CERN Summer Student Report

Mathias Luidor Heltberg

August 22, 2014

Abstract

During my stay at CERN I have worked on the ATLAS detector, in the luminosity group, under the supervision of Eric Torrence. Throughout the program, my project has focussed on working with the actual scripts for storing the online data from ATLAS, and making ready for the start of RUN 2.
I have succeded in making different programs work for different storage modes of the online data, and comparing the output of these to the results for run 1, it should be possible to implement the for the dataanalysis for Run 2.

Background and Theory

Precise determination of luminosity is one of the key features in the ATLAS physics programme, and in order to determine cross sections with high precissions, it is crucial to determine the luminosity with high precission, since the uncertainty on the luminosity is one of the major systematic uncertainties.
In general we want to determine the cross section, which describes the probability for at certain process, and what we measure is a number of observed processes. The relation between the cross section and the number of observed events is proportional given by:

$$\sigma = L \cdot R \tag{1}$$

where L is the luminosity, and this describes the number of interactions we create per time. This can be determined by the beam parameteres, and the luminosity can be described by:

$$L = \frac{n_b f_r n_1 n_2}{2\pi \Sigma_x \Sigma_y} \tag{2}$$

Here n_b and f_r are the number of bunch pairs colliding per revolution and the revolution frequency respectively, while n_1 and n_2 are the protons per bunch in beam 1 and 2. The Σ in the denominator describe the horizonthal and vertical convolved beam width.
In the data analysis we are allways interested in the total number of created collisions insted of the number of collisions per time, and therefor we calculate the integrated luminosity $\int Ldt$. The Luminosity is not constant over the entire run,

and start with a peak luminosity after the beams have been filled, and drops thereafter exponentially until a critical value is reached and the beams are dismissed.
The Luminosity is measured by the two detectors LUCID and BCM, and different parameters for the luminosity measurements is implemented to the online datafolder. In measuring the Luminosity, the different bunches is labeled by a BCID number (1-3564), that describes the placement of the bunch is the 25 ns spacings. The online data for the detectors is stored in the COOL database, where it is saved in different folders.

Working with online to offline algorithms

My first job was to work with the python script OflLumiMaker.py. In this script the online data from a specific run were copied, and analyzed and copied to an offline folder via the function CalculateOfflineLumi. This was a my introductory work, where it was my task to get a feeling with the way the script was structured. In this process I should furthermore try to implement some switches so the user could easily specify the choice of channels and starting time for the offline making.
It took some time to get to know the script, especially beacuse there were many different functions, I needed to understand. I the end I managed to implement several different switches to my own version of the script, but we decided that it would be of more use, if I spent my time on a second task, which was to write functions in C++, which was actually used in the online data storage.

Working with the Util programs

In the CoolOnline data there are 3-4 folders that are used in the calibration for storing data, and these needs to be unpacked, which we decided were my major task for the summer student project. The folders I should try to work with were:

/TRIGGER/LVL1/Bunch GroupContent	It is in this folder the BCIDs that ATLAS is triggering on are stored, which are important when calculating the total luminosity
/TDAQ/OLC/LHC/BUNCH DATA	It is in this folder the beam intensities per BCID is stored, and the data is stored for each beam separately
/TDAQ/OLC/BUNCHLUMIS	It is in this folder the raw uncalibrated data for the luminosity is stored. This means that it is the data from this folder we want to calibrate, and in the end apply

The programs for unpacking the data correctly should be written as a class in C++, but in such a way that it was able to work in Python scripts including the PyCIntex module, that is based on PyROOT. This was a very challenging task in the beginning, since I hadn't been working much with C++ before, but here I had a great opportunity to learn it. The data in these folders were all stored in blobs

(Binary Large Object) which is a data type that can store big amounts of binary data, and my first task was to get comfortable with this type. To read data of this type we needed a pointer, to read objects of special length, which was defined for each specific folder. // As a sketch I had a similar program for unpacking data in the folder /TDAQ/OLC/LHC/FILLPARAMS, in which the data for which BCID's that have a beam in them is stored. This program was written from my supervisor, and with this as a model I should be able to create the programs to do the correct unpacking of the data in the other folders.

Storing the Data

To begin with I worked on the script storing data in the BunchGroupContent. This was very similar to the one written by my supervisor, and I could soon read the blob and print out the data. I then met the problem that the numbers I actually read didn't match what was expected. This was solved when I changed the pointer to read datasizes of 1 byte instead of 2 bytes, and specified that we needed to read the data as an integer (and not as a character which is standard for 1 byte data). Now I could read contents of each BCID and the result was the following:

Bunchgroup	**Number**	**Meaning**
0	183	?
7	114	Collisions
6	70	?
5	390	?
4	6	Only one of the beams are filled
3	6	Both beams are filled, but no collision
2	1520	Background
1	1254	Collisions
-	21	No data

The interpretation of these numbers is important, since it is important to know which BCID's actually collided during a specific run. The Bunchgroup coloumn represents the separated data where each group has a special definition. The BCIDs is separated by the Central Processor Trigger, using combinatorical logic, and this separation into groups, is important in order to know for instance which BCIDs are empty, which are colliding and which are unpaired, which can be seen for instance in the collision group above. The second column specifies the number of BCIDs in the specific Bunchgroup.
Having done this, and having got a bit of understanding for the structure of the C++ classes, I moved on to the folder Bunchdata, in which the intensities of the BCIDs were stored. This folder was a further challenge, since the data were stored in a special way, in order to minimize the amount of data written to the online database.
In the blob where the intensities were stored, the first byte should be read separately, and this defines in what way the rest of the data was stored, and the size of the data

in the blob. The size defined the size of the pointer, that should read the data, and how the data should be calculated:

$$BunchValue = \begin{cases} \frac{Value \cdot Average}{100} & \text{if } x = 1 \\ \frac{Value \cdot Average}{10000} & \text{if } x = 2 \\ Value & \text{if } x = 4 \end{cases} \tag{3}$$

The way the data is stored is seen in figure 1. The reason for storing the data in this way, is that for some luminosity blocks we have almost no collisions, in which case it doesn't make much sense to save all the data. The work with these storage

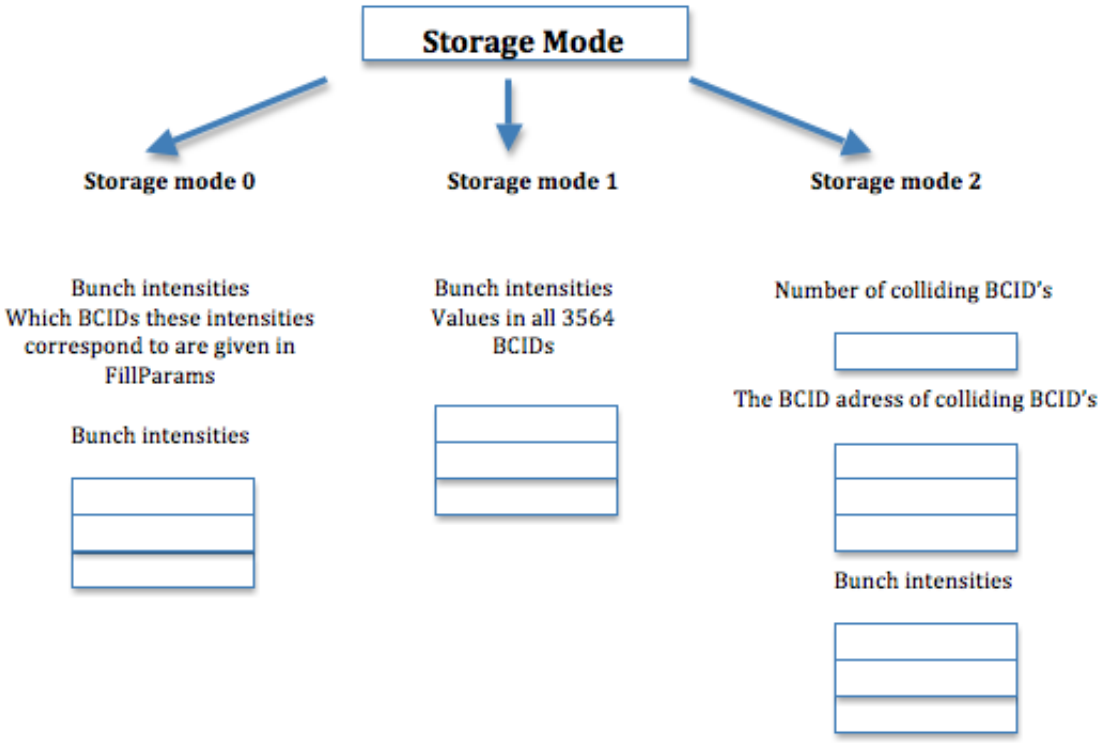

Figure 1: *Here is an overview of how the data is saved, in for different storage modes*

modes, and the definition of different pointers, took was challenging, and in the end I should for different runs compare the values I could get from my algorithms, with the values from old runs. This can be seen in figures 2 and 3.

Further Steps

Figure 2 shows the avarage intensities for all the luminosity blocks (372 in all) whereas figure 3 shows the stored data in the online database, each line representing the intensity for a BCID at different luminosity blocks. The reason for the cutoff at LB = 120, is that we do not want to include the data around the filling of the

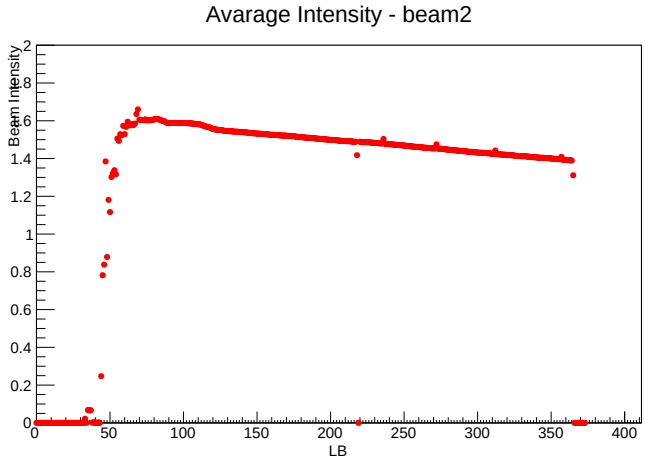

Figure 2: *Here is the intensities for beam 2, printed from my algorith*

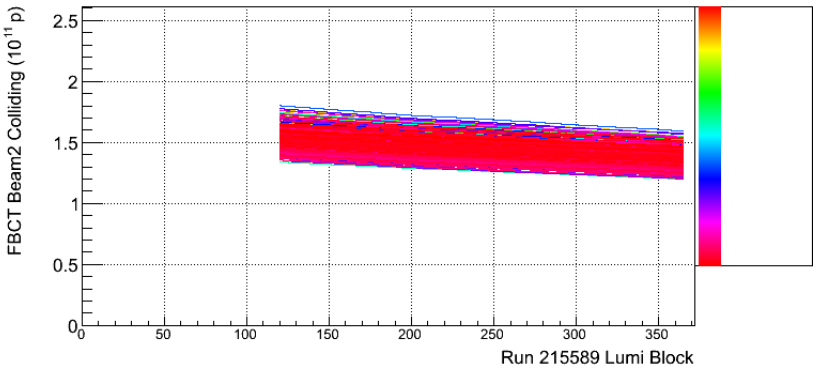

Figure 3: *Here is the corresponding data for the COOL database*

beam. What is important, is that the calculated avarage in figure 2, seems to agree with the results in figure 3.
Having implemented the algorithms for the BunchData and BunchLumis folders, and making these work in a similar way, the next step would be to make one script that could unpack the data, and could be called from both the BunchData and the BunchLumis. This would be efficient, since these two folders are both being unpacked in the way described above, and we would then only have to edit in one script, when improvements should be made for the scripts.
This was what I what worked on in the final week of my stay, and I this was done in the script named LumiBlobUtil. The way one calls a function from another C++ class, was all new to me, but it could be implemented for some parts of the functions in the two scripts. I made a function in the class LumiBlobUtil called unpack, which took a float and a blob as arguments. In this way I could specify the float and the blob in the classes BunchLumiUtil and BunchDataUtil, and then do the actual unpacking in the LumiBlobUtil class.

Concluding remarks

Throughout the summer, the process of working with the actual data from the online database, has been very exciting, and even though my skills in computer science were really put to a test, I have learned a lot, and in the end present some useful scripts.
In the beginning I worked with some algorithms written in python, which calculated the offline data, from the online data, but after two weeks I started on working with the unpacking of online data in C++. Then the majority of my work focussed on this unpacking of online data in different folders, and this turned out well, even though I would wish I could have finished even more and stayed here a bit longer.

Mathias Luidor Heltberg