Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Bachelor's Project

Interactive visualization system for hybrid active pixel detectors within the ATLAS experiment at CERN

Petr Mánek

Supervisor: Ing. Stanislav Pospíšil, DrSc.

Study Programme: Open Informatics

Field of Study: Computer and Information Science

April 2, 2016

Aknowledgements

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Declaration

Abstract

Translation of Czech abstract into English.

Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její obsah. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.
Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 About the Timepix Detectors

1.2 The Timepix Network at ATLAS

1.3 The Problem of Efficient Data Manipulation

1.4 Structure of This Document

Chapter 2

Data Structure and Storage

In this chapter, we describe the data which will be subject to visualization later on. By chronologically following the process of data acquisition, we start at the Timepix detectors, pass FPGAs, other intermediate hardware and terminate at the sensor readout. We then give details on structure of measured results and mention various permanent storage formats, their particular advantages and disadvantages. Considering all these properties, we then propose a data scheme capable of archiving such data for longer time periods, while striving to offer almost instantaneous access based on the time of measurement.

2.1 Output Produced by Timepix

Similarly to photodetectors found in common digital cameras, Timepix detectors generate measurements in the form of individual frames. A single captured frame consists of values recorded by all pixels over a given time period, length of which is referred to as *the acquisition time*. Returning to our camera analogy, this figure resembles the time of exposition of a photograph. Prolonging it, we can expect more particles to interact with our detector's pixels, making the resulting frames more saturated.

The technical principle behind the measurements is analogous to that of a Medipix sensor. Every pixel is equipped with an integer register called *the counter*. When acquisition starts, this counter is set to zero. Throughout the set time period, the counter is possibly incremented multiple times, producing a value which is read out as measurement's result for the individual pixel. This process is synchronized across all of detector's pixels, producing an integer matrix which constitutes the captured frame.

Since the pixels may not be identical due to material irregularities and manufacturing errors, every pixel has a *threshold* parameter, which is subject to calibration. If, during the measurement, the analog input measured from the pixel's semiconductor exceeds this threshold, the pixel is considered to be interacting with a particle.

2.1.1 Raw Output

Provided that every Timepix detector installed in the ATLAS network has 2 layers of 256×256 pixel matrices, every captured frame consists of 131,072 integer values in total.

The interpretation of these values depends on another parameter, *the operation mode*. While it is technically possible to configure every pixel to operate in a different mode, we have so far preferred to configure all pixels identically, making this essentially not a parameter of a pixel, but that of a frame.

The following operation modes are available:

Hit Detection Mode (also known as the One-Hit Mode) In this mode, the counter is set to one when the theshold is exceeded. Upon multiple interactions, the counter is not further incremented. The result is a Boolean value, indicating whether the pixel has interacted with a particle.

Hit Counting Mode (also known as the Medipix Mode) In this mode, the counter is incremented upon every transition from a state below the threshold to a state above the threshold. The result is an integer value representing the number of particles which have interacted with the pixel.

Time over Threshold Mode In this mode, the counter is incremented by every clock cycle spent above the threshold. The result is an integer value corresponding to the energy of the interacting particle. Further calibration to convert counter value to energy is required, though.

Time of Arrival Mode In this mode, the counter is incremented by every clock cycle after the threshold is first exceeded. The result is an integer value corresponding to the time interval before the end of the measurement.

If a captured frame contains data from pixels configured in multiple different modes, the frame is said to be measured in the **Mixed Mode** and should contain further details on the exact pixel configuration of the detector.

2.1.2 Cluster Analysis

In ATLAS measurements, we strive to configure our detectors to capture frames containing multiple disconnected components corresponding with individual interacting particles. Naively speaking, we don't want our frames to be neither fully saturated, nor empty, but *just right*. The task of achieving this level of balance is fairly straightforward, as it consists only of fine-tuning the acquisition time parameter while monitoring the levels of saturation in recently captured frames.

In well-balanced frames, we can then observe components of various shapes and sizes, depending on the experiments which were being performed in the ATLAS machine at the time of acquisition. These components, referred to as *clusters*, are discovered and evaluated in an automated process called *the cluster analysis*. This procedure involves a connectivity-checking algorithm, such as *flood-fill*, operating on the pixel matrices to distinguish individual clusters. In later stages, clusters are processed, measured and classified in various categories with regards to their shape. In addition, if the frame has been captured in TOT mode and calibration data are available, the automated processing script converts raw measured counter values to energy approximations.

The output of the cluster analysis consists of two separate lists of clusters, one per every sensor layer. It follows from the definition of a cluster that any pixel contained in it has a non-zero counter value. Consequently, all pixels unreferenced by any cluster are assumed to be equal to zero. The utilized technique of data encoding is well-known as it offers efficient compression rate for sparse pixel matrices which we are expecting to encounter in our measured data. It is however worth noting at this point that in certain cases (represented most notably by saturated or nearly saturated frames), this approach produces voluminous data structures, which may take long time to enumerate, and in turn slow down other algorithms operating on them.

In the cluster list, pixels are stored as tuples of their Carthessian coordinates and their respective counter values. From this information, the pixel matrix can be reconstructed at any time. The original pixel matrix is therefore discarded at the end of the cluster analysis, in order to minimize storage requirements. Please note that should there be any errors discovered in the future, the already processed data could be converted back into the form of pixel matrices by means of simple enumeration. Following that, the patched version of the cluster analysis process would analyze the pixel data once again, replacing any possibly erroneous output with correct one.

Let us now further inspect data generated by the process of cluster analysis. As we hinted at the beginning of this section, many other secondary values are calculated for every cluster during the automated processing, most notable of which are:

Shape Classification By measuring geometric properties of a cluster (such as radius or size), we are able to estimate whether the cluster resembles more a line segment or a circular blob. Similarly, we can also estimate if the cluster looks thin or thick. From that information, we can infer the type of interacting particle and direction of its movement relative to the plane of incidence. To formally define cluster categories, we will use terminology consistent with the ATLAS Medipix research.

Size, Volume The size of a cluster is equal to the number of connected pixels which constitute it. The volume is a sum of counter values of those pixels.

Centroid, Volumetric Centroid The centroid is defined as an unweighted average of pixel coordinates in the cluster. In analogous way, the volumetric centroid is the very same average weighted by corresponding counter values.

Minimum and Maximum Cluster Height These two figures refer to the lowest and the greatest counter values of pixels in the cluster.

Energy-based Properties ***(available only in TOT mode)*** If the energy approximations are available, many of the above-mentioned values can be also calculated with the energy substituted for counter values.

2.2 Common Storage Formats

2.2.1 The Single-Frame and Multi-Frame Formats

2.2.2 The ROOT Format

Another storage option is the ROOT Data Analysis Framework. Originally concieved at CERN in 1995, the framework provides a set of powerful tools with various applications in data mining, manipulation and visualization. Unlike other similar toolkits, ROOT comes with its own machine-independent binary file format (identified by the `.root` extension). This format is designed to store enormous amounts of data within various types of data structures efficiently, while maintaining good overall performance by employing low-level memory optimization techniques and multi-tier content caching.

Used by many physicists at CERN for several years now, ROOT seems like a good choice of a data archivation format as many researchers have already learned its caveats and know well how to operate it despite often lacking deeper background in Computer Science. For the purposes of programmatic access, ROOT also does well with documented APIs in Python, R and C++.

Should the processed data be stored in ROOT, a basic relational database concept comes to mind. With standard tables generalized in the form of *trees* and their columns in the form of *leaves*. One tree would suffice for the information about captured frames (such as acquisition time, operation mode, etc.) and other for the list of clusters for every frame. Such trees would efficiently abstract the entire storage structure, allowing for multiple frames to be stored in a single file, grouped for instance by a common time interval.

In spite of being over 20 years in development, ROOT is not perfect. Using memory monitoring tools such as *valgrind*, we have confirmed that the C++ implementation of the ROOT framework is riddled with various memory leaks, making it unsuitable for time-extensive operations. Some might also argue, that a full tree data structure might be overly-complicated and too general for a simple output described in previous sections. Lastly, ROOT framework has quite a complex object structure, making it hard to learn for first-time users.

2.3 Proposed Data Structure

2.3.1 Formal Requirements

Having defined the essence of information we wish to store and several data formats as means to do it, we are now ready to focus on the definition of our database. Requirements on such a data structure are as common as database requirements can get. It should be a reliable permanent storage element, accessible for reading from multiple workstations at a same time and robust enough to withstand minor hardware failures. With 15 detectors already installed at ATLAS, and possible option of installing another 5, the database should be designed to hold frames from up to 20 Timepix devices for the entire expected time period of their operation at LHC (that from June 2015 to LS3 in 2021).

As more and more frames arrive from the detector network, our database should allow to be periodically extended with new data, possibly processing and converting pixel matrices into cluster lists, as described in the previous sections. Since the database will be primary storage site for all research data, there should be multiple independent copies of it as backups and the database structure should be designed with logic to enable timely synchronization of these copies.

Apart from all the requirements already listed, we have the advantage of knowing how the majority of user queries will look like. With regards to this information, we may then optimize data storage and retrieval procedures to accelerate such queries. After discussing all use cases with the researchers who are going to operate the database, we have determined that most queries will filter data by time or by device. This is indeed a very natural method, provided that every device in the network is positioned and oriented in way allowing only for a certain type of particles to be observed. Researchers looking for signs of specific particles might often request data based on other experiments, which were conducted in a determinate time period and involved only a specific group of detectors in the network.

2.3.2 Definition

With all requirements in mind, we will now formally define the database. Accounting for the ever-growing nature of our data, we will separate the database into two parts. The first part is to contain data which has already been processed by the cluster analysis, and is ready to be accessed by users. The second part will contain data which has arrived from CERN in its raw form but hasn't been processed yet. As one might note, this separation of data serves a fundamental purpose, that is to distinguish the intermediate products from the finished ones.

For simplicity, the database will be represented by a UNIX file system. This will enable many users, not necessarily only those using UNIX-based operating systems, to access it directly by means of widely-used and standardized protocols, such as FTP, SMB, SSH, AFP or HTTP. Utilization of these protocols contributes not only to the universality of our database, it also takes care of shared resource access and other data concurrency issues for us. Some of these features may prove to be useful later on when synchronizing various storage sites in order to back up or restore data. UNIX file systems also offer fundamental security features, allowing us to grant read-write privileges to a certain group of users, while limiting others to mere read-only access.

The file system will have two directories named `processed` and `downloading`, corresponding to the respective sections of the database. In these directories, data will be further grouped in subdirectories by the device of origin. To make navigation easier, device directory names will use numeric identifiers in compliance with already published literature. For instance, all data originating from the sensor no. 7 will be stored in a directory named `ATPX07`. In such directory, data will be stored in time-coded files (or directories, should multiple files be grouped under single time code) according to the naming pattern: `[yyyy]_[mm]_[dd]_ATPX[id]` (where `[id]` is substituted for the device identifier and `[yyyy]`, `[mm]`, `[dd]` are substituted for year, month and day of the acquisition time respectively).

If it is not possible to group data by the day of acquisition for some reason, we define an alternative naming pattern with hourly granularity: `[yyyy]_[mm]_[dd]_ATPX[id]_[hh]` (where `[hh]` is substituted for the hour of acquisition and other entities are substituted in the same way as in the previous pattern). Note that in spite of grouping data files in separate subdirectories by the device of origin, we still include the device identifier in the naming pattern for reasons of redundancy.

The directory structure we have described so far satisfies all requirements we have stated in the previous section. What's more, it optimizes access to data generated from specific devices at specific times, so that the majority of user requests is satisfied in timely manner.

Let us now define the data files themselves. All data files will be stored at the lowest level of our directory structure and will have time-coded names according to our naming patterns. Should more files fall under the same time code (marginal scenario), they are to be grouped in a directory with a time-coded name. File structure in such a directory is undefined. We expect all files in the `processed` directory to be stored in the ROOT format, and all files in the `downloading` directory to be stored as multi-frames. All other files of different types will be tolerated as they may contain relevant information, but regarded as secondary.

To preserve storage space, we will allow usage of data compression in our database. The supported compression formats are ZIP, GZIP and TAR, or any combination of them. As we expect the individual data files to grow quite large in size, we will utilize compression only at the lowest level of our directory structure, that is in the time-coded data files (or directories). Every archive can store at most one time-coded file (or directory), hence the archive can adopt the file's time-coded name, while remaining unique in the file listing. It is preferred, but not required, that all data files stored in a single directory are either all compressed or none of them is, as any deviation from this scheme might point to an incomplete or broken data transaction. Lastly, we forbid any recursively compressed structures (such as archive within other archive, etc.). The recommended alternative is to increase compression level in already existing archives instead of creating new ones. This rule also applies for all data formats which use compression inherently, such as ROOT.

2.3.3 Expected Volume of Data

With our definition in mind, we will now perform a simple extrapolation to obtain an upper bound on the size of our database.

Assuming that one hour of footage stored in the multi-frame format may take up to 4 gigabytes in size (depending on the frequency of acquisition), we have 96 gigabytes per sensor per day. Accounting for the longest possible time of operation, our database will store up to 2,437 days of footage simultaneously recorded by up to 20 detectors. That means that our database will have to hold about 4.7 petabytes worth of uncompressed information. If we use Collin's compression algorithm benchmark from [1] as baseline, it is possible estimate that a common variant of GZIP algorithm will reduce the file size in average by 75.9%. Applying this compression on our multi-frame data files, our database would have to hold *only* about 1.1 petabyte of archives.

We will now perform analogous calculation for the ROOT file format. Since the file structure already utilizes its own proprietary compression algorithms, we expect the overall volume to decrease significantly in comparison with the raw uncompressed multi-frame data. From the data recorded by the ATLAS Timepix network in the fall of 2015, we observe that a single day of footage stored in the ROOT format may take up to 18 gigabytes in size. Using the same constants as before, we arrive at the conclusion that our database will have to hold about 877 terabytes of information. This result is in agreement with our expectations.

Please note that neither of these upper bounds is by any means, since we intentionally over-estimated the number of detectors in our network and the length of the operation period in our assumptions. In addition, it is likely that some of our detectors will be configured to capture data with frequencies lower than the maximum possible frequency as every device is configured separately to observe particles at different speeds. For all these reasons, our estimation only gives us vague information about the orders of magnitude of storage space required to operate our database and its subsequent backups. In spite of this limitation, the estimation suffices to design and rate other components of our system.

2.4 Performance Optimizations

Chapter 3

Communication Protocol

3.1 Requirements

3.2 Underlying Standards

3.3 Web Methods

Chapter 4

Data Server

4.1 Role of the Application

4.2 Decomposition

4.3 Dependencies

4.4 Object-Oriented Design

4.5 A Note on Parallelism

4.6 Performance Optimizations

Chapter 5

Web Visualization

5.1 Naive Decomposition

5.2 Final Decomposition

5.3 Underlying Standards

5.4 Dependencies

5.5 Website Structure

Chapter 6

Conclusion

6.1 System Deployment

6.2 Data Import

6.3 Automating Data Acquisition

6.4 Future of the Application

Bibliography

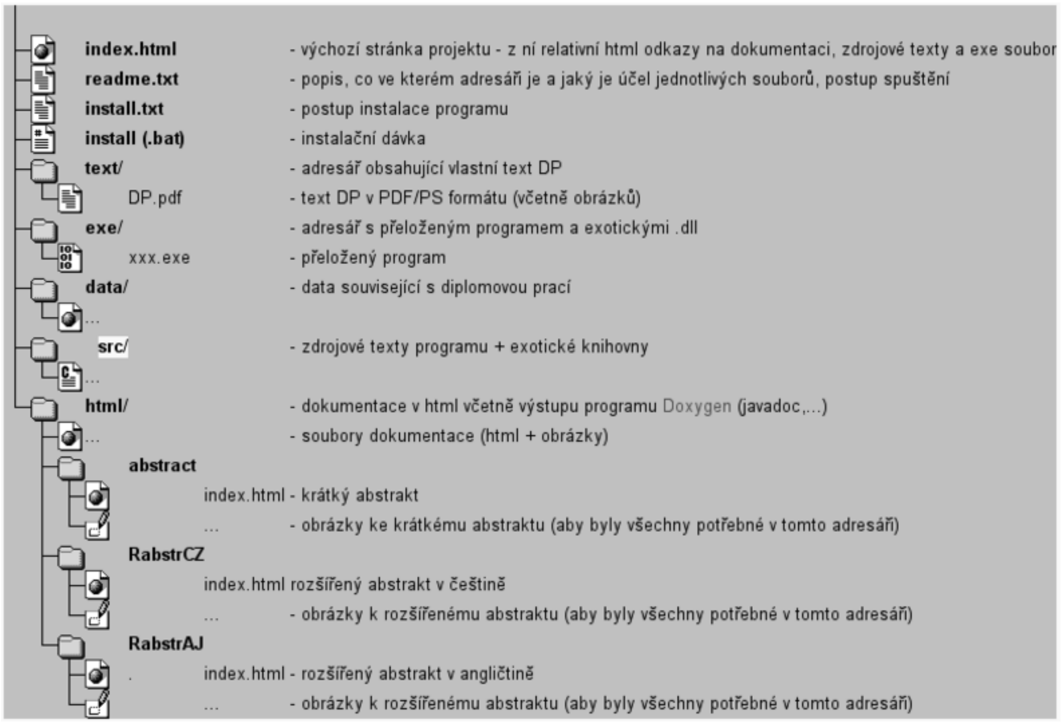
[1] COLLIN, L. *A Quick Benchmark: Gzip vs. Bzip2 vs. LZMA* [online]. 2005. Dostupné z: <https://web.archive.org/web/20150907021223/http://tukaani.org/lzma/benchmarks.html>.

Appendix A

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [?]):



```
index.html           - výchozí stránka projektu - z ní relativní html odkazy na dokumentaci, zdrojové texty a exe soubor
readme.txt           - popis, co ve kterém adresáři je a jaký je účel jednotlivých souborů, postup spuštění
install.txt          - postup instalace programu
install (.bat)       - instalační dávka
text/                - adresář obsahující vlastní text DP
    DP.pdf           - text DP v PDF/PS formátu (včetně obrázků)
exe/                 - adresář s přeloženým programem a exotickými .dll
    xxx.exe          - přeložený program
data/                - data související s diplomovou prací
  ...
  src/               - zdrojové texty programu + exotické knihovny
  ...
html/                - dokumentace v html včetně výstupu programu Doxygen (javadoc,...)
  ...                - soubory dokumentace (html + obrázky)
    abstract
          index.html - krátký abstrakt
          ...        - obrázky ke krátkému abstraktu (aby byly všechny potřebné v tomto adresáři)
    RabstrCZ
          index.html rozšířený abstrakt v češtině
          ...        - obrázky k rozšířenému abstraktu (aby byly všechny potřebné v tomto adresáři)
    RabstrAJ
    .     index.html - rozšířený abstrakt v angličtině
          ...        - obrázky k rozšířenému abstraktu (aby byly všechny potřebné v tomto adresáři)
```

Figure A.1: Seznam přiloženého CD — příklad