ANA - A framework for building ATHENA on ARM

Joshua Wyatt Smith, Andrew Hamilton

Department of Physics
The University of Cape Town, Rondebosch, South Africa
E-mail: joshua.wyatt.smith@cern.ch

Abstract. The ATLAS software framework (ATHENA) is large and dynamic, comprised of around 6.5 million lines of code. It is compiled using the ATLAS monitoring system, NICOS which uses tools and scripts located and tuned for the CERN services, LXPLUS and AFS. Furthermore, the constraints placed on the hardware that the software is based, limits compilations to traditional x86 architecture. With the sudden interest in ARM processors for large scale high energy physics computing, a new system needs to be implemented to build ATHENA versions for ARM, on ARM. This letter serves to introduce a building framework called Atlas Nightly on ARM (ANA). This new framework implements patches to suit the ARM architecture with the goal of a final ATHENA version for ARM.

1. Introduction

The ATLAS [1] experiment at the Large Hadron Collider (LHC) [2] is one of the experiments responsible for probing Standard Model and Beyond Standard Model physics, using the proton-proton collisions of the LHC. In 2011 and 2012 the center of mass energies reached 7 TeV and 8 TeV, respectively. The LHC has been shutdown for the last year due to upgrades but is expected to restart in March 2015 with a center of mass energy of 13 TeV. The detector’s hardware and software have to keep up with this increased flux of data. Furthermore, once these data have been collected, it must be analyzed at server farms around the world. Complex and powerful software has been created for the ATLAS experiment in the form of a “software stack” and is deployed on traditional x86 computers. The type of analysis that ATLAS (and all experiments at the LHC) do is that of high throughput computing. This means that there is no upper bound on the number of CPUs that can be used when analyzing the data. The cost of running these server farms from the perspective of power usage now becomes an important factor.

ARM [3] processors have recently seen a rise in popularity within the scientific community. Their respectable processing speed and very conservative power usage makes ARM an interesting choice in high energy physics computing. With the introduction of the new 64-bit ARM server it is clear that ARM processors are still
going through improvements. It is therefore desirable to have a system in place so that
the ATLAS software can be repeatedly and reliably compiled for the ARM architecture.
The work presented here was carried out on the Nvidia Jetson TK1 development board
as a proof-of-concept. The board has a quad-core cortex-A15 processor with a maximum
power draw of approximately 12 Watts, running Fedora 21.

2. The ATLAS software

The software stack used by ATLAS, Athena (named after the Greek goddess of wisdom)
is complex and dynamic. It thus is more practical to separate the different packages
into individual projects and then build from ground up. Fundamental projects known
as LHC Computing Grid (LCG) software, Gaudi [4] and ATLAS specific code comprise
an Athena release. A release consists of around 6.5 million lines of code. Roughly
4.5 million lines are written in C++, 1.5 million in python, while other languages
make up the rest. Athena versions are compiled using the NIghtly COntrl System
(NICOS) which facilitates the building of the individual packages and projects for such
a large framework. Complete versions are compiled every night for the x86_64-slc6-gcc48
platform (i686-slc6-gcc47 and x86_64-slc6-gcc47 are also compiled for older versions of
Athena) with standard tests being carried out simultaneously. Fixes are performed the
next day once bug reports have been generated. Releases are compiled using CMT [5] (a
package managing tool), however, there is a migration to using the faster more flexible
tool, CMake. This will enable the ability to cross-compile for the ARM architecture.

Figure 1 shows the software structure for an Athena build (read from bottom
to top), specifically for release 20.1.0. This is the release chosen to be compiled for
ARM where the projects in orange were neglected in the build. These focus more on
“online” computing whereas our focus will be “offline” computing. The LCG software is
comprised of common packages that are experiment independent. This includes around
200 packages such as Python, Boost and ROOT. The LCG software and Gaudi were
compiled natively on the Jetson-Tk1 board using the CMake project. The rest of the
projects (from DetCommon and up) were compiled using CMT in a new framework
called ATLAS Nightly on ARM (ANA). This tool will be explained in the next section.

3. ANA

The objective of ANA is to be able to repeatedly and consistently build versions of
Athena for the ARM architecture with minimal input from the user. In these early
stages of development it’s not feasible to go through the entire ATLAS code looking
for issues that may or may not appear. The approach taken with ANA is to build the
packages while taking note of what fails. The developer then writes a patch that can
ten be applied or reversed. This way, records are kept of what changes are made and
in the case of incorrect diagnoses this can be easily corrected.

It is important to note that only the ATLAS code compiled using CMT has been
Figure 1. The ATLAS software hierarchy with an experiment independent base, LCG software and Gaudi.

included into ANA version 1.0. The means that the LCG software and Gaudi need to be compiled beforehand and still require essential manual tweaks to accommodate the ARM architecture. For example, one of the more crucial tweaks is to define the new ARM architecture in the LCG Platforms requirements file. In this case it means creating the tag armv7l-fc21-gcc49-opt/dbg (the CMTCONFIG variable, which translates to the armv7l architecture on Fedora 21 using GCC 4.9) and defining it as Linux.

The structure of ANA is shown in Figure 2 with an explanation of the more important files/directories shown on the figure.

To run ANA one needs the source code located at https://github.com/jwsmithers/AtlasOfflineBuild-framework. Once this has been downloaded all that is required is to set the environment variables in Environments.sh and then do

```bash
>> source BuildAtlasOffline.sh.
```

When this command is run it creates the correct CMT directory structure, `project.cmt` and `requirements` files, and gives the option to pull the required SVN packages based on the VERSION variable set earlier on (provided this hasn’t been done before, it is a lengthy process). ANA then provides the options to either apply all the patches in the patches directory or skip them, as well as choosing the project that you want to build (or by default building them all). Around 65% of the ATLAS code has been ported to ARM using ANA with the rest closely following. Some basic example scripts (such as “AthExHelloWorld”) have run successfully.
4. ARM related issues

Issues arose when compiling on ARM architecture. This first few apply to LCG software:

- **ROOT-v34-21**: A patch written by a team from CMS [6] was used that enabled the building of Cintex. A further patch was written to enable ROOT to compile on Fedora 21.
- **Grid specific packages**: None of these packages can be compiled as they are specific to the x86 architecture. For now, these are not essential packages.
- **Oracle DB**: There are no ARM 32-bit builds and Oracle doesn’t plan to provide them. It is still being determined how much of a setback this will be.

For the ATLAS software most of the patches add the new ARM related tags into the code. They also fix path related errors due to the software being optimized for tools found in AFS and compiled for x86 architecture. There are still some oddities that need to be mentioned:

- **Syntax errors**: A few syntax errors were found such as missing “=” or “::” signs.
• Linking errors: Some libraries are linking in the wrong order and thus fail the first time being compiled. The temporary solution is to do a “make” twice.

• Genreflex compiler flag issues: The compile flags `-fPIC`, `-fsigned-char` and `-fsigned-bitfields` don’t get removed when the `lcgdict` macro is called. Thus, it sees it as passing an option and so fails. The solution is to add private macros which remove these options every time `lcgdict` is called. This could stem from a tag being defined incorrectly.

• Virtual memory use: Occasionally a package would fail to compile because it exhausts the virtual memory address space. A temporary fix is to just recompile the package.

These highlight some major fixes. There are many more which are all documented in the `patches` directory of ANA.

Summary

With the sudden and gaining interest in the ARM processor we felt it became prudent to build a framework in which stable and reliable ARM builds can take place overnight. The ANA framework builds the ATLAS software stack with minimal input from the users side. Developers also have a clearer view into the layout of ANA and how it functions should they want to make contributions. Currently, around 65% of the entire software stack has been ported to the ARM architecture.

Acknowledgements

The authors would like to thank the National Research Foundation of South Africa and the University of Cape Town for funding this research.

References

[1] ATLAS Collaboration, G. Aad et. al., *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST 3 (2008) S08003.
[2] L. Evans and P. Bryant, *LHC Machine*, JINST 3 (2008) S08001.
[3] “Arm Cortex-A series.” [http://www.arm.com](http://www.arm.com).
[4] G. Barrand et. al., *GAUDI - A software architecture and framework for building HEP data processing applications*, Comput.Phys.Commun. 140 (2001) 45–55.
[5] C. Arnault, *Configuration Management Tool*, 2006.
[6] CMS Collaboration (Chatrchyan, S. et al.), *The CMS experiment at the CERN LHC*, 2008.