Computer Physics Communications 210 (2017) 54–59

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Adapting the serial Alpgen parton-interaction generator to simulate LHC collisions on millions of parallel threads

J.T. Childers [a,*], T.D. Uram [a], T.J. LeCompte [a], M.E. Papka [a], D.P. Benjamin [b]

[a] Argonne National Laboratory, Lemont, IL, USA
[b] Duke University, Durham, NC, USA

ARTICLE INFO

Article history:
Received 21 November 2015
Received in revised form
14 March 2016
Accepted 22 September 2016
Available online 29 September 2016

Keywords:
Supercomputer
HEP
Simulation
Parallel

ABSTRACT

As the LHC moves to higher energies and luminosity, the demand for computing resources increases accordingly and will soon outpace the growth of the Worldwide LHC Computing Grid. To meet this greater demand, event generation Monte Carlo was targeted for adaptation to run on Mira, the supercomputer at the Argonne Leadership Computing Facility. Alpgen is a Monte Carlo event generation application that is used by LHC experiments in the simulation of collisions that take place in the Large Hadron Collider. This paper details the process by which Alpgen was adapted from a single-processor serial-application to a large-scale parallel-application and the performance that was achieved.

1. Introduction

The US Department of Energy (DOE) continues to invest in scientific computing, driving supercomputers to reach the exaFLOPS scale by the early 2020s. High Energy Physics (HEP) experiments have typically relied on internal resources for computing, with the largest example of this being the Worldwide LHC Computing Grid (WLCG or simply Grid) [1]. The Grid currently provides ∼150,000 concurrent computing cores to each of the experiments ATLAS and CMS, which is equivalent to 1.3 billion core-hours per year. The average INCITE computing award in 2015 is over 100 million core-hours per year on the massive parallel supercomputers hosted at current DOE Leadership Computing Facilities. The largest award is 280 million core-hours. The new machines coming online in 2018 will be a factor of twenty larger. The HEP community is preparing to use these new machines by demonstrating the ability to run on current machines.

There are general rules for using supercomputers; avoid filesystem access because it is slow, make use of the high speed cpu-to-cpu connections because they are fast, and run-time memory is limited so share it when possible. This article describes the methods used to follow these rules when running an HEP application written for serial desktop CPUs and how the answers are not always obvious.

Alpgen [2], a FORTRAN-based leading-order multi-parton generator for hadronic collisions, is used as an example case. The authors optimize the simulation's workflow to run on Mira, the fifth fastest supercomputer in the world, and describe their methods for reaching scales of one million parallel threads in order to facilitate future efforts in HEP.

Motivation. HEP experiments have traditionally constructed dedicated computing resources needed for simulating and analyzing data produced at accelerator facilities like Fermilab and CERN. However, the computing needs of the LHC experiments at CERN are expected to outpace the resources of the Grid during Run-II. Some of this growth is driven by computationally intensive calculations such as rare processes and matrix elements calculated at next-to-leading order, which are optimal tasks for a supercomputer. Every computing cycle offloaded to supercomputers frees a Grid cycle for other work. Supercomputers offer large, experiment-independent computing resources that should become an integral part of the HEP computing strategy.

The Grid is composed of Xeon-class servers and was designed in the era of single-core CPUs with ever-increasing clock speeds. This drove the idea of high throughput computing where performance was measured in the number of serial jobs the system could execute simultaneously. In five years, third-generation Xeon Phi chips with core counts near one hundred will likely replace 16-core Xeon chips as the server-class, commodity CPU. The Grid and millions of lines of experimental code are slowly transitioning to this many-core, parallel-processing model of next generation processors. Existing supercomputers already use this model. Using

* Corresponding author.
E-mail address: jchilders@anl.gov (J.T. Childers).

http://dx.doi.org/10.1016/j.cpc.2016.09.013
0010-4655/

them drives applications toward smaller memory footprints and efficient parallel algorithms resulting in code that will easily port to future parallel environments on the Grid.

Argonne Leadership Computing Facility. The Leadership Computing Facility at Argonne National Laboratory (ALCF) hosts Mira, the fifth fastest supercomputer in the world. HEP researchers at Argonne participate in the ATLAS experiment at CERN. This study was performed to assess the usefulness of supercomputers for LHC experiments and the challenges in making code that effectively uses the resources.

Mira is composed of 786,432 1600 MHz PowerPC A2 cores using the BlueGene/Q architecture, with a peak capability of ten petaFLOPS. Each computing node has 16-cores and 16 GB of RAM. Each core has four hardware threads. There are no local disks on the computing nodes. Dedicated file-I/O nodes mediate access to the remote GPFS filesystem with one file-I/O node handling file-I/O requests from 128 computing nodes with a peak bandwidth of 1.25 GB/s. A high-speed 5D Torus network provides 2 GB/s chip-to-chip communication.

Vesta is a smaller testing and debugging system with 2048 nodes, each with 16 PowerPC A2 cores and 16 GB RAM. In contrast with Mira's compute node-I/O ratio of 128, the I/O infrastructure of Vesta is configured such that 32 computing nodes share a single I/O node.

Alpgen. Alpgen is a FORTRAN-based simulation of multi-parton interactions in hadronic collisions. Its operation is summarized below, but more details can be found in [2]. Alpgen is chosen as it is used in roughly half of the ATLAS experimental results published thus far.

The Alpgen code execution is divided into three stages: initialization, event generation, and finalization. The initialization stage accepts user input to configure the physics processes being simulated, determine which simulation step (defined in next paragraph) is being executed, and other software characteristics, e.g. input/output file names. If no user input is provided defaults do exist, but in this work input configurations are always provided. The event generation is a loop in which each iteration simulates one multi-parton interaction (from here on referred to as an event). The number of events generated is defined by the user input.

The simulation is run as three steps: grid generation, weighted event generation, and unweighting. Each step is an independent execution of the Alpgen binary where a user input parameter (`imode`) defines which step is being run in the current execution. Each step produces output files, with the following step using those output files as inputs. Algorithm-wise the grid generation and the weighted event generation do the same thing, i.e. generating weighted events within the allowable phase space. During grid generation, events are generated and their weights are recorded in a phase space map, or grid. Two files, referred to later as grid files, are produced in this step that contain the phase space grid. All Alpgen files are stored as plain text.

The weighted event generation loads the phase space map into memory, generates weighted events, updates the grids with the new event's weight, and records the event in an output file. The grid is used to weight the selection of phase space while generating events. This reduces the number of events generated in regions more likely to be discarded in the unweighting step. To save disk space, the weighted event is recorded as the two random number seeds needed to regenerate the parton momentum four-vectors and the event's weight. This is 57 bytes per event compared to the 551 bytes per event needed to store the full event details in the case of $W + 5$jets. A parameter file is also produced which summarizes the cross-section of the process simulated with uncertainty, the number of weighted events generated, a copy of the phase space grid, and a copy of the input configuration. The parameter file ranges in size from 2 KB to 20 KB for $W + 0$jets to $W + 5$jets, respectively. In practice, the grid and weighted event generation steps can be performed in a single execution of Alpgen, but for the purposes of running on a supercomputer they were kept separate.

The unweighting step loops over the event weights (weights can be $0 \leq w \leq 1$) and regenerates the momentum four-vectors for events satisfying the condition $w \geq r \cdot W_{max}$ (W_{max} is the max weight in the sample, r is a random number where $0 \leq r \leq 1$). Unweighted events have a weight of one by definition. The unweighted events are stored as a list of partons using the particle identification numbers specified in [3] and their momentum four-vectors. Each event starts with a header followed by the two colliding partons and then the produced partons. In processes with vector bosons in the final state, their decay products can be specified in the input configuration and the momentum four-vectors of the products are included in the unweighted output. Events sizes range from 231 bytes to 551 bytes for $W + 0$jets to $W + 5$jets, respectively. There is also a parameter file written containing a summary of the configuration used for the production, the cross-section, luminosity, and the number of events produced. This file is typically 2 KB.

LHC experiments run Alpgen simulations on the Grid by consecutively running the three steps. Each step represents a single execution of the Alpgen code.

2. Adapting Alpgen for a supercomputer

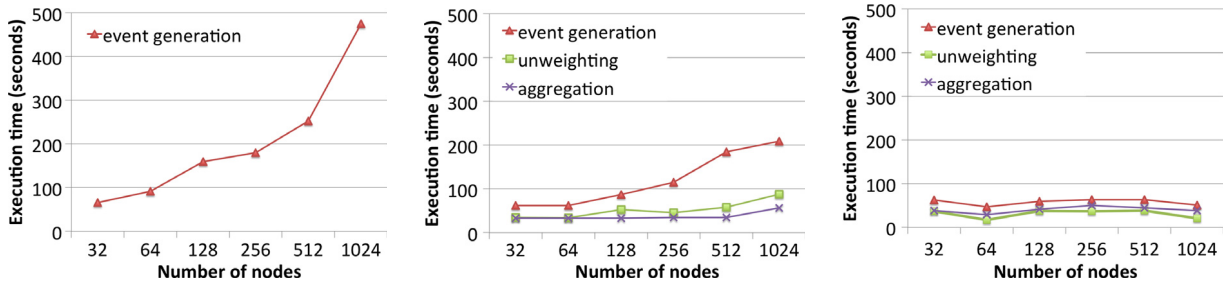
The focus of this article is simulating $W^{\pm}/Z$ vector-bosons in association with 4 or more partons (or jets) because these processes are more computationally demanding. For instance at $W + 5$jets, the number of weighted events generated to unweighted events recorded is of the order 10,000 : 1, compared to 10 : 1 at $W+0$jets. Thus, there is more computation per unweighted event recorded to disk making it better for supercomputers, which are optimized for computation not file operation.

The weighted event generation is the first target for parallelization in *Alpgen-v1*. Next, *Alpgen-v2* uses a script to run as a single job the weighted event generation, unweighting step, and adds an aggregation step. Finally, *Alpgen-v3* moves all the intermediate files from the filesystem to the compute-node memory. These developments attempt to minimize the job run-time and improve performance at large scale, while not making changes to the physics algorithms. Therefore, changes focus on the workflow details and filesystem usage. The ideal outcome for optimization is a code in which run-time is independent of the number of parallel threads; achieving this in light of particular application behaviors and infrastructure is challenging.

In all three versions, the grid generation is run serially on a local cluster before the weighted event generation to ensure each parallel Alpgen begins with the same input. The local copy of the input grid is updated with each newly generated weighted event so each parallel Alpgen will slowly diverge from their common starting point. To mitigate this divergence, the initial input grids are produced with a cross-sectional uncertainty of $<1\%$.

Preprocessor directives are used to optionally include the changes described below during compilation of Alpgen.

Alpgen-v1. The first parallel version of Alpgen, referred to as *Alpgen-v1*, runs only the weighted event generation step in parallel. The Mira queue system requires a minimum job size of 512 compute nodes and charges for all 16 cores per node. This sets the goal of *Alpgen-v1* to be simply running Alpgen on 512 nodes with 16 threads per node (written as 512×16 henceforth) on Mira with each thread generating different events. The three primary changes to Alpgen are summarized below that achieve this goal. First, an API is introduced to run parallel instances of Alpgen. Second, the random number seeds of each parallel instance are updated to generate different events. Third, each parallel instance makes



(a) *Alpgen-v1*. (b) *Alpgen-v2*. (c) *Alpgen-v3*.

Fig. 1. Weak scaling of Alpgen versions for the weighted event generation, unweighting, and aggregation phases. These tests were run with 32 ranks per node to simplify the comparison across versions.

a change working directory call to move to different directories avoiding each instance from overwriting files with the same name.

The Message Passing Interface (MPI) [4] is used to run codes in parallel. MPI handles the creation of parallel processes and provides an API for implementing inter-process communication within an application. MPI applications are run by prefacing the binary on the command line with `mpirun -n <N>` where *N* is the number of parallel instances to run. In fact, non-MPI applications can be run using `mpirun` and *N* instances will be run concurrently. MPI implementations can be as simple as multiple concurrent, independent processes or more complex with highly interdependent concurrent processes, inter-process communication, and shared memory structures. *Alpgen-v1* is of the simple variety.

To avoid all Alpgen instances generating identical events, the random number seeds that are read in from the user input file must be different. MPI assigns each parallel process a *rank* number, n_{rank}, where $0 \leq n_{rank} < N_{ranks}$ with N_{ranks} being the total number of ranks. This rank number is retrieved at run-time and used to change the random number seeds of each rank. To do this, three MPI functions calls were added to the initialization stage of Alpgen. MPI must be initialized by calling `MPI_INIT` in order to use any other MPI functions. Then, the rank number is retrieved with `MPI_COMM_RANK`. No additional MPI functionality is required in *Alpgen-v1* so `MPI_FINALIZE` is immediately called (again required by all MPI applications). No more MPI functions can be called after finalization or an error will occur and execution will cease. Two numbers, `iseed1` and `iseed2`, are provided in the user input file to seed the random number generator at the start of the weighted event generation. The rank number is used to increment `iseed2`.

Prior to running Alpgen a directory structure is created such that each parallel instance has a unique directory in which to write output files. The directory names range from 0 to $N_{rank} - 1$ with five characters each (padded with zeros, so rank 0 writes to '00000'). The Alpgen processes start in the same working directory and change to these rank-wise working directories at run-time.

To summarize, this is an example of running *Alpgen-v1* end-to-end.

1. The serial grid generation is run on a standard cluster.
2. The two grid files and the user input files are copied to the supercomputer.
3. N_{ranks} directories are created into which the grid files and a PDF file are copied.
4. *Alpgen-v1* is submitted to the supercomputer where it generates different events in each directory.
5. Upon completion, the weighted event files and output parameter files are aggregated into a single file.
6. These two files are copied back to a standard cluster where the unweighting is run.

This MPI implementation and workflow enables *Alpgen-v1* to run at the smallest job size allowed on Mira, 512 nodes.

Weak scaling runs of *Alpgen-v1* were conducted from 32 to 1024 nodes on Vesta (a small test system for Mira), with 32 ranks per node. Alpgen was configured to generate 100,000 weighted events per rank. It is not surprising that *Alpgen-v1* scales poorly, as seen in Fig. 1(a). This first naive attempt to run on a supercomputer included minimal code changes and did not address the common guidelines for using these machines, i.e. file I/O is slow so avoid it and reduce memory usage to increase parallel process count.

As described in the Introduction, Mira file-I/O nodes have a bandwidth of 1.25 GB/s per 128 compute-nodes. At the start of weighted event generation, each rank of *Alpgen-v1* reads one input configuration file, one Parton Distribution Functions (PDF) file, and two grid generation files, which is about 15 MB per rank. At the end of generation, each rank writes an updated grid file, a parameter file, and an event file, which are of the order 10 GB per file-I/O node. This is 12 GB total per file-I/O node, which is well within the file I/O node's capabilities. Based on this, one might naively think the file I/O is not a problem, which would be correct if one rank per file-I/O node was writing this data into single files. However, in this case, every rank is reading 4 files and writing 3 files. Like other parallel filesystems, Mira's GPFS filesystem scales poorly in the face of file-per-rank I/O.

Another standard rule in using supercomputers is to limit standard output (`STDOUT`). Each rank in *Alpgen-v1* writes data to `STDOUT`, which is collected by MPI and aggregated at the head node for the job. This flood of small, per-rank output messages stresses the communication infrastructure and prolongs the execution time while messages are collected and flushed to the filesystem.

The creation and removal of per-rank directories, which is needed to avoid file-I/O collisions between ranks in *Alpgen-v1*, contributes significantly to the overall end-to-end run-time of the weighted event generation step. Relying on the filesystem to isolate threads causes the job run-time to double when the number of threads is doubled, thereby limiting job sizes.

Finally, Alpgen-v1 is constrained by its memory footprint to use only 32 of the possible 64 compute threads per node on Mira. 64 instances of *Alpgen-v1*, with a memory footprint of ≈200 MB each would overwhelm the 16 GB of memory available on each compute node (the operating system consumes 2 GB). A small reduction in the memory footprint, therefore, would allow a doubling the number of parallel processes.

Alpgen-v2. *Alpgen-v2* improves the file access strategy using MPI, silences the per-rank `STDOUT`, combines the weighted event generation and unweighting – and a new file aggregation phase – into a single Mira job script, and reduces the executable memory footprint to support 64 Alpgen ranks per Mira node.

Filesystem access is reduced by restricting the reading of common files (user input, grid inputs) to one rank. This leverages the 2 GB/s node-to-node network over to the 1.25 GB/s file-I/O

nodes shared by 128 compute nodes. Unnecessary output files are disabled that relate to data analysis and run-time status. Output file names are updated to contain the rank number making unique run-time directories no longer necessary eliminating the per-rank overhead in pre-processing and greatly reducing the overhead in post-processing.

The Mira system supports running jobs using scripts that run sub-jobs. The weighted event generation and unweighting phases are combined using such a script to reduce the output data size which in turn reduces the post-processing time by at least a factor of two and grows non-linearly as the number of parallel instances increases. A single thread of *Alpgen-v2* writes two files to the filesystem, a weighted event file and a parameter file during weighted event generation. The unweighting step reads these files from the filesystem, and the unweighted events are written to the filesystem. Coupling these steps reduces the output data size: in the case of $W + 5$jets, though the unweighted events are larger than weighted events by a factor of 10, the unweighting reduces the number of events by a factor of 10,000 leading to a reduction in data size by a factor of 1000.

After unweighting is completed, Alpgen stores details of the cross-section and total events produced in a parameter file. *Alpgen-v2* aggregates this information using MPI reduction operations, such as `MPI_SUM`, to make a single global output parameter file avoiding another per-rank file output.

An aggregation step is included in the combined script. This runs a C-based MPI application to read the rank-wise unweighted event files and aggregate the data into a single output file using collective MPI I/O functions such as `MPI_WRITE_AT_ALL`. This is done in post-processing for *Alpgen-v1*. Aggregating files with MPI tools reduces the time needed for this step by a factor of 10 for a 512×32 size job. Aggregating the output also reduces the transfer time from Mira to the Argo cluster when the job has finished, because data transfer protocols, such as GridFTP in this use case, are slower when handling many small files.

`STDOUT` is limited to the $n_{rank} == 0$ rank to diminish the overhead of collecting many small messages over MPI, while retaining some output for debugging and logging purposes. Since the output is largely ignored with increasing scale, it can be disabled without impacting application diagnostics.

Alpgen-v2 reduces the memory footprint to achieve greater node-level parallelism. The original version of Alpgen has a run-time memory occupancy of 203 MB. Alpgen allocates memory in the data section of the executable for the eight possible PDFs, whereas only one is used during execution. *Alpgen-v2* is altered to include only the single PDF specified by the input configuration, reducing the memory footprint to 10 MB and allowing for a higher thread count per node. In this configuration, Alpgen is able to run with 64 MPI ranks per node, whereas previously it was memory-constrained to running with only 32 ranks per node.

Weighted event generation in *Alpgen-v2* exhibits much better scaling, as shown in Fig. 1(b). The figure shows the unweighting and aggregation steps perform well up to 512 nodes, with an increase in execution time at 1024 nodes. The limiting factor in weighted event generation is the writing of one file per rank to the filesystem, which is addressed in *Alpgen-v3*.

Alpgen-v3. *Alpgen-v3* improves file access by writing intermediate files to the compute-node persistent memory [5] (i.e. RAM-disk) on Blue Gene Q systems. The intermediate files, i.e. weighted event data and parameter files and rank-wise unweighted data files, are stored in the RAM of each compute node. The aggregation step is then used to read the rank-wise unweighted data files from the RAM-disk, and output a single data file to the filesystem. This strategy leverages the many-fold faster compute node memory in place of the filesystem, eliminating round-trip write/read cycles to the filesystem and avoiding the problem of many small file accesses. In the cases of $W + 5$jets, $W + 4$jets, and $W + 3$jets, the average rank produces 188 KB, 1.2 MB, and 3.0 MB of temporary data, respectively, when producing one million weighted events.

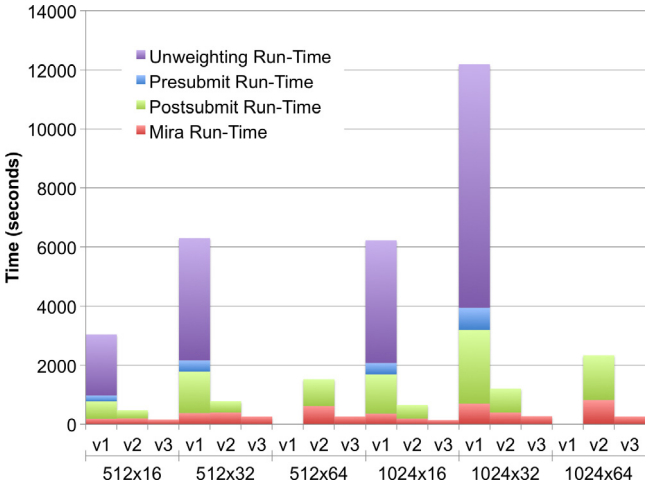


Fig. 2. Run-times for each Alpgen version for different node and thread per node configurations of Mira jobs. Presubmit run-time includes any file placement or directory creation needed prior to running the Mira job. Postsubmit run-time includes any file aggregation and file or directory cleanup after running the Mira job. The unweighting run-time represents the time taken to run the unweighting task on the weighted events, on the post-processing cluster.

Weighted event generation, unweighting, and aggregation all benefit from storing intermediate data files in persistent memory as opposed to the filesystem. Fig. 1(c) shows that the execution time for all three phases is nearly flat up to 1024 nodes.

End-to-end workflow run-time analysis. The three versions of Alpgen described above were tested with varying numbers of nodes and threads per node to measure the impact on run-time. The combinations 512×16, 512×32, 512×64, 1024×16, 1024×32, and 1024×64 are shown in Fig. 2, where the different Alpgen versions are represented as *v1*, *v2*, and *v3*. *Alpgen-v1* was not able to run with 64 ranks per node because its memory footprint exceeded the available system memory at this size. These Alpgen tasks were $Z+2$jets with each thread producing 350,000 weighted events. The task run-time is divided into these steps:

- The Mira run-time represents the actual job run-time on Mira and is the only step running parallel processes.
- The presubmit run-time represents the time needed to prepare to run on Mira. This only exists for *Alpgen-v1* as it required building the per-rank directory structure described above.
- The postsubmit run-time represents the time needed to clean up after a run. This only exists for *Alpgen-v1* and *Alpgen-v2*. In the former case, this accounts for aggregating the weighted event generation files and removing the directory structure. In the later case, this accounts for the removal of all the intermediate files which do not need to be kept and are stored in the RAM-disk in *Alpgen-v3*.
- The unweighting run-time accounts for the time to unweight the weighted events produced in the Mira job for *Alpgen-v1* only. *Alpgen-v2* and *Alpgen-v3* include the unweighting step in the Mira job.

The run-time of *Alpgen-v1* grows linearly with job size, as can be seen by comparing runs with increasing number of ranks; for example, 512×32 vs 1024×32. The primary reason for this poor weak scaling behavior lies in the time for the serial execution of presubmit, postsubmit, and unweighting steps to be performed on the Mira login nodes or the local cluster.

Considering overall runtime, *Alpgen-v2* exhibits more than $>6\times$ speedup over *Alpgen-v1*, primarily due to the unweighting

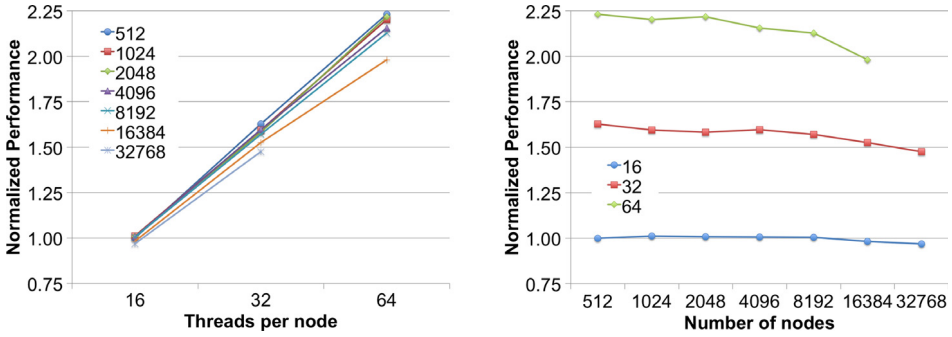


Fig. 3. (left) Normalized performance of *Alpgen-v3* as a function of the number of ranks running on each compute node. The curve is shown for jobs with a varying total number of compute nodes. (right) Normalized performance of *Alpgen-v3* jobs as a function of the total number of compute nodes. The curves correspond to different number of Alpgen ranks running on each compute node. All jobs are $W + 5$jets running 450,000 weighted events per rank.

step being run in parallel after being combined with the weighted event generation step on Mira, in which case the unweighting time has shrunk to be no longer evident on the chart. The presubmit time has similarly vanished from the graph. Postsubmit time remains and increases correspondingly with the number of ranks.

The run-time of *Alpgen-v3* consists entirely of the time for the weighted event generation, unweighting, and aggregation to run on Mira, for an overall speedup of more than 20× than *Alpgen-v1*.

3. Parallel scaling performance

The performance of Alpgen on Mira can be characterized by the run-time of identically configured jobs with differing parallel size. Computing time on Mira is charged per core-hour; therefore, the performance metric will be defined as the number of unweighted events, N_{evt}, per Mira core-hour, C:

$$P = \frac{N_{evt}}{C} = \frac{N_{evt}}{tN_{node}N_{core}}. \quad (1)$$

t is the job run-time in hours, N_{node} is the number of nodes in the job, and N_{core} is the number of cores per node which is always 16 for Mira. Increasing P indicates better efficiency.

The performance is driven by both the total number of nodes and the number of threads per node. Using the total number of threads ($N_{core} \times N_{node}$) is ambiguous because, for example, a 512 × 32 job and a 1024 × 16 job contain the same total number of threads. However, the performance of these two configurations may vary. The thread count per CPU-core is higher in the 512 × 32 job resulting in a higher CPU-load that can increase run-time. If the job is file-I/O intensive, recalling that 128 worker nodes share a single file-I/O node, the 512 × 32 job will generate more network traffic through the file-I/O node, which can also make job run-times longer.

Fig. 3 shows the performance of *Alpgen-v3* as a function of threads per node and number of compute nodes. These jobs represent the performance for generating $W+5$jets with each rank generating 450,000 weighted events. The 512 × 16 configuration is used as a reference; therefore, all points are normalized to the performance of this configuration, $P/P_{512\times16}$.

Having made no algorithmic changes to Alpgen, the performance achieved at large parallel scales is impressive. The data shows the best performance at 512 × 64 which is 2.2× more efficient than the baseline, with slowly decreasing performance as the number of nodes increases. The largest job, 16,384 × 64 or 1 million threads, gives 2.0× better performance than the baseline, down 11% from the best. If core-hour cost is the driving factor this data motivates the use of jobs with fewer nodes. However, for LHC event generation, throughput is more important and current queuing policies on Mira give larger jobs higher priority. Therefore, the authors typically run in 16,384 × 64 configurations to achieve a higher throughput at the expense of a small decline in efficiency. The job run-time is another consideration for generating large sets of data; for instance, a 1 fb^{-1} $W + 5$jets sample runs for 30 min at 16,384 × 64 as compared to more than 13 h at 512 × 64. The maximum job duration on Mira is longer for larger jobs: 12 h for 512-node jobs, but up to 24 h for jobs with 8192 nodes or more. Jobs with 16,383 nodes constitute, therefore, a fair compromise that maximizes time-to-solution at the expense of an acceptable loss of efficiency.

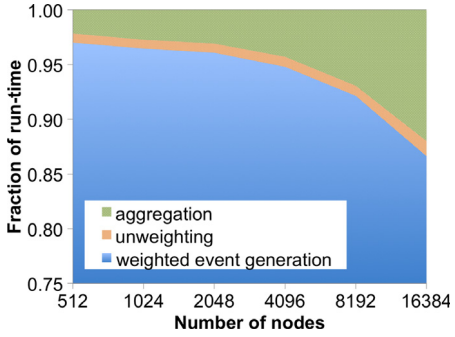


Fig. 4. The fraction of the total run-time taken by the weighted event generation, unweighting, and aggregation steps at 64 threads per Mira node of *Alpgen-v3* for different numbers of nodes. All jobs are $W+5$jets running 450,000 weighted events per rank.

The performance begins to decline beyond 8192 nodes. Fig. 4 shows how the fraction of run-time changes with increasing job size at 64 threads per node. The aggregation is the limiting factor because *Alpgen-v3* combines all unweighted event files to a single file. To do this, MPI communicates the per-rank data to a single rank to be written to the filesystem, limiting throughput. This could be optimized by aggregating to some subset of files, e.g. writing one file per some number of ranks; in the case of Mira, this approach would ideally target the ratio of compute nodes to I/O nodes of 128:1. The increase in the aggregation run-time as the job size increases is responsible for the performance difference between the 512 × 64 and 16,384 × 64 configurations in Fig. 3.

4. Conclusion

Event generation is essential for LHC experimental studies. With Run II currently underway, and the high-luminosity upgrade planned for Run III, an order of magnitude increase in the required computing resources is expected, outpacing the expected growth in Grid capacity. Meanwhile, the trend in leadership computing is toward orders of magnitude larger systems in the coming

decade. It is strategic to adapt codes to run on these systems to deliver increased capacity today and to prepare for future systems. This work describes adapting the serial application Alpgen to run as a large-scale parallel application on Mira. By introducing MPI to improve data movement, combining multiple phases of application execution in a single batch job, and utilizing compute-node RAM disks to store temporary data, Alpgen scaled to over a million threads.

Acknowledgments

This work is supported by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, under contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. An award of computer time was provided by the DOE Office of Advanced Scientific Computing Research (ASCR) Leadership Computing Challenge (ALCC) program.

References

[1] C. Eck, J. Knobloch, L. Robertson, I. Bird, K. Bos, N. Brook, D. Düllmann, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, J. Harvey, A. Heiss, F. Hemmer, S. Jarp, R. Jones, D. Kelsey, M. Lamanna, H. Marten, P. Mato-Vila, F. Ould-Saada, B. Panzer-Steindel, L. Perini, Y. Schutz, U. Schwickerath, J. Shiers, T. Wenaus, LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005), Technical Design Report LCG, CERN, Geneva, 2005.
URL https://cds.cern.ch/record/840543.

[2] M.L. Mangano, F. Piccinini, A.D. Polosa, M. Moretti, R. Pittau, J. High Energy Phys. 2003 (07) (2003) 001. URL http://stacks.iop.org/1126-6708/2003/i=07/a=001.

[3] K. Olive, P.D. Group, Chin. Phys. C 38 (9) (2014) 090001.
URL http://stacks.iop.org/1674-1137/38/i=9/a=090001.

[4] MPI: A Message-Passing Interface Standard: Version 3.0. URL http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

[5] IBM System Blue Gene Solution Blue Gene/Q Application Development.
URL http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf.