Journal of Physics: Conference Series **664** (2015) 092006 doi:10.1088/1742-6596/664/9/092006

Simulation of LHC events on a millions threads

J T Childers[1], T D Uram[1], T J LeCompte[1], M E Papka[1,2] and D P Benjamin[3]

[1]Argonne National Laboratory, Lemont, IL, USA
[2]Northern Illinois University, Dekalb, IL, USA
[3]Duke University, Durham, NC, USA

E-mail: jchilders@anl.gov

Abstract. Demand for Grid resources is expected to double during LHC Run II as compared to Run I; the capacity of the Grid, however, will not double. The HEP community must consider how to bridge this computing gap by targeting larger compute resources and using the available compute resources as efficiently as possible. Argonne's Mira, the fifth fastest supercomputer in the world, can run roughly five times the number of parallel processes that the ATLAS experiment typically uses on the Grid. We ported Alpgen, a serial x86 code, to run as a parallel application under MPI on the Blue Gene/Q architecture. By analysis of the Alpgen code, we reduced the memory footprint to allow running 64 threads per node, utilizing the four hardware threads available per core on the PowerPC A2 processor. Event generation and unweighting, typically run as independent serial phases, are coupled together in a single job in this scenario, reducing intermediate writes to the filesystem. By these optimizations, we have successfully run LHC proton-proton physics event generation at the scale of a million threads, filling two-thirds of Mira.

1. Introduction

Historically computing resources for High Energy Physics (HEP) research are built by experiments for experiments. This is also the case in Large Hadron Collider experiments at CERN where computing demand is so great the Worldwide LHC Computing Grid (WLCG or Grid) was constructed. This conglomeration of computing centers is designed to simultaneously run hundreds of thousands of serial jobs. While computing needs are predicted to continue to grow during the LHC Run-II, the growth of the Grid will only follow CPU performance improvements and not increase in aggregate size. The community will need to use the Grid more efficiently and utilize non-traditional resources.

1.1. Why use high performance computing?

There are two main drivers for using supercomputers. First, supercomputers currently offer a window into future CPU technology. The Grid is designed to take advantage of commodity CPUs that, at the time it was conceived, were high power single core processors. However, the chip manufacturers are moving away from ever increasing clock speeds and moving to single dies containing many lower power, lower speed cores. This will slowly push Grid member sites to install these chips as high speed Xeon chips become a more costly niche market for chip makers. HEP software will need to be optimized to effectively use these new highly parallel

CPUs. Supercomputers are highly parallel computers and optimizing HEP software for these machines will prepare HEP for tomorrow's commodity CPUs.

Second, the US Department of Energy (DOE) is investing in supercomputing resources. Current machines are many times the size of the current Grid, and their recently announced successors represent another factor of twenty increase in computing power. Ignoring supercomputers excludes HEP experiments from these powerful resources. More importantly, supercomputers are uniquely suited for performing new physics analyses that benefit from parallel execution. This includes high-multiplicity next-to-leading order calculations and production of highly filtered, low cross-section processes, both of which will dominate LHC Run-II studies.

1.2. Argonne Leadership Computing Facility

This work utilizes the Mira supercomputer hosted by the Argonne Leadership Computing Facility (ALCF). Mira is a 10-petaflop IBM BlueGene/Q system composed of 49,152 computing nodes. Each node contains 16 PowerPC CPU-cores where each core has four physical compute units giving Mira a total capability of running 3.1 million parallel processes.

1.3. Event generators as a guide

The software frameworks of LHC experiments evolved with the Grid such that the software is optimized for serial operation. Porting and optimizing the software for supercomputers requires an effort beyond what is available in this study. Therefore, event generators were chosen as a test subject because they originate from outside LHC experiments and remain largely independent within LHC software frameworks. Targeting these smaller codes retains the possibility for productive output, while reducing the initial effort to migrate the code.

1.4. Alpgen

The leading-order event generator, Alpgen (v2.14) [1], was chosen for this study because it is one of the largest sources of W/Z+jet events. High multiplicity samples (n-jets ≥ 4) are well suited for supercomputers because they are computationally intensive. For example, when generating W/Z+5jets, for every 100,000 events generated only one is written to disk. Reading and writing to disk is handled differently on supercomputers because worker nodes do not have local disks. Data must be transmitted to special nodes, called 'I/O nodes', that interface to physical disk drives. On Mira every 128 worker nodes shares a single I/O node which has a bandwidth limit of 1.25 GB per second for reading and writing to disk.

A workflow for Alpgen event generation jobs running on the grid is diagrammed in Figure 1. Each segment represents an independent execution step where Alpgen is composed of three steps: integration, weighted event generation, and unweighting. The user specifies which mode is being run via the input configuration file (or run-card) that Alpgen reads at the start of each run. The run-card also specifies details about the physics process, applies cuts, defines phase-space, and sets random number seeds.

Typically, the output Alpgen events are showered using Pythia [2] and sometimes have filters applied. This work focused on parallelizing the weighted event generation and the unweighting steps. The integration and showering steps are not time consuming and therefore were not targeted for running on Mira, but instead run on a standard Grid-like machine.

2. Parallelization of Alpgen

2.1. Alpgen-v1

The minimum job size to run on Mira is 512 computing nodes (or 8192 cores each capable of 4 threads). The Message Passing Interface (MPI) was used to introduce parallelization in to

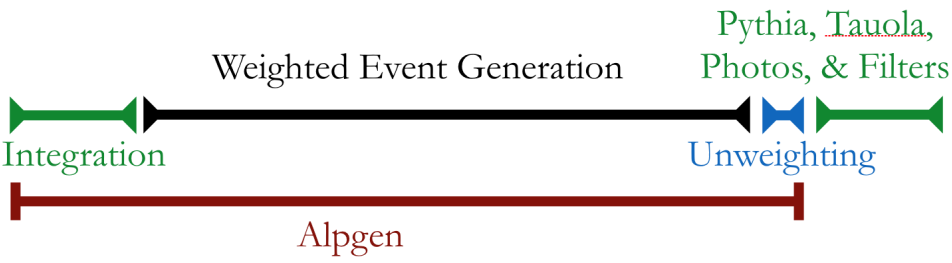

Figure 1. Typical Alpgen workflow running on the grid. Each line segment represents an independent execution step.

Alpgen. The first implementation (referred to as Alpgen-*v1* for clarity) essentially runs parallel instances of Alpgen that are independent and do not communicate. Alpgen-*v1* uses the unique number MPI assigns to each parallel thread, called a rank number, to increment the random number seeds of Alpgen. This avoids identical events being produced by all threads.

This first version focused on parallelizing the weighted event generation step of Alpgen as this is the bulk of the calculations. Since the user specifies which mode is being run, integration, weighted event generation, or unweighting, in the run-card, it is easy to trigger mode-specific parallelization in the code. Alpgen-*v1* can run on 512 Mira computing nodes with 32 threads per node (16,384 threads total). 64 threads per node was tried but overflowed the local memory, which was fixed in a later version.

Before submitting a job to the Mira scheduler a folder must be created for each thread so files would not be overwritten. After a job completed, the output in these directories would be aggregated into a single output and all the folders would be deleted. This pre- and post-processing could be up to 50% of the total run time of an Alpgen job on Mira and increasing the job size from 512 computing nodes to 1024 computing nodes could double the total run time.

2.2. Alpgen-v2

The following steps were taken to address the excessive pre- and post-processing time. The folder structure was removed and Alpgen-*v2* was modified such that only the first thread read files from disk. Then the MPI broadcast method was used to transmit all the file input from the first thread to all other threads over the high speed interconnects. This is much more efficient than all threads reading from a remote disk using the file I/O nodes. This gave a 20% reduction in run-time at 4096 nodes of Mira.

A script was created to run the weighted event generation and unweighting steps together in a single Mira job. This reduces the output data size at the end of the job, for example in W/Z+5jet production the unweighting rejects 99% of all weighted events that were generated. This reduced the post-processing time compared to the previous version because the folder structure is removed and the amount of output data was reduced which shortened the aggregation time.

It was found that Fortran was buffering the output data in memory until Alpgen execution completed instead of flushing the data periodically throughout execution. This causes a spike in data traffic which can approach the bandwidth limits of the file I/O nodes in some cases. To ensure this is not a performance limitation, Alpgen-*v2* includes a flush statement to regularly write the data to disk and maintain a more constant flow of output data. The standard output stream is also limited to one thread to reduce data output and to avoid the slow down that many small writes can cause.

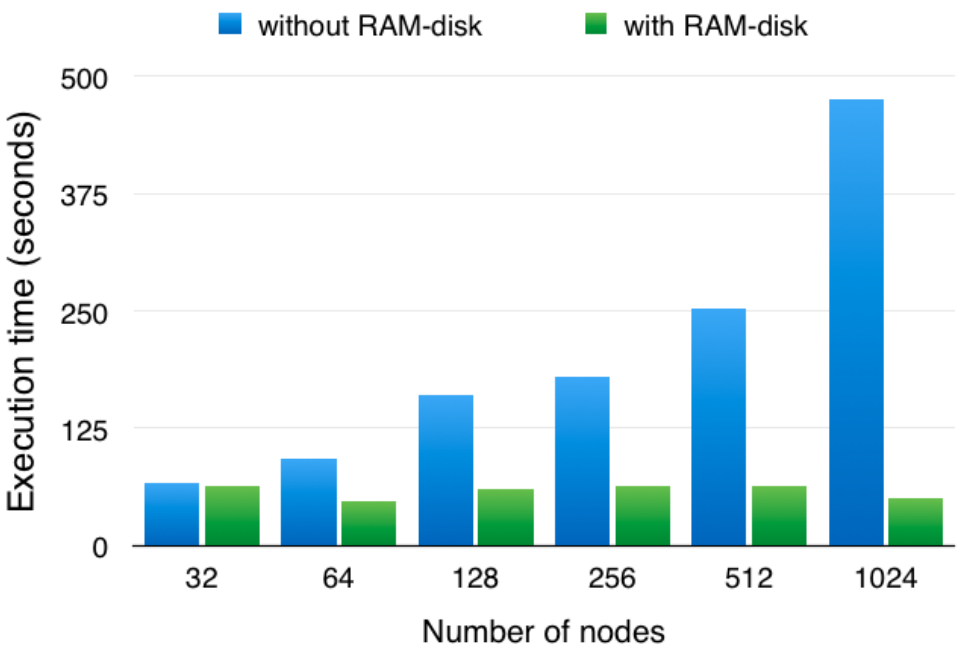

Figure 2. The execution time for Alpgen-*v2* (without RAM-disk) and Alpgen-*v3* (with RAM-disk) versus the jobs size in number of Mira computing nodes.

2.3. Alpgen-v3

Alpgen-*v2* added more intermediate file I/O to the job because the weighted event generation writes files to disk that are then immediately read from disk by the unweighted event generation. The intermediate files are ultimately discarded. A RAM-disk was introduced to avoid writing the intermediate files to remote disk drives. Figure 2 shows the execution time as a function of job size for both Alpgen-*v2* (without RAM-disk) and Alpgen-*v3* (with RAM-disk). By storing this data locally on the computing nodes the job run-time is significantly reduced when going to larger job sizes.

Alpgen-*v3* introduced a third step into the job workflow, output aggregation. After the unweighting, instead of each thread writing its output to remote disks, the files are written to the RAM-disk. A C++-based, MPI-enabled aggregation program reads these output files from the RAM-disk in parallel and uses the MPI coordinated file writing tools to parallelize the writing of an aggregate file to disk.

For the same generation task, Alpgen-*v3* run-times are shorter than Alpgen-*v1* by a factor of twenty. One Mira core running Alpgen-*v1* was $1/15^{\text{th}}$ as effective as a core on the Grid, but Alpgen-*v3* is 1.5x as effective as a core on the Grid.

3. Event Production on supercomputers

This work leverages a DOE ASCR Leadership Computing Challenge (ALCC) Award of 50M core-hours on Mira. The distribution of how the allocation of computing time was used as a function of time is shown in Figure 3. The allocation begins in July 2014 and ends in June 2015. The colors indicate the different job sizes and closely follows the Alpgen version. Each successive improvement to Alpgen enables moving to larger job scales. Alpgen-*v1* jobs use the smallest allowable job of 512 nodes (seen in green in Figure 3). Alpgen-*v2* enabled jobs of the next size class (in blue), 8192 nodes, and Alpgen-*v3* enabled the largest jobs up to the full machine at

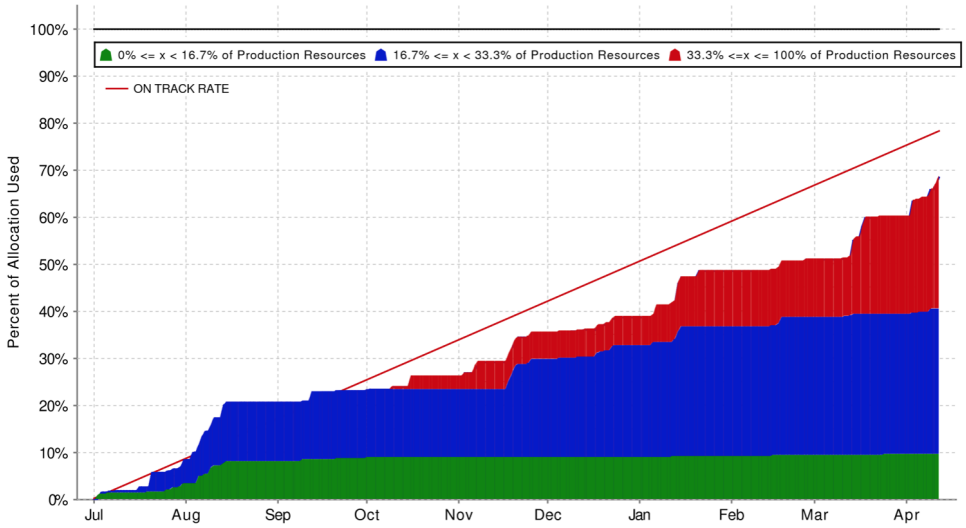

Figure 3. Core-hour usage on Mira as a function of the one year ALCC allocation. The distribution of job sizes is shown using colors. The 'On-Track Rate' indicates the slope needed to fully utilize the allocation by the end of the allocation time period.

49,152 nodes (in red). Each node can execute 64 threads, enabling Alpgen-*v3* to run at a parallel scale of 3M-threads.

The allocation on Mira has been used to generate over 5 trillion weighted W/Z+0-6jets events of which tens of billions of unweighted events have been stored. 6-jet multiplicity samples are not generated on the Grid because it is not practical. The supercomputer makes these samples practical to produce. This is an example where supercomputers can be used to enable new physics processes to be studied.

4. What is next?

Alpgen is a leading-order generator and Sherpa [3], a next-to-leading order generator, is becoming popular with LHC experiments. Therefore, Sherpa will be targeted for optimization on Mira. Sherpa will pose different challenges to Alpgen as it already has some MPI integration in version 2.1.1. Initial tests indicate that the integration step in Sherpa scales to 250 threads efficiently and the event generation step has already been run at scales of 500 parallel threads on Mira. Despite these capabilities, the current Sherpa version shows non-scaling behaviors in the initialization. Removing these scaling limitations will enable growing the job sizes and increase the efficiency of Sherpa.

Event generation represents 10-15% of current Grid usage and while every job run on a supercomputer frees the Grid to do other jobs, simulation represents 60% of Grid usage. This makes simulation a valuable target for running on supercomputers. However, the simulation is typically deeply embedded in LHC experimental software frameworks. This means the entire framework ($\sim$ 7.5 million lines of code) must be ported to supercomputers, or the experiment must produce a slimmed version of the simulation to reduce the code to be migrated. There is work beginning to slim simulation codes.

5. Conclusions

This work shows that HEP experiments can make effective use of supercomputing resources and in some cases can enable the study of new physics processes such as W/Z+6jet sample production on Mira. A year ago, an ad-hoc parallel version of Alpgen was running at small

scales on Mira. After some effort to understand how these supercomputers operate, the largest event generation jobs ever run, with millions of parallel threads, are routinely run at Argonne's Leadership Computing Facility on Mira.

References

[1] ALPGEN, a generator for hard multiparton processes in hadronic collisions, M.L. Mangano, M. Moretti, F. Piccinini, R. Pittau, A. Polosa, JHEP 0307:001,2003, arXiv:hep-ph/0206293.
[2] An Introduction to PYTHIA 8.2, T Sjstrand, et al., arXiv:1410.3012 [hep-ph].
[3] Event generation with Sherpa 1.1, T. Gleisberg and S. Hoche and F. Krauss and M. Schonherr and S. Schumann and F Siegert and J. Winter, JHEP 02 (2009) 007 arXiv:0811.4622 [hep-ph].