JOINT INSTITUTE FOR NUCLEAR RESEARCH
Veksler and Baldin Laboratory of High Energy Physics

FINAL REPORT ON THE START PROGRAM

Optimisation of Decoding process in the BmnRoot software package

Supervisor: Ilnur Gabdrakhmanov

Student: Andrei Islentev

Participation period: 2 July 2023 - 11 August 2023

Contents

Abstract

This report discusses different optimisations which can be applied to the Conversion and the Decoding algorithms in *BmnRoot*. A new multithreading algorithm for the Conversion step for converting a small number of raw files is proposed. The significance of taking different pedestal events on the Decoding step is analysed. A new algorithm for finding noisy channels is developed and compared with the current algorithm in *BmnRoot*.

1 Introduction

The goal of the BM@N experiment is to investigate the behavior of matter at nuclear densities comparable to those at neutron stars. Such nuclear densities are achieved by accelerating particles to energies of 6 GeV and smashing them into a stationary target. As a result of these collisions, new particles, including various strange particles, are created. Hopefully, observing these particles would improve our understanding of strange matter.

1.1 Experiment setup

The process of particle acceleration in the BM@N experiment is involved. At first, newly created particles are accelerated by consecutive linear accelerators to energies of a couple of MeV. Then, they are passed to a booster where they are accumulated and accelerated to energies of hundreds of MeV. These patches, on average consisting of a few thousand particles, are called *spills*. The booster then passes the accelerated spill into a nuclotron. The nuclotron accelerates multiple spills at the same time to energies of 6 GeV per particle and directs spills that have reached the required energy into a tube which leads towards the target and detectors. Spills are discharged over a period of $5 - 10\,\text{s}$, giving the detectors about a millisecond to process one particle.

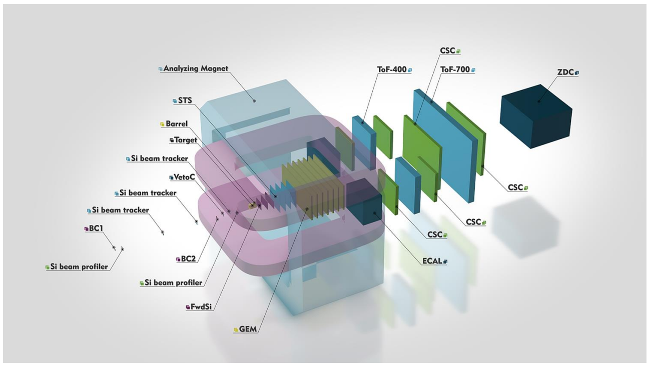

Figure 1: BM@N experiment detectors setup.

After entering the tube, at first, a particle encounters a group of detectors which notify the rest of the detectors that there is a particle incoming and check whether it is going to collide with the target. After hitting the target, the particle splits into many new particles which travel through a magnetic field. On their way, they pass through various *strip detectors*. Some strip detectors specialise in measuring particles' exact position and approximate time when passing the detector. Others operate vice versa: they measure the approximate position and exact time of passing the detector. All of the detectors up to this point were as thin as possible to not affect the energies of the particles. Finally, these particles collide with a thick lead calorimeter, which measures their energies.

The velocities and the mass-to-charge ratios of the particles can be measured after reconstructing their paths through the strip detectors. Together with the data from the calorimeter, this is enough to identify the mass and the charge of each of the particles.

1.2 Strip detectors

Strip detectors play a central role in reconstructing the trajectories of particles. There are various types of strip detectors used in the BM@N experiment, though they all have a lot of similarities. In the *BmnRoot* software package, this is represented by inheriting all strip detector classes from the same parent class *BmnAdcProcessor*. An oversimplified scheme of a strip detector is shown in **figure 2**.

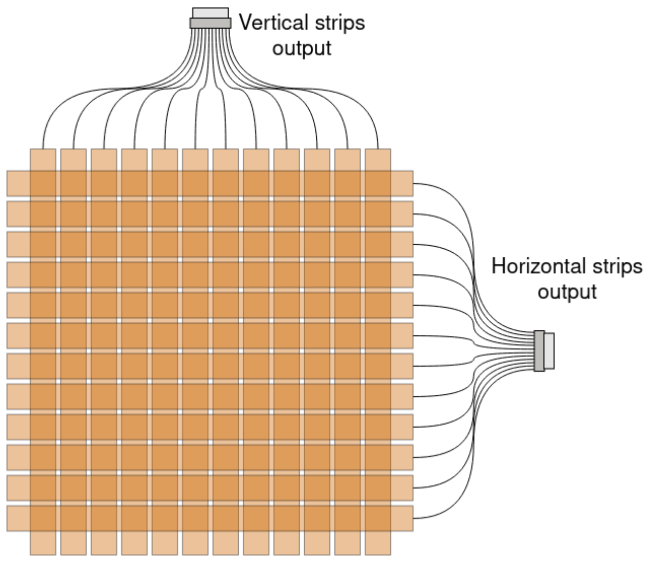

Figure 2: A simple model of a strip detector.

All strip detectors consist of layers of parallel strips. Ideally, the strip detector should output a 0 signal on all of the strips when no particles are passing through it and, once a particle passes, it should give out 1 on the strips through which it passed and the time when a particle passed.

In reality, strip detectors output an analog signal instead of a digital one. There is a potential difference being held across the strips, and the strips have no conduction electrons. Once a particle passes through a strip, it knocks out some electrons which generate an electric current. The current's magnitude is recorded and saved in *raw files*.

2 Conversion step

The first step of the BM@N experiment's data analysis is conversion. The binary raw files, which consist of blocks of analog information from various detectors, are separated into *BmnRoot* classes, which are then written into *root files*. For blocks coming from strip detectors, this information about the position is stored in *BmnADCDigit* and about time in *BmnTDCDigit* classes.

2.1 Multithreading in the Conversion step

2.1.1 Multithreading model

The information for each event (a particle from the spill hitting a target and the resultant particles triggering the detectors) can be converted independently from each other. Therefore, we can utilise `C++` standard library's *thread*, *mutex*, and *condition_variable* headers to greatly increase the performance of the conversion step on devices with multiple cores.

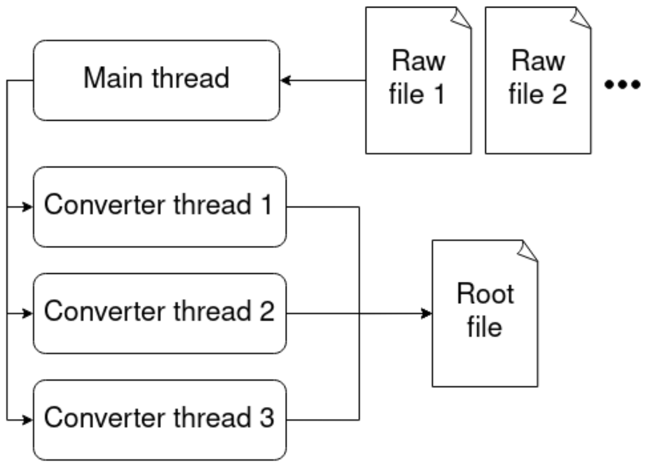

Figure 3: Multithreading approach to conversion.

Figure 3 shows one possible approach to multithreading. The main thread reads a portion of information from a raw file corresponding to one event. Then, it waits for any converter thread to become free, copies this information to this thread, signals the thread to start converting the information, and this process repeats. Converter threads wait for the main thread to copy information into them, then they convert the information and write it into a root file, after which they notify the main thread that they are free.

2.1.2 Execution time comparison

An important thing to understand is that this model does not decrease the execution time proportionate to the number of converter threads. The reading from the input raw files is still made in one thread. Moreover, writing into one file from multiple threads cannot be performed faster than the disk writing limit speed. So, a great fit for the execution time would be a hyperbola curve of a form $T_{Total} = T_{I/O} + T_{Conversion}/N_{Threads}$. T_{Total} stands for the total time of execution, $T_{I/O}$ for the time spent on reading from and writing to files, $T_{Conversion}$ for the time used for converting events, and $N_{Threads}$ for the number of converter threads.

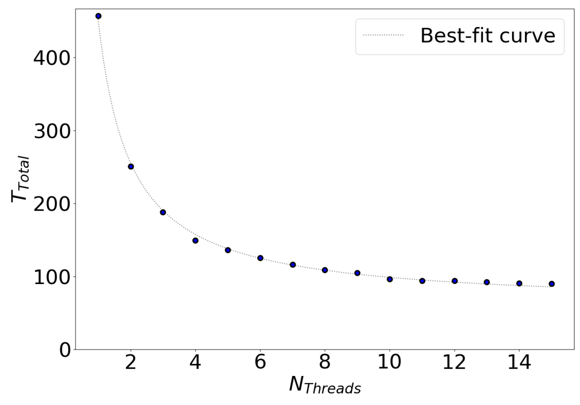

Figure 4: Best-fit curve for the multi-thread conversion implementation. The execution times were obtained by converting 50 002 events for each number of threads and averaging them over 10 runs. $T_{I/O} = 59.4\,\text{s}$, $T_{Conversion} = 393.0\,\text{s}$.

As evident from **figure 4**, this approach can significantly decrease the execution time of the conversion step. However, this approach is only effective for quickly converting a small number of raw files.

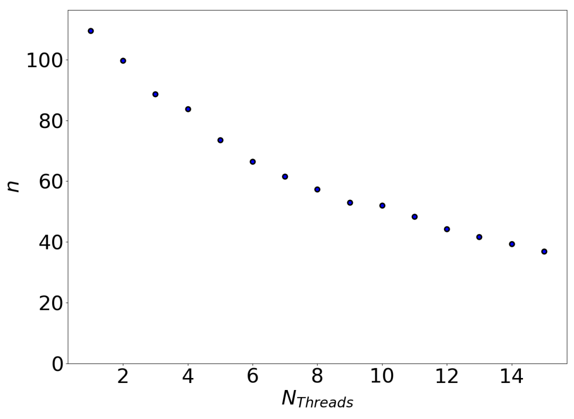

Figure 5: Number of converted events per processor per second n vs $N_{Threads}$.

The total amount of raw files produced during the BM@N experiment is in the thousands. In this case, if we want to convert all of them, we need to maximise the number of converted events per processor per second n. **Figure 5** shows that the maximum n is achieved when $N_{Threads} = 1$. Therefore, it is substantially faster to launch multiple main threads on all available processors with only one converter thread for each main thread. This conversion implementation with just one converter thread is slower than the original *BmnRoot* conversion algorithm by 0.6%.

3 Necessity of the Decoding step

The next step of data analysis in the BM@N experiment is decoding. Unlike most other steps, the events are processed as a group and not individually. The main purpose of the decoding step is to correct the output from the detectors. There are two problems with trying to proceed with data analysis without the decoding step. Firstly, strip detectors give out some current even when no particles are flying through them. And, secondly, some of the strips always output a very high signal due to some errors in electronic components.

3.1 Pedestal events

At the start of each spill, strip detectors record their electrical current on each of the strips and write it to a raw file 100 times, just as if there was an actual event. These events are called *pedestal events*. The events where there is a particle hitting a target will be called *signal events* from now on. By looking at multiple pedestal events, we can calculate the average current on each of the strips. Then we can treat the signal from a strip on signal events as 1 if it is higher than the average calculated in pedestal events by some constant, and 0 otherwise.

3.2 Noisy channels

The second problem becomes apparent once we construct a histogram of how often strips on some of the detectors output 1.

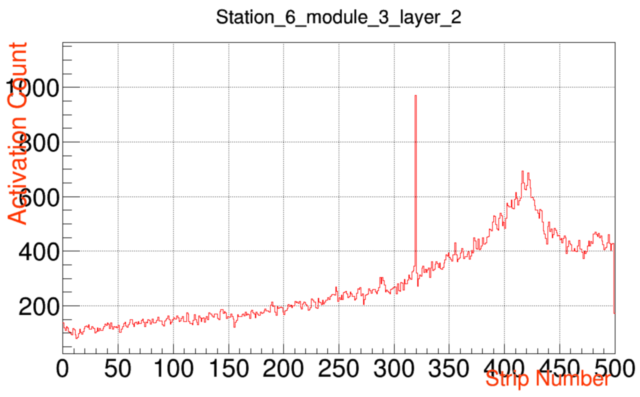

Figure 6: Number of times strips in layer 2 were activated on a GEM strip detector station 6, module 3.

From **figure 6**, it is evident that one of the strips in this strip detector outputs 1 way more often than it should. This strip is called a *noisy channel* and its output value must be set to 0 during the decoding step.

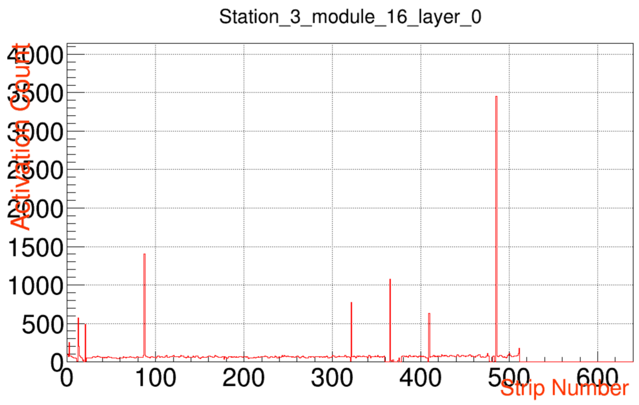

Figure 7: Number of times strips in layer 0 were activated on a silicon strip detector station 3, module 16. Most likely, the strips at the end were not working; hence, no signal from them.

Figure 7 shows that on silicon detectors the problem of noisy channels is way more severe than on GEM detectors. Currently, in *BmnRoot* noisy channels are marked if their output value is 3 times higher than the average value on a segment of 32 adjacent strips. This is not the best algorithm, as different types of strip detectors require different constants.

4 Decoding step optimisation

4.1 Raw data format

BM@N experiment consists of multiple runs. Some runs can be as short as 10 spills while other runs can last for a couple hundreds of spills. To make sure that the raw files are not too large, they were separated into parts. Each part, other than the last part, contains 25 001 events, which on average weights 14.1 GB. Moreover, the rate at which data was produced was too high to write it into one file, so the events were written into two files: even into one file and odd into another. Therefore, the raw files are named in the following way:

$$\ldots(RunID)\_\mathrm{ev}\,(Even/Odd)\_\mathrm{p}\,(Part)\,.\mathrm{data}$$

For example, a file 7444_ev0_p0.data contains 25 001 events from event number 2 to event number 50 002, and file 8001_ev1_p2.data contains

25 001 events from event number 100 005 to event number 150 005.

4.2 Different methods of data decoding

The general approach to the decoding step is clear. At first, we should look at the pedestal events and calculate the mean output currents on them. Then we go over the signal events and decide whether a strip outputs 0 or 1. Finally, we calculate the amount of times strips were activated on signal events and mark the noisy channels accordingly. The question is what pedestal events and what signal events should we take.

4.2.1 Original *BmnRoot* approach

BmnRoot decodes raw files without reordering events. What it means is that there are half of the pedestal events from 5-7 spills. The average value of current on strips during pedestal events varies with time. So, taking pedestal events from different spills and applying them to all spills at once may worsen the quality of the decoding step.

4.2.2 Decoding events in order

This method is similar to that used in *BmnRoot*. We merge ev0 and ev1 files corresponding to the same run and part together and send it to decoding. In this case, there are twice as many pedestal events and signal events. Therefore, measuring means for pedestal events and noisy channels should give a more accurate result. However, there is still that problem where we use pedestal events from all the spills on all of the spills. Moreover, some noisy channels may be only noisy during some spills, so marking them as noisy for the rest of the spills worsens the quality of further data analysis. And the opposite can happen: a noisy channel which was only noisy during one spill won't be marked as a noisy channel, which also worsens the quality of the decoded data.

4.2.3 Decoding events by spills

For this method, we need to separate ev0 and ev1 merged file from the previous method. In this method, there won't be an issue with including the pedestal events from different spills, but the amount of signal events is significantly lower, so it might not be enough to accurately find noisy channels.

4.3 Comparing pedestal events calculations

For this part, we will mostly focus on comparing the effect that taking different pedestal events has on the first 6 spill in files 7444_ev0_p0.data and 7444_ev1_p0.data.

The total number of events in these 6 spills is 45 704. The remaining events are a part of a spill, half of which is located in p1 files, so they are ignored in this comparison. The noisy channel detection is turned off, as we only want to compare the effect of taking different pedestal events for the same signal events.

All three methods were able to reconstruct the same 27 461 events and could not reconstruct the same 18 169 events, including all of the pedestal events (which is expected since there was no particle). This only leaves 74 signal events, about 0.16% of the total number of events, where the methods performed differently. Moreover, among these events, most are not even real events. The location of the *primary vertex* (a point where the particle hit the target) is located far away from the target, meaning that the reconstruction, probably, used the noises on the detectors.

We can still check whether the currents on detectors during pedestal events change significantly during one run. To do that, we can take a spill from the end of the run, reconstruct it, and then replace its pedestal events with pedestal events from the first spill in the run and reconstruct it again. In run 7444, the last spill was discharged 25 minutes later than the first spill.

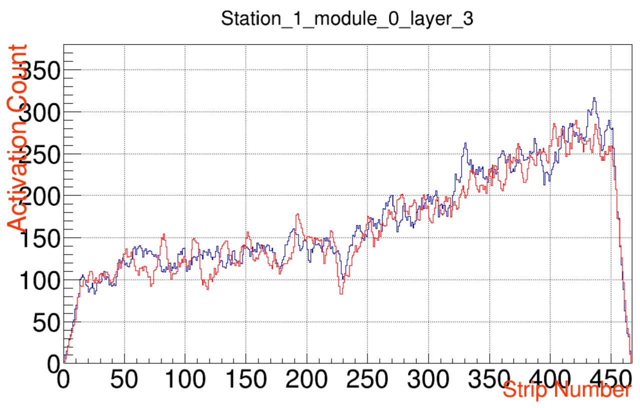

Figure 8: Number of times strips in layer 3 were activated on a CSC strip detector station 1, module 0. Blue histogram counts the activation for the first 25 001 odd events and the red one counts for the first 25 001 even events in run 7444.

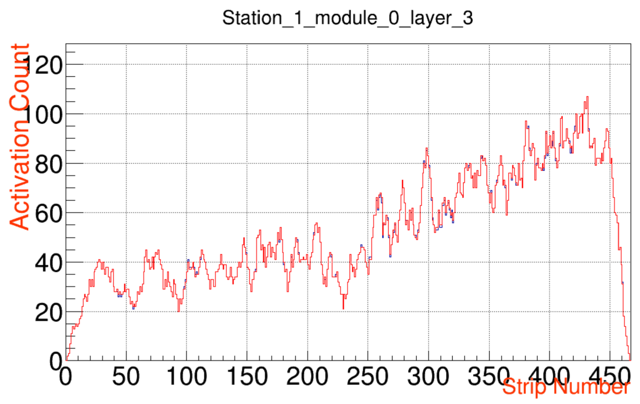

Figure 9: Number of times strips in layer 3 were activated on the same strip detector. Blue histogram counts the activation for the last spill in run 7444 with replaced pedestal events, and the red one counts for the same spill but with correct pedestal events.

Figure 8 shows that the histograms are affected by taking different signal events. **Figure 9** reveals that picking different pedestal events does not affect the strip activation count nearly as much. The blue histogram is almost entirely blocked by the red one. Further reconstructing the last spill shows that there are 4670 reconstructed events with correct pedestal events and 4671 reconstructed events with substituted pedestal events. The two methods only performed differently on 17 events, which is about 0.2% of the total number of events. And once again, most of these events result from the noises in the detectors, since their primary vertex is located far away from the target.

4.4 Optimising noisy channels calculation

The last part of this report focuses on improving the algorithm for calculating noisy channels. Current implementation in the *BmnRoot* software separates the strips into blocks of 32, calculates the average on each of these blocks and then compares the value of each strip activation count with the average on this block. If the value of this strip is higher by a somewhat arbitrary threshold constant of 3, this channel is marked as noisy.

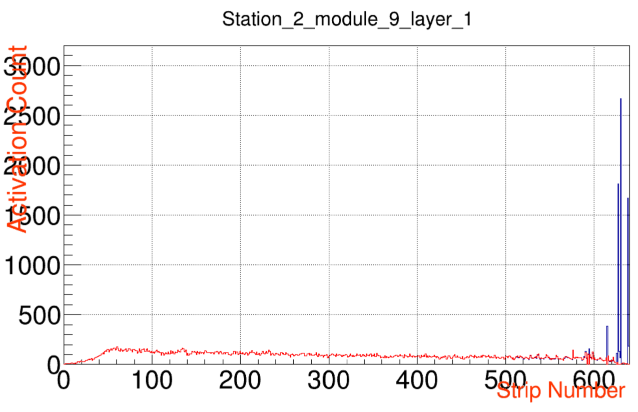

Figure 10: Bug in the *BmnRoot* algorithm My implementation is shown in red, original *BmnRoot* implementation is shown in blue.

A better implementation would be to move the window on which the average is calculated. This makes more sense, as the original implementation compared the strips on the edges of the blocks with the strips either to their left or right. I also added an extra constant that does not mark noisy channels if there is an insufficient statistic on the histogram. Moreover, as **figure 10** shows, the original implementation has a small bug where the end of the histograms is not examined for noisy channels. I also changed the values of the constants for the window size and threshold constant, until most of the noisy channels were marked. **Figure 11** shows how decreasing the sizes of blocks improves the algorithm.

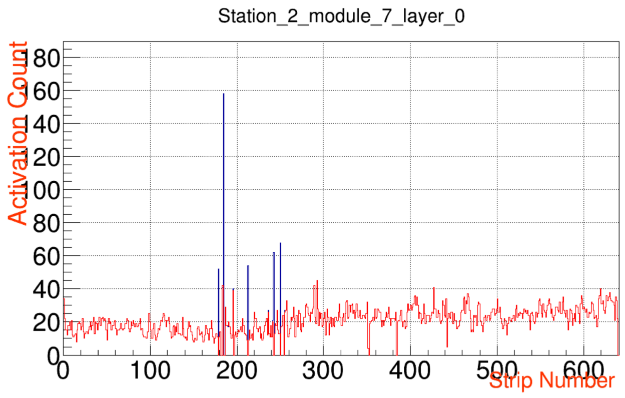

Figure 11: Comparison of the two algorithms My implementation is shown in red, original *BmnRoot* implementation is shown in blue. Previously, these noisy channels increased the average on the block of 32 and were not marked. After decreasing the window size, they became marked.

Although the algorithm for detecting noisy channels undeniably improved, reconstructing the events does not show an improvement. The standard deviation of the components of the primary vertex (which should be around the size of the target) was 8.763 for the new algorithm, 8.653 for the old algorithm, and 8.648 without marking noisy channels on a sample of 7851 events, all of which is significantly higher than the size of the target. The new algorithm reconstructed 4693 events, the old algorithm reconstructed 4699 events, and the algorithm without marking noisy channels reconstructed 4700.

The amount of reconstructed events does not identify which algorithm is better. There are events which are reconstructed from the noises and events that are not reconstructed because of the noises. So, turning off more noisy channels should decrease the number of the former and the latter, meaning there should not be a clear trend in the number of reconstructed events based on how strictly the noisy channels are filtered. However, the standard deviation of the reconstructed events should have decreased. It may be that the number of events was not enough to properly analyse the new algorithm for clearing noisy channels.

5 Conclusion

The newly-developed algorithm for the Conversion step performs significantly faster than it comes to converting one raw file, though it is not efficient when it comes to converting multiple files at once.

Tests taking different pedestal events reveal that the currents on the pedestal events remain relatively constant. Therefore, the results of the reconstruction do not depend on what pedestal events were used during the decoding step. However, the results were obtained only on the run 7444 of the BM@N experiment, so further testing is required to verify it for other runs.

A new method for clearing noisy channels was implemented. The histograms show that it manages to clear more noisy channels, though analysing the data after the reconstruction does not show improvement, possibly, due to a lack of statistics. Further testing is necessary to identify whether this algorithm is better.