Journal of Physics: Conference Series **664** (2015) 092011 doi:10.1088/1742-6596/664/9/092011

ATLAS computing on CSCS HPC

A. Filipcic[1], S. Haug[2], M. Hostettler[2], R. Walker[3], M. Weber[2] on behalf of the ATLAS collaboration

[1] Jozef Stefan Institute, Ljubljana, Slovenia
[2] Albert Einstein Center, University of Bern, Bern, Switzerland
[3] Ludwig Maximilian University of Munich, Munich, Germany

E-mail: michi.hostettler@cern.ch

Abstract. The Piz Daint Cray XC30 HPC system at CSCS, the Swiss National Supercomputing centre, was the highest ranked European system on TOP500 in 2014, also featuring GPU accelerators. Event generation and detector simulation for the ATLAS experiment have been enabled for this machine. We report on the technical solutions, performance, HPC policy challenges and possible future opportunities for HEP on extreme HPC systems. In particular a custom made integration to the ATLAS job submission system has been developed via the Advanced Resource Connector (ARC) middleware. Furthermore, a partial GPU acceleration of the Geant4 detector simulations has been implemented.

1. Introduction

Processing and analyzing data taken by the experiments at the Large Hadron Collider (LHC) is a major computational challenge, both due to the amount of data to be distributed, and due to the computing time needed to process it. Currently, data processing is based on a hierarchical, worldwide distributed grid computing system, the Worldwide LHC Computing Grid (WLCG) [1]. WLCG computing sites are usually dedicated clusters specifically set up to meet the needs of the LHC experiments. More than one million grid jobs run on the distributed computing sites all over the world per day on more than 200000 CPU cores.

While this approach showed to be reliable during LHC Run 1, it also bars the LHC experiments from using large shared supercomputers, also referred to as "High Performance Computing (HPC) systems". Purchasing and operating the dedicated computing clusters requires a major effort from CERN and the experiments. Therefore, getting a share on a few supercomputers might be a very efficient approach for large-scale data processing complementary to running many independent computing clusters. Also, the need for computing resources will increase with increased data yields due to the LHC upgrades already implemented for Run 2 or planned for the future.

In this paper, we study the feasibility of running Monte Carlo detector simulation jobs for the ATLAS experiment [2] on Cray HPC systems at the Swiss National Supercomputing Centre (CSCS). For our studies we use "Todi", the HPC integration system for the "Piz Daint" machine at CSCS. We developed a back-end to the Advanced Resource Connector (ARC) Grid Middleware [3] to submit WLCG Grid jobs to a remote HPC batch system over a SSH connection [4], and show that unmodified ATLAS detector simulation jobs based on the Geant4 [5] detector simulation toolkit can run on shared Linux-based Cray HPC systems with Intel-x86 compatible CPUs with user-level changes to the environment. We also show that within the ATLAS software framework,

Geant4 could be replaced by a version optimized for particular HPC systems, e.g. including code to run on accelerators such as GPUs.

2. Implementation

The ATLAS production workflow is quite different from the usual self-contained HPC applications run directly by individual users, and common restrictions of HPC systems (no root access, no possibility to install system-wide software, no internet access from the compute nodes, not being able to run any persistent services) pose different challenges for running ATLAS production jobs. First, ATLAS production jobs depend on the ATLAS software stack and the Athena framework [6], which must be made available in order to run jobs. Second, ATLAS production tasks are managed by PanDA [7] and submitted to the WLCG, so a solution for submitting PanDA jobs to an HPC system jobs, monitoring them and getting back the results in an automated way is required.

2.1. ATLAS Software Access

On WLCG clusters, the ATLAS software repository is made available through the CVMFS distributed file system [8]. Mounting CVMFS requires root access and the FUSE kernel module, which is in general not available on shared HPC systems. In order to still run standard ATLAS jobs, we evaluated two possibilities for accessing and running ATLAS software on HPC systems: the "Parrot" virtual file system wrapper [9], and maintaining a reduced local copy of the ATLAS software repository on a shared file system of the HPC facility.

The "Parrot" file system wrapper uses the "ptrace" process debugging interface of the Linux kernel to attach to a wrapped process, intercept system calls which access the file system and simulate the presence of arbitrary file system mounts, e.g. CVMFS mounted on the common `/cvmfs` mount point. While this approach could allow ATLAS software to run without any modification with minimal local disk space requirement, it turned out to be unreliable in a multi-threaded environment with inter-process communication in our tests (i.e. when running multi-core jobs) due to race conditions not properly handled by the "Parrot" wrapper code causing deadlocks.

Our alternative solution was to copy the versions of the ATLAS Software stack used by large-scale detector simulation jobs from the ATLAS CVMFS software repository to a shared file system available on the HPC system, e.g. project or scratch space, by using rsync from a remote system with CVMFS available. This approach requires the absolute CVMFS paths in several initialization scripts and file catalogs to be patched in order to make the framework relocatable. We developed tools to automate this process and synchronize the local copy to the ATLAS CVMFS repository regularly. We note that each release of the ATLAS software stack is ~500 GB in size, but it also occupies ~500000 inodes on the file system, which is significantly more than average data of the same size and can put significant load on the file system if it is versioned or backed up regularly. It is to be mentioned that while this approach works well for the ATLAS detector simulation jobs we run, it would require a frequently updated local copy of the ATLAS detector conditions database [10] or online database access for running jobs relying on the detector conditions, e.g. event reconstruction.

2.2. Job Management

Running Grid Middleware on HPC systems is usually not allowed; we thus developed a back-end to the Advanced Resource Connector (ARC) middleware which can run outside of a HPC facility, handles data staging and submits jobs to a remote HPC system over SSH. This is described in detail in [4].

3. Performance Considerations

Due to the major differences between standard WLCG sites and HPC systems, the ATLAS code to run was carefully chosen in order to efficiently exploit the resources provided. In particular, data staging through the external ARC Grid interface is slow compared to the performance of an HPC system, and jobs must at least run multi-threaded to efficiently use a single node (if not distributed over multiple nodes of the HPC system) since no node sharing is possible on CSCS Cray machines. For this reason, we chose the Geant4-based full detector simulation step as a workload. It can run on multi-core machines through AthenaMP event-level parallelization, and has low data throughput compared to its CPU load.

3.1. CPU Scaling and Memory Usage

We profiled the CPU scaling with respect to the number of AthenaMP worker threads and the memory usage of a 16-threaded full ATLAS Geant4 simulation processing 100 events, which is a typical job size. As shown in Fig. 1, the scaling is nearly linear with an offset of ~30 minutes due to the initialization and finalization of each job which can only run serially on a single core.

The peak memory usage, as shown in Fig. 2, is lower than ~7 GiB, which is significantly lower than the available 32 GiB per HPC node. Hence we conclude that memory usage is not a limiting factor for multi-threaded ATLAS detector simulation on HPC.

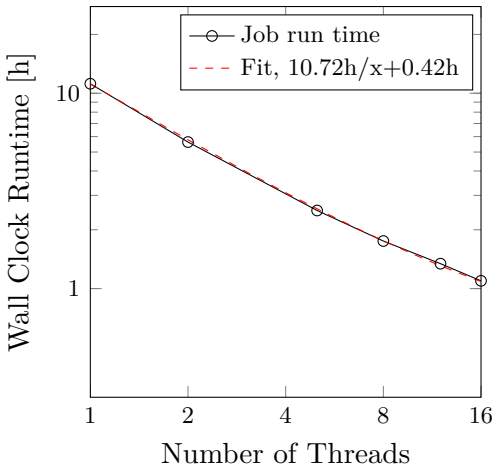

Figure 1. Multi-core scaling of ATLAS Geant4 detector simulation jobs.

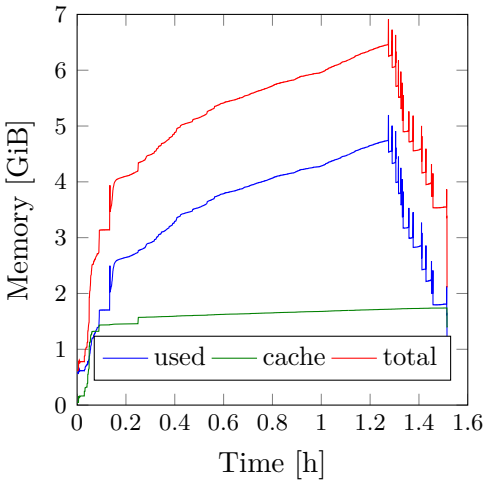

Figure 2. Memory usage of ATLAS Geant4 detector simulation jobs.

3.2. Compiler Optimization

The CSCS Cray HPC systems provide different compiler toolchains. We evaluated whether recompiling the existing ATLAS Geant4 codebase using the Cray Compiler Collection (CrayCC) or the GNU Compiler (gcc), specifically optimizing for the compute node CPU architecture, led to a significant speedup compared to the pre-compiled binaries from CVMFS. Both compilers were run using the settings recommended by Cray [11]. For comparison, a set of 10 events was simulated using the different Geant4 binaries on a single core on the "Todi" HPC integration system. Results for three different random seeds were compared in order to estimate the uncertainty introduced by different series of random numbers generated by the different compilers. The total processing time per event can be found in Table 1.

Our tests show that a speedup of about ~5% is possible by recompiling the Geant4 libraries with gcc platform-specific optimizations enabled, while for our code, the CrayCC build was significantly slower.

Table 1. Processing time per event for different ATLAS Geant4 builds.

Random Seed	Precompiled	Optimized gcc	CrayCC
539155	880 s	834 s	1219 s
939155	879 s	833 s	1208 s
139155	887 s	840 s	1178 s

3.3. GPU Code Tests

Since both the "Todi" HPC integration system and the current CSCS flagship system "Piz Daint" are hybrid CPU/GPU systems featuring NVIDIA Tesla GPUs, possibilities for using the GPUs to accelerate certain parts of the simulations were studied.

At the time of writing there is no Geant4 release running on GPUs. There are ongoing studies on using GPU code in the Geant collaboration [12], but only prototype code exists at this stage, and it is yet to be evaluated if it can and will be implemented into the Geant4 codebase, or if it will be part of the future Geant5 release, which ATLAS computing would have to adapt to and validate first. Due to GPU memory limitations and code divergence penalty, the traditional event-level parallelization aproach is not an option for GPU code.

For this study we considered the performance gain when replacing the Geant4 random number generator (RNG) by a GPU counterpart. For this purpose, we implemented a GPU-based RNG based on the cuRAND [13] library. The generator provides both flat and Gaussian distributions, while using double-buffering and asynchronous data transfer to avoid a bottleneck when copying the generated random numbers from GPU to CPU memory.

Our tests show that generating flat distributed random numbers using our GPU-based RNG is faster by a factor of 5 compared to the RNG previously used by Geant4, while the generation of Gaussian distributed random numbers is faster by a factor of 10. The ATLAS Geant4 codebase was then patched to use this CUDA-based RNG instead of the default Athena RNG. In total, a performance gain of 5% was observed by this improvement. While this is not much on itself, the study proves that dynamically including HPC specific optimizations in ATLAS Geant4 is possible.

4. Production Tests

We tested our HPC integration solution with an allocation on the CSCS HPC integration system "Todi" using standard ATLAS detector simulation production jobs. Our tests included a short-term load test with 100 parallel ATLAS jobs on the machine, and a long-term stability tests with a maximum of 50 parallel jobs running on 50 nodes (with 800 CPU cores in total) over several months.

Over the course of our tests, the ARC-CE front-end was found to be stable, and contributed to ATLAS production at a level comparable to the dedicated Swiss ATLAS Tier-2 clusters, as pointed out in Fig. 3: in total, ~26800 ATLAS production jobs equivalent to more than 500000 CPU-hours were processed on "Todi" by the end of January 2015. For the largest part of the time, the limiting factor was the number of suitable multi-threaded ATLAS detector simulation jobs available.

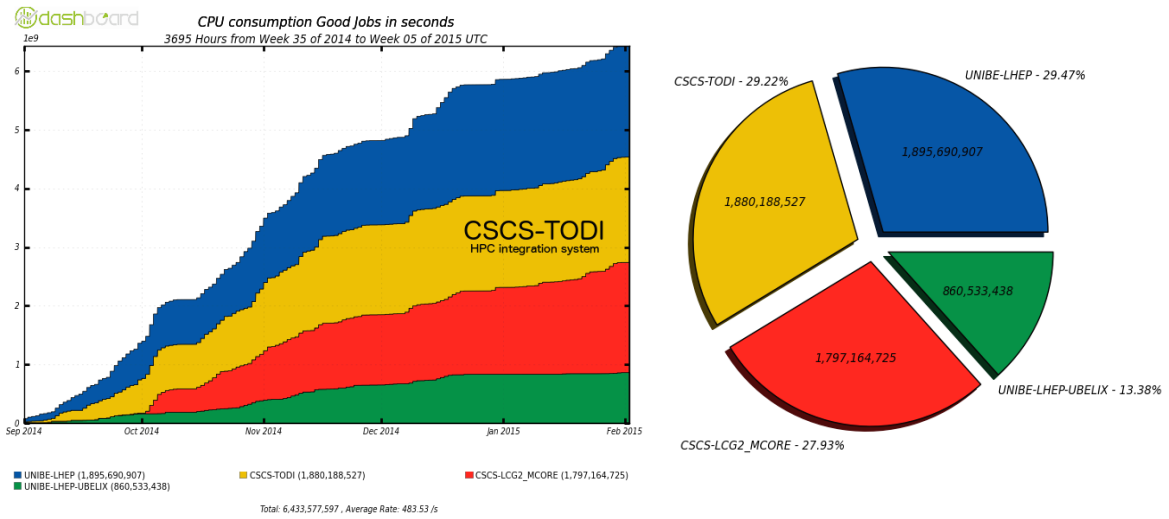

Figure 3. Total CPU time contributed to ATLAS production by our test run on "Todi" and dedicated swiss ATLAS Tier-2 clusters as reported by the ATLAS job dashboard. The contributions to ATLAS multicore production are in the same range.

5. Conclusion and Outlook

We conclude that running the detector simulation step of ATLAS production on HPC systems is feasible. With our ARC based Grid interface and the ATLAS software and databases copied from CVMFS to a shared file system, ATLAS production WLCG jobs can be run on a HPC machine. With our approach, both using an allocated CPU time share and back-filling a HPC system in an opportunistic way by submitting jobs to a low-priority back-fill queue are possible. Since we submit reasonably short single-node jobs, the latter would allow the scheduler of the HPC system to fill gaps between large-scale jobs with ATLAS workloads.

In our tests, our HPC integration approach shows a stable throughput for running 50 parallel jobs (using 16 CPU cores each, 800 CPU cores in total) over several months on the CSCS "Todi" HPC system, contributing more than 500000 CPU-hours to ATLAS production. Submission handling of up to 100 simultaneously submitted jobs worked in a short-term load test. If jobs are released staggered when starting up the external ARC Grid interface (e.g. 50 jobs every few minutes) to flatten the peak load, this implies that hundreds to thousands of jobs running in parallel on the HPC system could be managed. For our test, we run the ARC Grid interface for job management and data staging on a small virtual machine at the University of Bern. This can easily be scaled up in the future in case it becomes a bottleneck. Also, having multiple ARC Grid interface systems submitting jobs to the same HPC system for increased reliability or throughput is possible using our approach.

An adapted version of our ARC HPC interface, submitting jobs to a remote LoadLeveler resource manager instead of SLURM, is currently used on the SuperMUC HPC system[1]. An ARC based integration approach will also be used to run ATLAS production on the Pi supercomputer[2] with first runs successfully completed in January 2015. Plans exist to integrate other Chinese and European HPC systems. In the future, SSH job submission to remote systems will become a common ARC feature, which would allow all supported batch systems to be transparently used over SSH and simplify the integration of further HPC systems using ARC.

[1] HPC system based at the Leibniz Supercomputing Centre in Munich, Germany; adaptation by R. Walker.
[2] Supercomputer based in Shanghai, China; ARC SSH re-implementation by A. Filipcic, J.K. Nielssen and S. Raddum.

Acknowledgements
The authors would like to thank the ATLAS HPC working group for fruitful discussions on this approach, and the Swiss National Supercomputing Centre (CSCS) for providing the HPC resources which made this study possible. In particular, we would like to thank P. Fernandez and M. Gila at CSCS for their support throughout this project.

References

[1] Bird I *et al* 2005 LHC Computing Grid, Technical Design Report *CERN-LHCC-2005-024*
[2] The ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider *J. Instrum. 3 (2008) S08003*
[3] Ellert M *et al* 2007 Advanced Resource Connector middleware for lightweight computational Grids *Future Generation Computer Systems 23 (2007) 219-240*
[4] Haug S *et al* 2015 The ATLAS ARC ssh back-end to HPC *Proceedings of the 21st International Conference on Computing in High Energy and Nuclear Physics, J. Phys.: Conf. Ser.*
[5] The Geant4 Collaboration 2003 Geant4, a simulation toolkit *Nuclear Instruments and Methods in Physics Research A 506 250-303*
[6] Barrand G *et al* 2001 GAUDI - A software architecture and framework for building LHCb data processing applications *Comp. Phys. Comm. 140*
[7] Maeno T *et al* 2008 PanDA: distributed production and distributed analysis system for ATLAS *J. Phys.: Conf. Ser. 119 062036*
[8] Blomer J *et al* 2012 CernVM-FS: delivering scientific software to globally distributed computing resources *Proceedings of the first international workshop on Network-aware data management*
[9] Thain D, Livny M 2005 Parrot: An Application Environment for Data-Intensive Computing, *Scalable Computing: Practice and Experience, 6(3)*
[10] The ATLAS Collaboration 2005 ATLAS Computing Technical Design Report *CERN-LHCC-2005-022*
[11] A. Lazzaro (Cray) 2014 Overview of Compilers on XC30 *CSCS Piz Daint course, Lugano, Switzerland*
[12] The Geant4 Collaboration 2013 GPUs in Geant4 *Annual Concurrency Meeting, Fermilab, USA*
[13] NVIDIA 2015 cuRAND *http://docs.nvidia.com/cuda/curand/*