Journal of Physics: Conference Series **513** (2014) 052014 doi:10.1088/1742-6596/513/5/052014

Measurements of the LHCb software stack on the ARM architecture

S. Vijay Kartik, Ben Couturier, Marco Clemencic, Niko Neufeld
CERN (European Organization for Nuclear Research), Geneva, Switzerland.
E-mail: vijay.kartik@cern.ch

Abstract. The ARM architecture is a power-efficient design that is used in most processors in mobile devices all around the world today since they provide reasonable compute performance per watt. The current LHCb software stack is designed (and thus expected) to build and run on machines with the x86/x86_64 architecture. This paper outlines the process of measuring the performance of the LHCb software stack on the ARM architecture - specifically, the ARMv7 architecture on Cortex-A9 processors from NVIDIA and on full-fledged ARM servers with chipsets from Calxeda - and makes comparisons with the performance on x86_64 architectures on the Intel Xeon L5520/X5650 and AMD Opteron 6272. The paper emphasises the aspects of performance per core with respect to the power drawn by the compute nodes for the given performance - this ensures a fair real-world comparison with much more 'powerful' Intel/AMD processors. The comparisons of these real workloads in the context of LHCb are also complemented with the standard synthetic benchmarks HEPSPEC and Coremark.

The pitfalls and solutions for the non-trivial task of porting the source code to build for the ARMv7 instruction set are presented. The specific changes in the build process needed for ARM-specific portions of the software stack are described, to serve as pointers for further attempts taken up by other groups in this direction. Cases where architecture-specific tweaks at the assembler lever (both in ROOT and the LHCb software stack) were needed for a successful compile are detailed - these cases are good indicators of where/how the software stack as well as the build system can be made more portable and multi-arch friendly. The experience gained from the tasks described in this paper are intended to i) assist in making an informed choice about ARM-based server solutions as a feasible low-power alternative to the current compute nodes, and ii) revisit the software design and build system for portability and generic improvements.

1. Introduction

As is the norm for each of the four major experiments based on the Large Hadron Collider (LHC) at CERN, physics analysis of the data acquired by LHCb is performed using a fairly complex software stack that encompasses a wide range of computing goals including data parsing, event triggering and event reconstruction. This software stack is a combination of individual programs and libraries written by various physicists and engineers over different times and in different teams. As a consequence, the resulting stack is fairly heterogeneous in terms of coding and tool packaging conventions. With a partial software stack size of around 3.6 million lines of code, porting and building this stack on a relatively unknown architecture(for high performance computing) is a novel exercise and potentially rewarding in terms of the insights to be gained about the behaviour of LHCb software on a non-x86 platform for the first time.

The ARM architecture [1] is the most widely used processor architecture in all modern mobile devices, from tablets to game consoles to cellphones. ARM-based processors are used in billions of devices today, owing to their extremely low power consumption and favourable licensing arrangments with embedded device manufacturers. Despite this overwhelming popularity of ARM in the mobile device industry, it has so far not been considered in high performance computing(HPC) fields, mainly due to the lower performance compared to x86/x86_64 architectures, and power consumption being a lower priority for HPC outfits. This situation is now slowly changing, with ARM-based processors becoming more performant, full-fledged operating systems being released for ARM (this was not the case until as recently as two years ago), and the foreseen introduction of ARM-based computing nodes into the server market by mainstream server vendors [2].

Owing to the upsurge in the popularity of the ARM architecture, we consider it worthwhile to investigate the claims of performance and power consumption, and to test the feasibility of ARM-based servers at LHCb in particular. This paper shall, in addition to the central focus of the effects of the architecture on the performance of ported LHCb software, also touch upon the effects of non-conforming conventions and architecture-specific artefacts of existing code.

2. Cross-build toolchain and porting efforts

A toolchain is the first step in any software porting exercise, and to begin the porting of the LHCb software stack to the ARM architecture, a complete cross-build toolchain was created in order to minimise the turn-around time for multiple builds of the stack. The common tools for cross-building included the GNU Compiler Collection(GCC) and associated packages, which in turn were used in the initial stages of building the ancillary packages like Python, Boost, ROOT [3] (for most computation libraries) and SQLite, COOL, CORAL (for database-related functions) *etc.* After several iterations of building the toolchain, the configuration chosen contained GCC 4.7.2, Python 2.7.1, Boost 1.51 and ROOT 5.34.05.

Cross-building some standard prerequisites(*e.g.* ROOT) required further modifications. Modern tool packaging and building includes so-called 'system introspection' which involves running self-tests and sanity checks with the help of intermediary binaries generated and executed on the fly during the build process. For software packages that are not designed/flexible enough to be cross-compiled, this poses a problem as the intermediary binaries cross-compiled for ARM targets fail to run, subsequently failing the sanity checks and triggering an abort of the build process. As a workaround, a Linux kernel capability, *viz.* 'binfmt_misc' [4] was used to pause the system introspection, ship intermediary binaries to an available external ARM node, run the introspection tests and ship the results of these tests back to the build node to continue the cross-compilation. The problems occuring during cross-compilation may also be avoided by deploying virtual machines and ARM emulators.

The motivation for setting up cross-compilation was to reduce the turnaround time for successive builds and to possibly add an ARM target to the regular LHCb software build process. Further to the aim of cross-compilation for ARM targets on standard build nodes at LHCb, the testing/benchmarking process (described in detail in Section 4) was also used with the integrating test framework used at LHCb for standard production builds [5].

3. Building on multiple platforms

The complete cross-build toolchain has been tested with the versions of the individual components frozen as mentioned in Section 2. For further tests, increasing in both complexity and scale, native builds were performed on two full-fledged ARM operating systems bundled on the following testbenches:

(i) "CARMA" development kit (from SECO Ltd.): A "CUDA on ARM" platform [6] with the following specifications -
- CPU: NVIDIA Tegra 3, a quad-core ARM Cortex-A9 CPU running at 1.3 GHz
- RAM: 2 GB onboard
- GPU: NVIDIA Quadro 1000M with 2GB dedicated memory (not available for general processing)
- OS: Ubuntu 11.04 for ARM with custom filesystem and soft-float kernel

(ii) Viridis server (from Boston Ltd.): An ARM-based "microserver" [7] with the following specifications -
- CPU: Calxeda EnergyCore(TM) system-on-chip, with a quad-core ARM Cortex-A9 CPU running at 1.1 GHz
- RAM: 4GB per node (Upto 48 nodes per microserver)
- TDP: Claimed 5 watt per node maximum power requirement
- OS: Fedora 18 for ARM with hard-float kernel configuration

As mentioned earlier, the complete stack that has been natively built on multiple ARM platforms at LHCb consists of around 3.6 million lines of code and is handled through a 'homemade' configuration management tool [8] designed for quick packaging and distribution among different teams. These two factors contributed to the issues faced during the build process, requiring hand-picking and modifying partial configuration parameters, patching several pre-packaged tools, and in extreme cases, modifying the physics code in order to reach a working build. The most prominent issues are mentioned below.

- Faulty code reflection in ROOT: Reflection features built into ROOT (Cintex, in particular) contain highly architecture specific instructions (as was also pointed out by a team working on ARM at CMS, another LHC experiment), and after reviewing proposed patches and the assembly code, a workaround to disable particular reflection-specific portions of ROOT was put in place, as we concluded that LHCb software (up to Brunel) would remain unaffected by it.
- x86/x86_64-related sanity checks in physics code: Custom sanity checks on datastructure sizes written into the physics code triggered crashes in Boost libraries, which were traced down to certain LHCb packages and modified manually. This could be avoided in general by not making assumptions about the architecture that the code would finally run on.
- Mismatch between compiler output and assembler: Fully functional object code could not be generated from the assembler code that was emitted by the compiler that was custom-built and tuned for ARM. Manual edits of the assembler code served as a workaround intermediate step in the build process. The reasons for the mismatch are not clear, but it is possible that the toolchain options may lead to such behaviour by the compiler.
- Unavailability of source code for certain computing libraries: Source code for proprietary libraries like 'NeuroBayes' [9] was not available, hence LHCb software that used such libraries had to be disabled during the build process.

4. Testing and benchmarking results

The successful build and full port of the LHCb software stack has been a good proof-of-concept in itself for the potential of the ARM architecture to serve as an alternative for running LHCb software, barring performance issues which were found as a result of extensive testing on different platforms on different scales. Results are presented in this section, from benchmarks and tests performed on the test platforms described in Section 3.

Two synthetic benchmarks were run on the different platforms: HEPSPEC (a version of SPEC CPU2006 modified to reflect High Energy Physics(HEP) taskloads better) [10] and Coremark

(a common open-source benchmark used in the embedded device industry) [11]. As can be seen in Fig. 1, the per-core performance of the ARM processors is a factor of 5-7 times slower than both AMD and Intel processors. It must be noted that the AMD and Intel processors tested are server-grade, and were used in platforms with correspondingly high specifications (faster disks, more RAM *etc.*), especially when compared to the CARMA development kit. However, in all foreseen ARM-based microservers, the larger number of cores per server is not enough to cover up the lower performance per core.

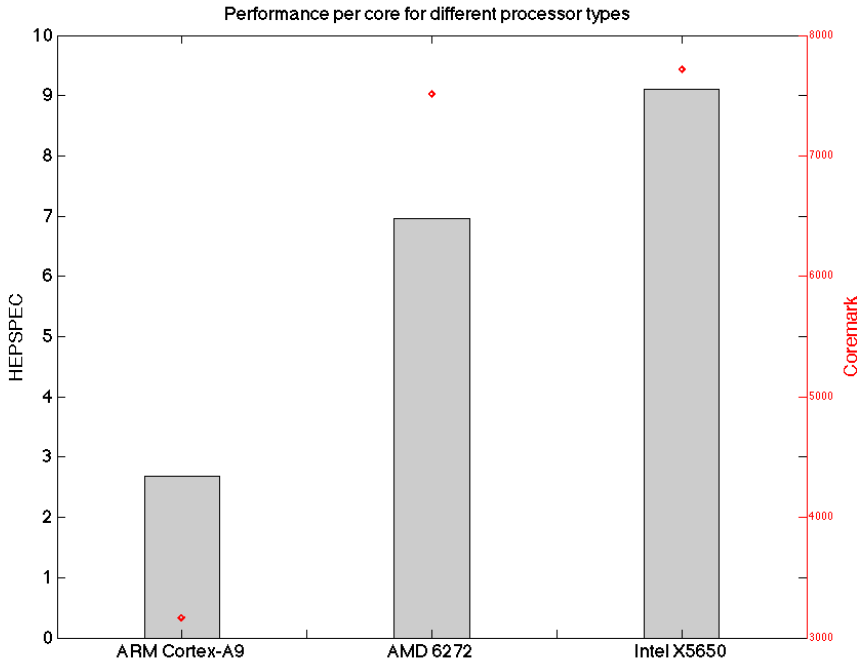

Figure 1. Per-core HEPSPEC performance of different platforms

A real-world benchmarking of the LHCb software stack is equally essential since the synthetic benchmarks do not account for the particular requirements of running the LHCb software online. For instance, much of the code is designed and run in a largely monolithic fashion, executes as a single-thread program (hence limiting programs to run on a single core) and has a predictable and fixed memory footprint. With this in mind, the LHCb online team runs multiple instances of the same program in parallel, subject to available hyperthreading. To measure such workload and usage, the LHCb event reconstruction program 'Brunel' was chosen as a benchmark, and processing time for 1000 events was decided as the comparitive measure across platforms. Event data for reconstruction was taken from proton-proton collisions at LHCb in 2012 with 4000 GeV beams.

As shown in Table 1, ARM-based processors on current 'microservers' are a factor 5 slower on average than a high-performance Intel-based server. The performance of the CARMA development kit is slower by a further factor of 4. The particularly low performance of the CARMA kit may be attributed mainly to the absence of a hard-float kernel: this necessitates all floating point operations to be trapped and re-routed through emulation of a hardware floating point unit(FPU) in software. For intensive computing tasks such as Brunel, this affects the overall processing time very strongly.

Table 1. Processing time required for running Brunel on 1000 events.

	CARMA dev. kit	Boston Viridis	Intel Xeon
CPU+Arch.	Cortex-A9 (ARMv7)	Cortex-A9 (ARMv7)	Intel L5520 (x86_64)
Frequency (GHz)	1.3	1.1	2.27
No. of physical cores	4	4	8(x2 with hyperthreading)
Total RAM (GB)	2	4	48
Total time for Brunel	*ca.* 10 hours	*ca.* 2.5 hours	*ca.* 0.5 hours

A second level of tests was performed by gradually scaling up the number of parallel Brunel jobs on each compute node to load the entire machine. The observed results are shown in Fig. 2. The performance of each individual job is clearly seen to deteriorate with increasing number of parallel jobs on each compute node. This test was performed on a large scale with a fully fitted Viridis microserver with 24 compute nodes. Each individual node remained unaffected by loading jobs on adjacent nodes, although the scaling issues persisted within individual nodes. This points to the RAM bandwidth not scaling well on ARMv7-based systems, as against linear scaling with increasing physical cores observed on x86/x86_64-based systems.

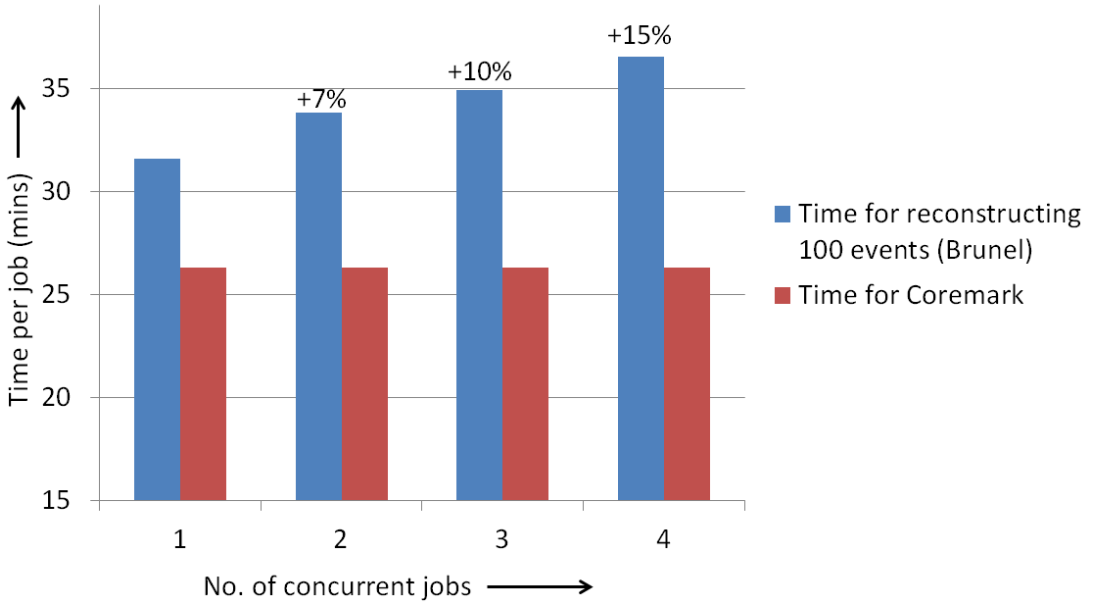

Figure 2. Adverse performance effects by running concurrent jobs on Viridis microserver nodes.

One of the requirements of an alternate processor architecture for computing in HEP is the equivalence of the physics results obtained from all the computation. With correct usage of the FPUs that are now standard on all ARM-based platforms, the final numerical physics results were equivalent to those from x86/x86_64-based platforms (within acceptable limits of precision). A portion of the results of one of the Brunel algorithms is shown here in Fig. 3 as an example. The minor differences may possibly be due to rounding errors and the different number of bits in the architectures ARMv7 and x86_64 (32 bits and 64 bits respectively).

"Cluster Match"	sum		mean/eff^*		rms/err^*	
	ARMv7	x86_64	ARMv7	x86_64	ARMv7	x86_64
#calos	81793	81793	98.784	98.784	51.547	51.547
#chi2	1.117573e+09	1.117574e+09	462.97	462.97	292.95	292.95
#links	2413906	2413907	2915.3	2915.3	3762.5	3762.5
#overflow	8979472	8979471	10845.	10845.	13205.	13205.
#tracks	84166	84166	101.65	101.65	82.226	82.266

Figure 3. Comparison of minor differences between results obtained on ARMv7 and x86_64.

5. Conclusions

The results of the build and benchmarking tests firstly show that ARM is a potential architecture for the LHCb software stack, especially when the low power consumption of ARM processors is taken into account. However, the current performance obtained on ARM-based computing nodes shows that the processing power is not comparable to Intel/AMD processor-based nodes. The low performance per core cannot be offset by the increased number of cores per node for two reasons: firstly, the performance per core is a factor of 5-7 times lower, while the number of cores per node is not increased in the same proportion; secondly, the memory bandwidth available to each core does not scale well at all when the node is taxed with increasing number of concurrent processes. These two factors lead to the conclusion that ARM-based servers are not yet competitive with Intel/AMD-based servers in the high performance computing arena.

Physics results using the ARM-architecture are numerically correct (within acceptable precision), which is a pleasant observation, considering the vastly different and reduced architecture, both in terms of the number of bits (32) and the number of possible instructions. For this reason, it is worth re-evaluating ARM processors when the AArch64 (64-bit) architecture-based ARM processors make their way into mass produced servers.

A by-product of this exercise of rebuilding/porting the LHCb software stack to an as-yet untested architecture has been the identification of systemic deficiencies/features in LHCb software that are, knowingly or unknowingly, dependent on x86-specific features. The packaging and release mechanism for the various tools (Gaudi, Brunel, Phys, Hlt *etc.*) can also be further improved to be more architecture-agnostic, thus requiring fewer modifications to multiple configuration settings to enable successful builds on newer architectures like ARM. Indeed, considerable progress has already been made in this direction by LHCb, where we are now in the process of migrating away from an in-house configuration management tool to CMake, thus enabling neater integration and more flexibility for multiple architectures.

Acknowledgements

We would like to acknowledge the assistance provided to us by the SFT group [12] at CERN in matters related to tool packaging conventions (LHC Computing Grid software details) and ROOT-specific issues. We are also grateful to Boston Ltd. for kindly providing us with remote access to their Viridis platforms for native builds and large-scale performance tests.

References

[1] "ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition", retrieved from *http://www.arm.com/products/processors/instruction-set-architectures/index.php.*
[2] "AMD, Dell, HP announcements for ARM servers", retrieved from *http://www.amd.com/us/press-releases/Pages/press-release-2012Oct29.aspx, http://www.dell.com/learn/us/en/555/campaigns/project-copper, http://h17007.www1.hp.com/us/en/enterprise/servers/products/moonshot/index.aspx.*
[3] "ROOT Homepage", retrieved from *http://root.cern.ch*
[4] "Kernel support for miscellaneous binary formats v1.1" retrieved from *https://www.kernel.org/doc/Documentation/binfmt_misc.txt*
[5] "Jenkins build system configuration" retrieved from *https://twiki.cern.ch/twiki/bin/view/LHCb/JenkinsConfiguration*
[6] "CARMA - The CUDA on ARM development kit" retrieved from *https://research.nvidia.com/news/meet-carma-cuda-arm-development-kit*
[7] "The Boston Viridis" retrieved from *http://www.boston.co.uk/solutions/viridis/default.aspx*
[8] "CMT - Configuration Management Tool" retrieved from *http://lhcb-comp.web.cern.ch/lhcb-comp/Support/CMT/cmt.htm*
[9] "Neurobayes - General information" retrieved from *http://neurobayes.phi-t.de/index.php/public-information*
[10] "HEPSPEC - DVInfo" retrieved from *https://dvinfo.ifh.de/HEPSPEC*
[11] "The Embedded Microprocessor Benchmark Consortium" retrieved from *http://www.eembc.org/coremark*
[12] "SFT Projects - CERN" retrieved from *http://sftweb.cern.ch/*