Riccardo Maria BIANCHI

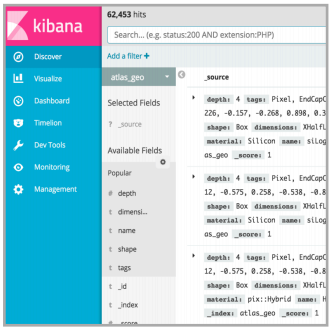
DEVELOPER PORTFOLIO

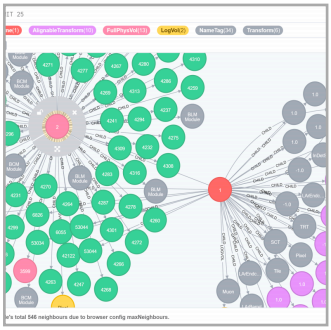
Scientific Software Development
selected projects

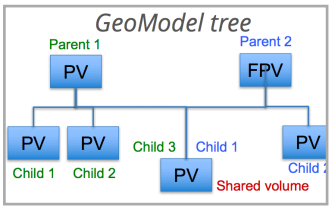
August 2017

Distributed RESTful services to serve Geometry data

Interactive querying and retrieval of detector description data







2017

C++, Python, SQL, Cypher

Neo4j, ElasticSearch, SQLite

Until now, **Detector Description** data were accessible only wrappers from within the ATLAS framework; and only from memory.

That prevented to use the detector description data in standalone applications; also, was not possible to query geometry data, neither retrieve data subsets.

We built two web services, with two different technologies, with two different targets in mind. A graph query and REST API for interactively querying the actual ATLAS Detector Description data, based on **Neo4j**; where users can explore and debug the ATLAS geometry used in all ATLAS tasks. And a REST service, based on an **ElasticSearch** cluster, for interactively fast retrieve of the final volumes composing the ATLAS geometry: all transformations are computed and all attributes are collected at indexing time.

Facts & Figures

The new web services have been presented at the *ACAT 2017* conference.

https://indico.cern.ch/event/567550/contributions/2628864/

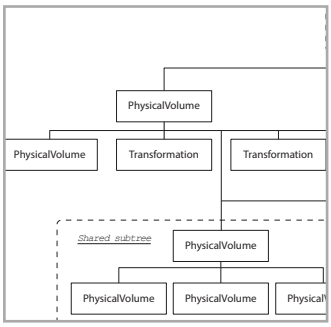
GeoModel persistification

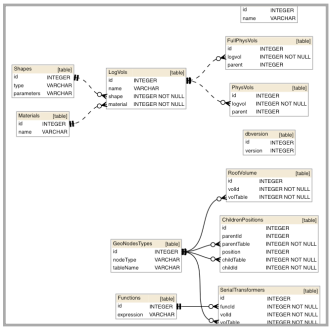
An experiment-agnostic detector description on file

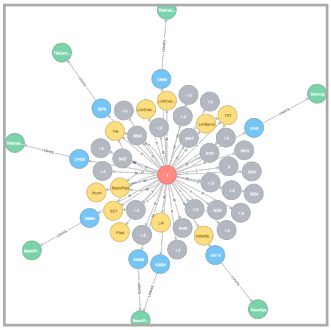
2016-2017

C++, Python, SQL, Cypher

Qt5, SQLAlchemy, SQLite, Neo4j







GeoModel is a library used for describing the **detector geometry** of the ATLAS experiment. The configuration is stored in an online database and the geometry tree is built in-memory, to be used in the reconstruction and in the simulation (through converters for *Geant4*). GeoModel always lacked a persistification mechanism.

As a start, I have extracted GeoModel from the ATLAS software: it is **now a standalone package**, ready to be used in other HEP experiments. Then, I created **new packages** to **dump** the geometry and to **restore** it; and I designed a data model and a **database** (SQLite).

A great effort has been put in optimizing the **data model**, to have **fast access** to data and a **small size** of the database file.

A new development is focused on serving the geometry through a **REST** service from ElasticSearch and from a Neo4j **graph DB**.

Facts & Figures

The memory footprint of the ATLAS detector description is about 150 Mb, while the size of the new database file storing it is about **45 Mb**. Small enough to be sent in an email.

The new GeoModel has been presented at the *CHEP 2016* conference. The outcome of the new development will be submitted to *ACAT 2017*.

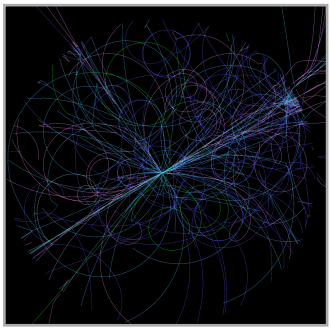
VP1 Light

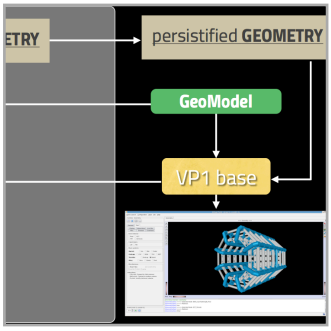
A standalone experiment-agnostic event display

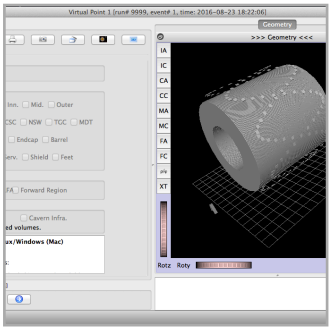
2016-2017

C++

Coin3D (OpenInventor), Qt5







VP1 Light is a standalone event display aimed at being used in physics analysis.

VP1, the ATLAS 3D event display, is a powerful tool, integrated in the experiment's framework and capable to access and visualize all kinds of ATLAS data. But it must be run within the experiment's software release, so it is heavy and not user-friendly; in particular for users who want to use it for physics analysis.

I have extracted all VP1 base packages from the experiment framework; then I integrated the new GeoModel persistification mechanism. VP1 Light is now **independent from ATLAS**, but capable to show the full ATLAS geometry as VP1 does.

The next step will be the development of an **interface to ROOT files**. In this way, VP1 Light would be able to show physics objects as well, like tracks and jets.

VP1 Light is developed in *C++*, making use of the **3D graphics** library *Coin3D*.

Facts & Figures

The initialization of VP1 takes a very long time, because the whole ATLAS framework must be started, even when not needed. On the contrary, VP1 Light now shows the ATLAS geometry, in all its glory, in only **few seconds**.

Presented at the *CHEP 2016* conference.

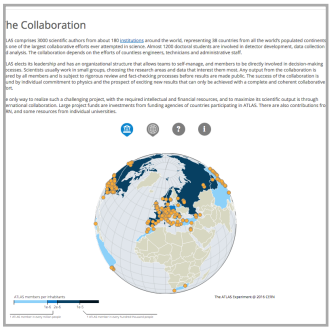
ATLAS Collaboration Map

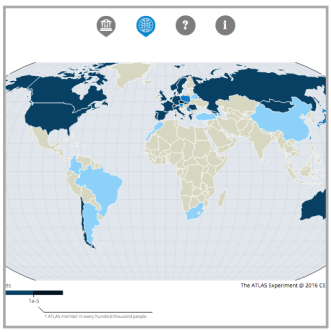
An web interactive map of the ATLAS Collaboration

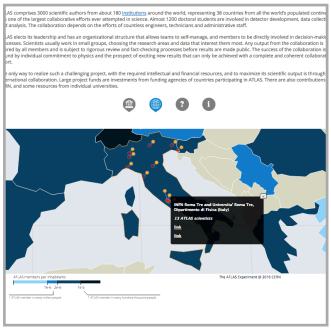
2015-2016

Python, Javascript

D3.js, JSON







The **ATLAS Collaboration** is composed by around 140 institutions from around 40 countries. And it is always evolving: new members and institutes join the collaboration, others leave.

So far, the only map of the collaboration was on a poster, drawn by hand. The ATLAS management wanted a more modern map.

Thus, I proposed to build a **web interactive map**, from data fetched directly from the ATLAS database, *Glance*. I developed *Python* tools to get, clean, filter and organize data from the database. Then I developed an interactive map to let the user explore the member institutes and their countries, and to **visualize different information** from the database.

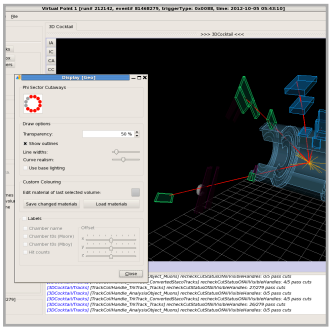
The *Python* tools export data in the *JSON* format, which are then read from the **web application** written in *Javascript*. The data visualization and the map are developed using the library *D3.js*. Cron jobs **automatically update** the map data by regularly fetching new data from the Collaboration database.

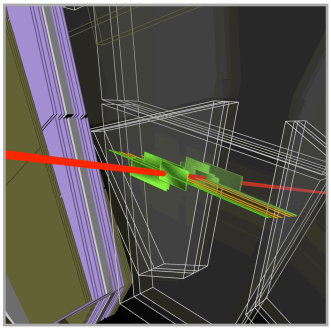
VP1
The ATLAS 3D event display

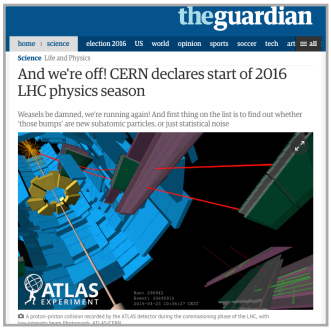
2012-2017

C++, Python

Coin3D, Qt5







VP1 is the ATLAS **3D event display** and it is part of the experiment's **Core Software**. Being integrated in the experiment's framework, VP1 can access and **visualize all kind of experimental data**. It also visualize the actual detector **geometry**, whose configuration is taken directly from the online ATLAS Geometry database.

VP1 let users interact with the data and the visualization, applying cuts and filters, editing the visualization settings and setting views.

VP1 is a **framework** of about **40 packages**. Base packages, interfaces and plugins are written in *C++*. For the 3D graphics, it makes use of the *Coin3D* (*Open Inventor*) graphics library. Users can add additional views and functionalities through **plugins**.

I am the **lead developer** and the **maintainer** of the whole VP1 framework.

Facts & Figures

VP1 is used by experts at all phases of the ATLAS data chain, for a number of different tasks: detector development, verifying the reconstruction and the simulation, data analysis.

It is also used to produces images for Outreach&Education, press releases and multimedia.

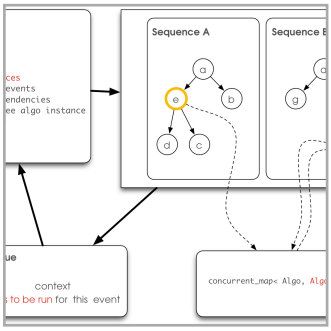
Whiteboard

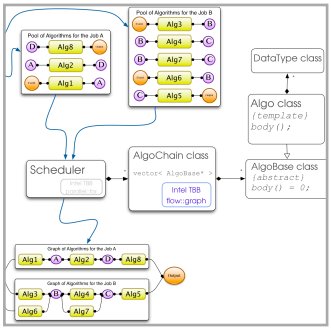
A prototype for a new multi-threaded HEP framework

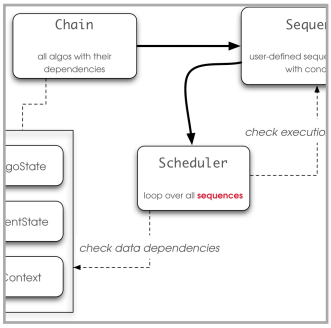
2012

C++

Intel TBB: tbb::flow_graph, tbb::task







HEP experiments rely on very complex software frameworks to handle and to analyze experimental data. In the current multi- and many-cores era, we need to provide the current **HEP frameworks** with multi-threading and parallelization features, in order to better or fully exploit the speed and the throughput of **modern hardware**.

Many **modern multi-threading technologies** are currently on the market, and one of the most interesting ones is the Intel *Threading Building Blocks* (**TBB**).

To test TBB performances in the HEP context the prototype implements a basic **parallel HEP framework**: a transient data storage, an event manager, a scheduling mechanism, an Algorithm class and its derived classes.

WHITEBOARD has been developed in **C++**, using the *tbb::graph* and *tbb::task* libraries to **schedule and run algorithms and events in parallel**.

Facts & Figures

WHITEBOARD inspects the possibility of using a **task-based design in HEP**.

It has been presented at the **inter-experiments** *Forum on Concurrent Programming Models and Frameworks* at CERN:

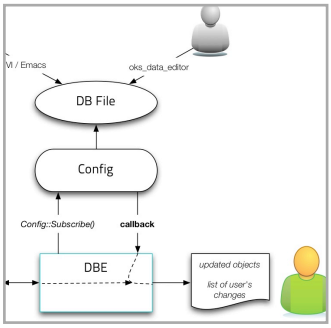
http://cern.ch/go/zB98

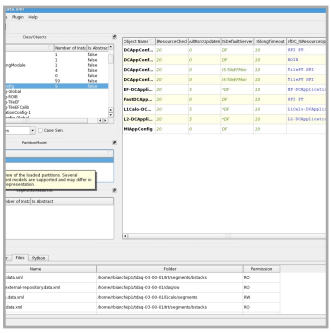
DBE
Multi-threaded multi-user Online DataBase Editor

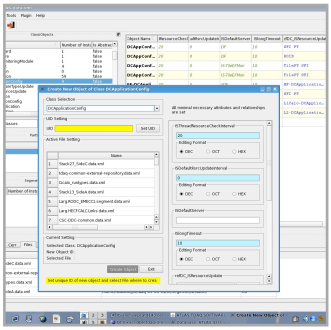
2010-2012

C++, Python

Qt Model-View-Controller GUI, Boost.Python, Qt threads







DBE is a **multi-threaded multi-user editor** which lets the users to create, edit and handle configuration files **for the online system of the ATLAS experiment**.

The development of the DBE is part of the ATLAS TDAQ software re-engineering. It has been designed and developed to replace the previous editor, which was based on the old *Motif* graphical libraries and which lacked most of the features nowadays expected in any modern editor, like **multi-threading, drag&drop, the UNDO/REDO mechanism, a plug-in interface**.

DBE has been developed in **C++**, using the **Qt graphical libraries**. The architecture is based on the Model-View-Controller pattern.

A **Python embedded interface** has been implemented to let the user extend the editor functionalities by mean of Python scripts (SWIG, REFLEX).

Facts & Figures

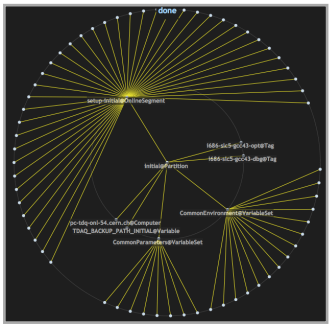
DBE is fully integrated in the ATLAS TDAQ software release. It is also deployed in the experiment control room as the default ATLAS Configuration editor.

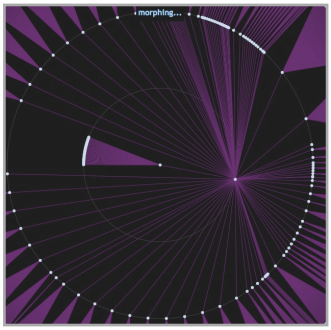
DBV
DataBase Viewer

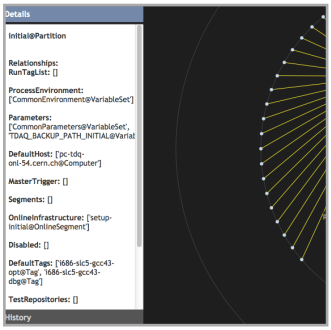
2011-2012

Python, Javascript

CherryPy, Mootools, InfoVis







DBV is a modern **web-based interactive database viewer**, designed to let the users easily look at the relationships among the objects defined in the **ATLAS online system configuration database**.

At present the ATLAS Online Configuration database stores *O(100k)* objects and many of them have relationships towards several hundreds of objects.

A great effort is being put in **data visualization** to find the best way to present this large dataset to the user, with clarity and simplicity, assuring a **nice user experience and clever interaction**.

A special attention is being used to find the best **data format** as well, in order to handle this large amount of data, while assuring a **fast query and a prompt answer** to user commands.

DBV is implemented as a **web application**. The **back-end** has been written in **Python** using the *CherryPy* framework; this part handles the requests from the users, returning the results of the queries to the graphical interface. The graphical **front-end** is written in **Javascript**, using the *Mootools* and *InfoVis Toolkit* libraries.

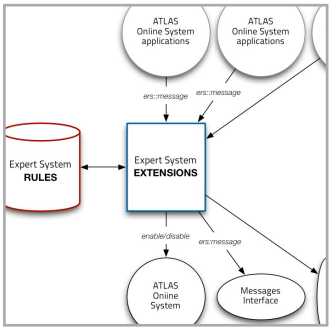
Expert System Tester

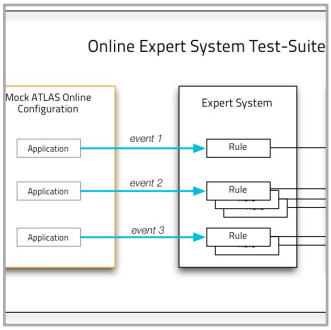
Automated multi-threaded test suite for the ATLAS online Expert System

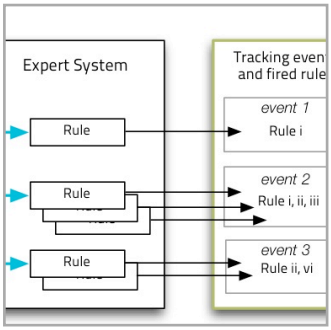
2012

C++, Python, CLIPS

Intel TBB, OpenMP, Python threading







The ATLAS Online Expert System takes care of the control of the correct setup of the ATLAS detector during the data-taking.

The Expert System is implemented in **CLIPS**, it's part of the TDAQ Software Release and its rules are fired by messages passed through a CORBA protocol by the many applications running the sub-detectors. Several actions are then taken by the system, following the fired rules, to assure the smooth run of the detector.

Being now the number of implemented rules very high (~2000), a need for an **automated test suite** has been arisen from the TDAQ community. The new test suite would emulate the ATLAS running configuration, test the active rules **generating fake messages from dummy applications** and test new rules before applying them in production, keeping track of the occurred conditions.

Facts & Figures

The new test suite is now in the design phase, where demonstrators are being developed to test the available technologies in term of complexity, maintainability and speed.

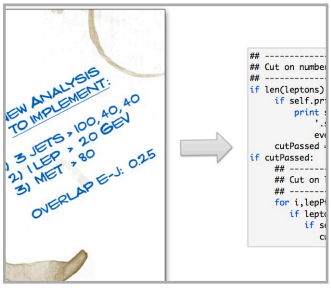
Several **multi-threading** libraries are being tested to assure a smooth test of many rules in parallel on multi-core machines.

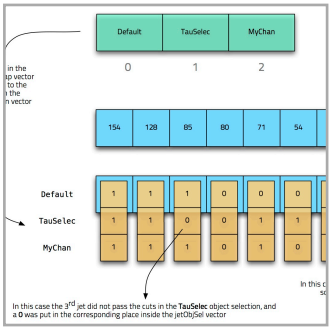
WatchMan
Analysis multi-threaded CASE framework

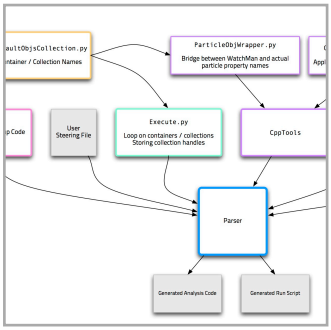
2008-2010 *PhD Thesis*

C++, Python

ROOT, *PyROOT*, *Athena*, *Python threading*







WatchMan is a multi-threaded **CASE** (Computer Aided Software Engineering) framework, to **easily implement and handle a large number of Physics data analyses**.

The idea came from the consideration that most of the analysis classes have the major part of their code in common. Accessing data, looping over them, saving quantities and building histograms, for example, are all common operations whose code is usually copied and pasted from a class to another.

WatchMan takes the user settings through a simple text-based interface, and **generate the final analysis code** ready to be run on data.

Interfaces for different data formats can be added by users.

Facts & Figures

WatchMan has been used very successfully by the Freiburg group and partially by the ATLAS SUSY group to analyze the data from the first LHC collisions in 2010, and to make *ntuples* from Athena COOL data files on Grid.

WatchMan is now an independent package, presented at ACAT 2010 and EuroSciPy 2010 **conferences**. A **paper** has been published in the peer-reviewed *Journal of Computational Science* (Elsevier):

http://dx.doi.org/10.1016/j.jocs.2012.04.005

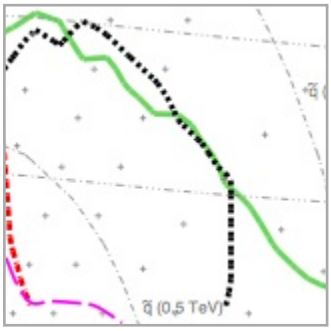
SUSYTools

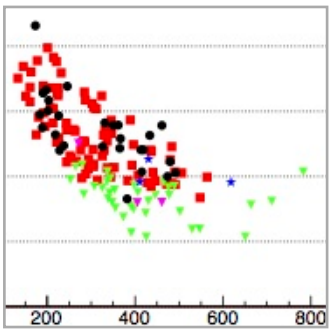
SUSY Analysis Toolkit

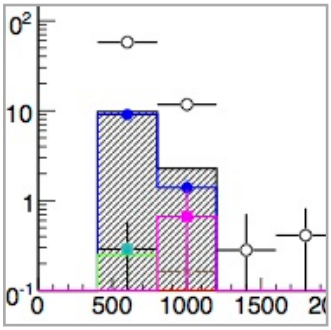
2007-2009 *PhD Thesis*

C++, Python

ROOT, *PyROOT*, *Athena*







SUSYTools is a collection of common algorithms, functions and tools used in **data analysis**, targeted to the search of Supersymmetric particles.

SUSYTools has been developed in C++ as an Athena package, and loadable as external library in users' analysis code.

Python bindings are generated with **Reflex**.

Facts & Figures

SUSYTools was initially used for Athena-based data analysis. Then it was extended to be used on ATLAS D3PD data files. It is now maintained by the ATLAS SUSY Physics Group.

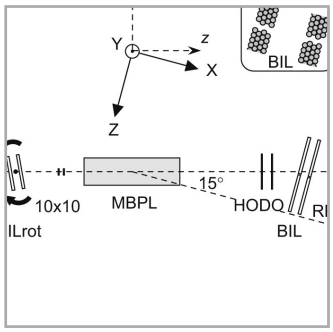
MDT Calibration

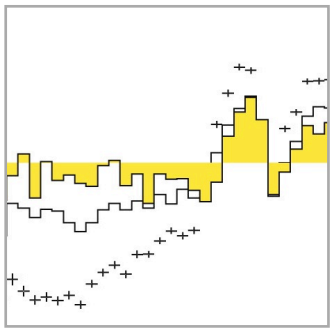
Optimizing the ATLAS MDT calibration

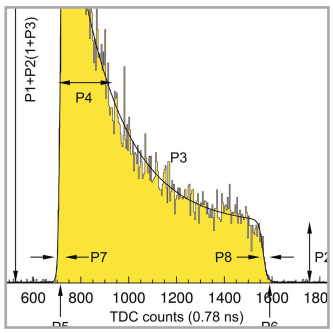
2006 *Master Thesis*

Fortran, C++

PAW, *ROOT*







MDT muon chambers need accurate **calibration constants** in order to extract position and timing information from the detector signal.

At the time of my thesis, new test-beam data were taken with test beams, and I was involved in the the **optimization** of the computing algorithms for the simulation, reconstruction and calibration (*Fortran*, *C++*), in the development of MC simulation tools (*Fortran*), and in the data **analysis** (*ROOT*, *PAW*).

Facts & Figures

The work ended with **new** and better understandood calibration constants, together with **faster and optimized algorithms**.

The results of my studies and the new algorithms were then **adopted as new standards by the ATLAS MDT** community and published as a part of a peer-reviewed journal paper on *Nucl. Instr. Meth. A* (Elsevier):

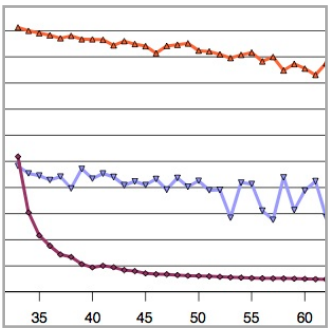
http://dx.doi.org/10.1016/j.nima.2008.09.031

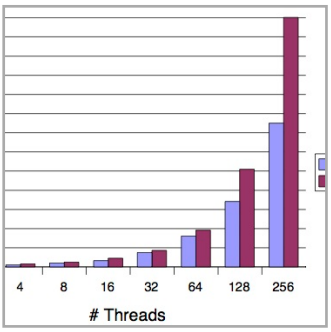
https://cdsweb.cern.ch/record/968565

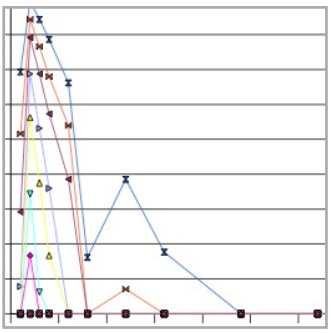
POSIX Threads evaluation

Multi-threading for ATLAS DAQ ROS applications

2003 *Bachelor Thesis*







C++, C

Posix Threads (NPTL, NGPT), Linux Kernel Optimization

NPTL and NGPT were developed around 2003 (by RedHat and IBM, respectively) to solve the problems found in the original Linux implementation of the **POSIX threads standard**, *LinuxThreads*. NPTL is based on a *1x1* model, to give a kernel task to each thread; while NGPT adopted a *NxM* model.

As the **ATLAS ReadOut System** (ROS) makes use of the Linux threading extensively, an evaluation of the two new threading implementations was needed to understand their influences in the ROS performances.

Various **Linux kernels** with the two libraries and two versions of the **Linux scheduler** were **built and optimized**, and their performances have been tested and quantified in terms of threads scaling, bottlenecks and speed.

Facts & Figures

NPTL gave better results compared with LinuxThreads. And the modifications to the Linux scheduler solved several bottlenecks which had been observed before. The ATLAS ROS could benefit from this new features.

The NGPT project was abandoned by IBM in late 2003, and the NPTL was adopted as the new POSIX Threads implementation for the Linux kernel, starting from glibc 2.4 and kernel 2.6.

Programming skills:

Programming languages: C++, Python, Javascript, Fortran, Processing • **Multithreading**: Boost, Intel TBB, OpenMP, pthread, QtThread, pythreading, MPI • **GUI libraries**: Qt4/Qt5 • **Javascript libraries**: D3.js, jQuery, Bootstrap, Mootools, Qooxdoo, InfoVis, Flot • **Markup**: XML, JSON, XHTML+CSS • **Query languages**: SQL, Cypher • **Software Optimization** • **Builders:** CMake, QMake, Makefile, CMT • **CI**: BuildBot, Jenkins • **Dev Tools**: gdb, gproof, Intel Parallel Studio, Valgrind, CppUnit, Boost.Test, Google Test, Catch • **IDE**: Eclipse, XCode • **Data Analysis**: Pandas, Matplotlib, ROOT, PyROOT • **Physics frameworks**: Athena • **Databases**: SQLite, Neo4j graph DB • **Version control**: Git, SVN, CVS • **Code repositories:** GitLab, GitHub, Bitbucket • **Issue-tracking & User support**: JIRA, Trac, Savannah, Bugzilla, Sharepoint • **Typesetting**: LaTeX, XeLaTeX • **Software development cycle** • Agile development • **Project management** and development organization with JIRA and other platforms

Developer Portfolio *(this document)*:

https://ric-bianchi.github.io/cv/BianchiPortfolio.pdf

Contact:

Dr. Riccardo Maria BIANCHI
University of Pittsburgh

c/o CERN
CH-1211 Genève 23
Switzerland

email: rbianchi@cern.ch
www: https://www.riccardomariabianchi.com

Last update: 25.08.2017