Simulation of LHC events on a million threads

J. Taylor Childers
of Argonne National Laboratory
with Tom LeCompte (Argonne)
Tom Uram (Argonne)
Doug Benjamin (DukeU)
Why use non-Grid resources?

- The Grid is an amazing resource that will continue to be the primary resource for LHC analysis.
- As the computing needs of LHC experiments continue to increase, and funding sources remain flat, new resources must be identified.
- Grid grew up with serial fast CPUs however consumer machine architecture has changed from ever increasing clock speeds to large core counts, meaning our code will need to evolve.
There are many High Performance Computers (HPC) available via DOE LCFs.
Computing time is allocated via a competitive application process, but is otherwise offered at no upfront cost.
The upfront cost is in learning to effectively use these machines.
However, this cost is well spent considering future desktops look more like highly parallel machines.

- 48k Nodes: 64 threads, 16GB each
- 1.6 GHz BlueGeneQ PowerPC
- 3.1M parallel threads possible
- 6.8B core-hours/year (Grid ~2.5B/year)
Event Generators as a Pathfinder

Why Event Generators?
• Porting the massive simulation and analysis code stacks of LHC experiments to HPCs will require many people-hours.
• Event Generators come from outside these collaborations and remain largely independent inside the frameworks.
• When we began, success was not guaranteed so wanted to invest minimal effort in code porting while keeping the possibility for productive output.

Begin with Alpgen and Sherpa which are used to produce some of the largest Monte Carlo samples, specifically W/Z+jets, W+heavy flavor+jets.

High multiplicity processes are also optimal for HPCs because they have a high CPU to File I/O ratio.
• For instance, W/Z+5jets writes 1 event for every 100,000 generated compared to 75 of every 100 events for W/Z+0jets.
• Applications for CPU-time cannot be based on needing more compute time, it must be based on how the unique capabilities of the machine will enable new research.
• W/Z+4,5,6jets is very costly on the Grid and routinely fails because the jobs produce 0 events. HPCs offer a unique ability to produce the large datasets required.
• W/Z+6jets are not produced on the Grid.
Serial Event Generation Workflow on the Grid

Weighted event generation scales with sample size and process cross-section. Efficiency for Z+5 jets shown in parenthesis.

- Warmup
- Weighted Event Generation (Eff. 0.3%)
- Pythia, Tauola, Photos, & Filters (5%)
- Unweighting (1%)
- Alpgen
Serial Event Generation Workflow on the Grid

Weighted event generation scales with sample size and process cross-section. Efficiency for Z+5 jets shown in parenthesis.

Weighted Event Generation (Eff. 0.3%)

Pythia, Tauola, Photos, & Filters (5%)

Unweighting (1%)

Inherently serial

Fast

Warmup

Alpgen
Converting to HPC Workflow

Use the event-wise independence to chop up a single weighted event generation step into many

Weighted Event Generation

Pythia, Tauola, Photos, & Filter

Warmup

Unweighting

Alpgen
Early on the Warmup, Unweighting, and other filtering was done on a condor cluster running SLC6. Only weighted event generation ran on Mira.
Begin with simple MPI

- Began with Alpgen a straightforward, fortran-based LO event generator.
- Added MPI_INIT, determined rank number, then immediately called MPI_FINALIZE.
- Used rank number to update random number seeds.

```fortran
  c beginning of MPI block
  c retrieve MPI rank information
      call MPI_INIT(ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
      mpirank = rank
      call MPI_FINALIZE(ierr)
  c end of MPI block
```

Throttled by File IO

For Every Rank:

- At the start:
  - Read Input Configuration
  - Read PDF datafile
  - Read Integration Grid
  - Open output files

- 1 IO Node per 128 worker nodes (~1.25GBps disk read/write speed)
- 4096 ranks reading/writing put us in the ~1GBps range.
- Creation of all the folders before running, concatenation of all the results, and removing all the folders were up to 50% of total job run time.
- Doubling the number of ranks, doubled the process time.

- At the end:
  - Write Run Parameter Files
  - Write Histograms
  - Write Monitoring Files
Incremental Changes

- We switched to a single folder.
- Writing 100k files to the same folder only provided a marginal improvement in the post-job processing.
- Unknown restriction was only one IO-Node can access the same directory at a time.
- But we are now routinely running 4096 nodes (131,072 ranks per job)
Set Rank 0 as the only rank to read from disk.

**MPI_BCAST** used by Rank 0 to send input file data to all other ranks via the high speed interconnects.

Resulted in a 20% improvement in performance.

Still writing many files to disk and time to aggregate these events into a single file took as long as the jobs ran on Mira.

Began pushing to 8192 node jobs (262,144 ranks per job)
Incremental Changes

- Created a script that runs the weighted event generation and unweighting in the same rank.
- Files were still written to disk, but the final data size had been reduced (unweighting rejects 99% of events in W/Z+5jets), speeding up the post-run concatenation.
- A big problem: Our run times were wildly unpredictable, even for identical jobs.
  - Wanted to be able to predict job times for filling holes in the machine queue.
- Some investigation into this by ALCF experts revealed the problem was related to how Fortran was buffering the data Alpgen was writing to disk.
  - Data written over the course of a job’s run time was not over IO bandwidth limits, however Fortran was buffering ALL output until the program completed. This caused large to-disk data rates at the end of the job.
  - Added flush-statement every N events.
  - A similar effect was seen with the STDOUT so limited it to rank-0 only.
- Produced an 80% improvement in performance
- Gets us to the 16,384 node scale (524,288)
Incremental Changes

- Created a script that runs the weighted event generation and unweighting in the same rank.
- Files were still written to disk, but the final data size had been reduced (unweighting rejects 99% of events in W/Z+5jets), speeding up the post-run concatenation.
- A big problem: Our run times were wildly unpredictable, even for identical jobs.
  - Wanted to be able to predict job times for filling holes in the machine queue.
  - Some investigation into this by ALCF experts revealed the problem was related to how Fortran was buffering the data Alpgen was writing to disk.
  - Data written over the course of a job’s run time was not over IO bandwidth limits, however Fortran was buffering ALL output until the program completed. This caused large to-disk data rates at the end of the job.
  - Added flush-statement every N events.
  - A similar effect was seen with the STDOUT so limited it to rank-0 only.
  - Produced an 80% improvement in performance.
  - Gets us to the 16,384 node scale (524,288 nodes).
Final big step was to setup a RAM-disk on each node where intermediate data could be stored locally instead of on disk.

Each rank runs the

- **Weighted Event Generation,**
- **Unweighting,** and
- **Aggregation**

which is a simple C++ program that reads the output file from the unweighting step and aggregates the data via MPI (high speed interconnects) into a single file.

This resulted in an 80% performance improvement over writing to disk between the **weighted event generation** and **unweighting** and having each rank writing the unweighted output to disk.

This moved us to the full Mira machine: 49,152 nodes x 32 ranks per node = 1,572,864 ranks of Alpgen running.
Incremental Changes

- Final big step was to setup a RAM-disk on each node where intermediate data could be stored locally instead of on disk.

- Each rank runs the **Weighted Event Generation**, **Unweighting**, and **Aggregation** which is a simple C++ program that reads the output file from the unweighting step and aggregates the data via MPI (high speed interconnects) into a single file.

- This resulted in an 80% performance improvement over writing to disk between the weighted event generation and unweighting and having each rank writing the unweighted output to disk.

- This moved us to the full Mira machine: 49,152 nodes x 32 ranks per node = 1,572,864 ranks of Alpgen running.
Production Progress

- Used 36.3M of 50M CPU-hours
- Produced over 6B unweighted Alpgen events.
- Still have another ~9B Alpgen to produce for Run II analyses (30M CPU-hours of work remain)
- Have another allocation proposal under review for July 2015 - June 2016 to continue production with Sherpa.
Complementary Work

- Developed a ARGO-Balsam (see Tom Uram’s talk in Track 4) Workflow/Scheduler interface in an effort to tie Mira into PanDA in order to run LHC event generation jobs.
Complementary Work

- Developed a ARGO-Balsam (see Tom Uram’s talk in Track 4) Workflow/Scheduler interface in an effort to tie Mira into PanDA in order to run LHC event generation jobs.
- Working with Sherpa Event Generator developers to make it more efficient at these large scales.
  - Sherpa has the capability to do the warmup step in parallel as well.

![Mira Activity](image)

12k node Sherpa job
Complementary Work

- Developed a ARGO-Balsam (see Tom Uram’s talk in Track 4) Workflow/Scheduler interface in an effort to tie Mira into PanDA in order to run LHC event generation jobs.

- Working with Sherpa Event Generator developers to make it more efficient at these large scales.
  - Sherpa has the capability to do the warmup step in parallel as well.

- Working with Geant4 and ROOT developers to ensure support for PowerPC is built in.
  - Next Generation Machines (pre-Exa-scale) have been announced and they include PowerPC machines

- Next Generation Machines will use Co-processors (Xeon Phi, GPUs) and work is ongoing to learn how to prepare for these machines ensuring HEP is not excluded from such resources.
Summary

- Alpgen is at 1.5 Million Threads!
- Sherpa at 98k threads and rising...
- Pythia8 at 32k threads.
- MadGraph_MC@NLO will be a future target.

### Mira Activity

![Mira Activity Grid](image-url)
BACKUP SLIDES
Computing holding back Physics Results

- Here is a limit set by ATLAS, however, it is missing limits at higher gluino mass.
- Waited 2 months for Grid jobs to get the plot as it is, didn’t want to hold up the paper another 2 months waiting for remaining points.

http://arxiv.org/abs/1208.0949