Limited STL integration (although ROOT predates STL and support has improved!).
- Lack of formatting simplicity: No auto placement of labels/legends, lots of work to make plots suitable for journals.
What do you like about ROOT? Please list anything that you find particularly useful.

Common responses:

- Fully populated and feature-rich classes/libraries eg: Math, TLorentzVector, TRandom3 etc
- RooFit, TMVA: We are glad to see these bundled and well supported
- Efficient I/O and straightforward Tree/Ntuple access eg: 
  `TTree::Draw("variable","cuts")`
- PyROOT is loved (by those who know it)
- Helpful and generally well documented class reference
- Wide range of tutorials (though we’d like to see more of them in compilable C++)
Conclusions

- LHCb uses ROOT for all analyses
- There are many ways to do the same thing with ROOT, and every analyst will have a different preference
  - Compiled C++ binaries are preferred to CINT macros, but more people are moving to PyROOT
  - The majority of analysis tasks are performed on laptops/desktops
  - ROOT's exceptional I/O performance is increasingly important at the analysis level: Datasets are ever-expanding
  - Not many analysts are aware of PROOF-lite, although many have multi-core laptops
- LHCb’s analysis ‘wish-list’ for the future of ROOT:
  - We would like to see continued/enhanced support for ROOT-based applications, particularly ROOFIT
  - Continued PyROOT integration and support is encouraged
  - The Gaudi framework will take time to incorporate ROOT 6. We would like to see backports of bugfixes and continued support of ROOT 5
  - We would appreciate simpler, more automated formatting tools for plots
  - We find the inheritance structure, use of globals and some OO decisions in ROOT to be unintuitive
- We look forward to the many improvements and additions in ROOT 6!