Challenges in Scaling NLO Generators to Leadership Computers

J. Taylor Childers (Argonne)
with Tom Uram (Argonne)
Stefan Hoeche (SLAC)
Tom LeCompte (Argonne)
Doug Benjamin (Duke)
Motivation

- As we have seen in many talks at CHEP, LHC computing needs are expected to grow by more than a factor of 60 through the HL-LHC.
- Any cpu cycle offloaded to an HPC frees up a cycle on the Grid.
- NLO generators, like Sherpa & MadGraph5_aMC@NLO, involve computationally intensive integrations.
  - In Sherpa, W+4 jets with 1% precision requires a 16-core Intel Xeon 6.5days to compute.
- Increased multiplicity and going to NNLO will continue to increase the compute time of these calculations.
- Scaling codes on leadership machines reduces the time to get results and these optimizations typically result in better efficiency on all systems.
High Performance Computers

- **MIRA**
  - 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)

- **Edison**
  - 5,200 nodes: 24 cores per node
  - 2x2.4GHz Intel Ivy Bridge
  - 24 GB DDR3 1866 MHz
  - 1.1B core-hours/year (Grid ~2.5B/year)

- **Titan**
  - 18,688 nodes: 16 CPU cores, 1 NVIDIA Kepler GPU
  - 2.2GHz AMD Opteron with 32GB
  - 6GB RAM on GPU
  - 2.6B CPU-core-hours/year
High Performance Computers

- 48k Nodes: 64 threads, 16GB each
- 1.6 GHz BlueGene
- 3.1M parallel threads
- 6.8B core-hours/year (Grid ~2.5B/year)

- 5,200 nodes: 24 cores per node
- 2x2.4GHz Intel Ivy Bridge
- 24 GB DDR3 1866 MHz
- 1.1B core-hours/year (Grid ~2.5B/year)

- 18,688 nodes: 16 CPU cores, 1 NVIDIA Kepler GPU
- 2.2GHz AMD Opteron with 32GB
- 6GB RAM on GPU
- 2.6B CPU-core-hours/year

Currently: 10B core-hours per year on HPCs
LHC Grid Usage ~2.5B per year
High Performance Computers

Coming online in the next few months

- 3240 nodes: 64 cores x 4 HW threads
- 256 threads/node
- Intel Xeon Phi (Knights Landing)
- 16GB on-chip memory
- 192 GB DDR4 2400 MHz
- 1.8B core-hours/year (Grid ~2.5B/year)

- 9,304 nodes: 68 cores x 4 HW threads
- 272 threads/node
- Intel Xeon Phi (Knights Landing)
- 16GB on-chip memory
- 96 GB DDR4 2133 MHz
- 5.5B core-hours/year (Grid ~2.5B/year)
Where We Started: Alpgen, an LO Generator

- Alpgen is an LO parton generator written in Fortran
- Every process gets a binary
- Most configurable settings are values of physics constants and do not affect program flow
- Ran serial Alpgen in parallel with minimal MPI additions for random number seeds and file I/O
- Used RAM-disks for intermediate data
- Allowed to fill Mira (6th fastest on the Top500) with the largest generation job ever. 1.5M parallel processes
- Code in Github
Sherpa, a Next-to-Leading-Order Generator

- Sherpa is an NLO event generation framework that supports many pluggable algorithms (both LO & NLO)
- Sherpa is a much more complex code AND framework than Alpgen.
- Since it supports multiple plugins and integrators it has much more program flow, meaning the CPU spends much of its time deciding which code is going to be run next.
- Unfortunately, increased flexibility causes decreased performance
- But there’s hope, Sherpa supports MPI and supported pthreads in the past.
Sherpa Operational Model: Parallel Integration

Initialization & Matrix Element Loading
- ~1 min on Xeon
- ~8 min on Blue Gene/Q

All Ranks doing the exact same thing

Phase Space Integration

n points per rank

All Ranks calculating different phase space points

(1) Exchange phase space data
(2) Update all rank’s grid
(3) Check xs precision

Continue until xs precision goal is reached
Sherpa Operational Model: Parallel Event Generation

**Initialization & Matrix Element Loading**
Read grid data from Integration

**Event Generation**

- **n events per rank**
- **All Ranks doing the exact same thing**
- **All Ranks generating different events**
- **MPI_Allreduce**

1. Exchange phase space data
2. Update all rank’s grid
3. Check xs precision

Continue until requested number of events are generated
Sherpa Operational Model: Parallel Integration

We focus on the Integration which is the compute intensive part

Initialization & Matrix Element Loading
- ~1 min on Xeon
- ~8 min on Blue Gene/Q

All Ranks doing the exact same thing

Phase Space Integration

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Space points (1) Exchange phase space data
(2) update all rank’s grid
(3) check xs precision

n points per rank

All Ranks calculating different phase space points

Continue until xs precision goal is reached
Sherpa Operational Model: Parallel Integration

We focus on the Integration which is the compute intensive part

Initialization & Matrix Element Loading
~1 min on Xeon
~8 min on Blue Gene/Q

All Ranks doing the exact same thing

Phase Space Integration

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.

We focus on the Integration which is the compute intensive part

When we started, this initialization time scaled with number of ranks. On Mira a 512 node job with 8 ranks per node it took 50 minutes! Every rank is doing the exact same thing, very inefficient.

Today this only takes 8 minutes, which is much more tolerable. Thanks to Stefan’s many improvements to make this happen.
Sherpa Operational Model: Parallel Integration

We focus on the Integration which is the compute intensive part

What do we expect to happen when we double the number of MPI ranks?

Initialization & Matrix Element Loading
- ~1 min on Xeon
- ~8 min on Blue Gene/Q

All Ranks doing the exact same thing

Phase Space Integration

n points per rank

All Ranks calculating different phase space points

(1) Exchange phase space data
(2) update all rank's grid
(3) check xs precision

MPI_Allreduce

Continue until xs precision goal is reached

n points per rank

All Ranks calculating different phase space points

(1) Exchange phase space data
(2) update all rank's grid
(3) check xs precision

MPI_Allreduce

...
Sherpa Operational Model: Parallel Integration

We focus on the Integration which is the compute intensive part

What do we expect to happen when we double the number of MPI ranks?

Initialization & Matrix Element Loading
-1 min on Xeon
-8 min on Blue Gene/Q

Phase Space Integration

n points per rank

All Ranks doing the exact same thing

Now Constant

All Ranks calculating different phase space points

(1) Exchange phase space data
(2) update all rank’s grid
(3) check xs precision

MPI_Allreduce

Continue until xs precision goal is reached
Sherpa Operational Model: Parallel Integration

We focus on the Integration which is the compute intensive part

What do we expect to happen when we double the number of MPI ranks?

Phase Space Integration

Initialization & Matrix Element Loading
~1 min on Xeon
~8 min on Blue Gene/Q

Now Constant

All Ranks doing the exact same thing

All Ranks calculating different phase space points

~1 min on Xeon
~8 min on Blue Gene/Q

n points per rank

This process should decrease by half approximately as the total number of phase space points should not change.

(1) Exchange phase space data
(2) Update all rank’s grid
(3) Check xs precision

All Ranks calculating different phase space points

All Ranks calculating different phase space points

All Ranks calculating different phase space points

Continue until xs precision goal is reached

n points per rank

n points per rank

n points per rank
Sherpa Performance Testing

- **Ratio of Runtime of $\nu+4$jets** on Intel Xeon Phi (KNL) and Mira (BGQ) exhibits how Sherpa scales with the job shape
  - job shape: ‘nodes’ x ‘MPI processes per node’
  - normalized to the first shape on the left
- **In the ideal case, there would be linear scaling.** The integration has a fixed amount of work that needs to be done, therefore doubling the number of parallel processes should halve the time needed to calculate the integration.
- On the KNL Sherpa quickly leaves the linear scaling curve and flattens by 48 processes per node
- On Mira there is no evidence of scaling.
- Let’s look at why this may be
**Sherpa on Mira**

- What’s going on to prohibit the scaling?
- Smallest jobs are 512 nodes with 8 processes per node (4k processes)
- Illustration:
  - Each process is calculating 5 phase space points
  - Let’s say most phase space points take 1 minute to calculate, but every now and then a complex point takes 10 minutes.
  - Most processes finish within some short time period, but one takes an extra long time.
  - If each process is calculating thousands of points, this averages out, but because Mira is so large we only need each processes to calculate a few points per iteration.
  - This is of course process dependent as W+5 or 6 jets will require more phase space points.
Sherpa on Mira

- Disclaimer: Profiled Sherpa 2.2.0 before Stefan’s many speedups, need to do it again, but profilers are fussy and I will need to spend a week getting it working again.
- Profiled using Score-P showing the fraction of run time spent in each function.
- This was taken on the ALCF Cooley cluster
  - Two 2.4 GHz Intel Haswell E5-2620 v3 (12 cores total)
  - This is for Rank 0, which is the manager of other ranks in Sherpa and always has the smallest time spent in MPI_Allreduce
  - 7% of the time for this rank 0 was spent in MPI_Allreduce, most likely waiting on other ranks to reach the same MPI call.
  - 12% of the time is spent converting things to strings
  - The values quickly fall to sub-percent levels exhibiting the challenge of Sherpa, i.e. no one performance blocker.
Sherpa on Mira

- Disclaimer: Profiled Sherpa 2.2.0 before Stefan’s many speedups, need to do it again, but profilers are fussy and I will need to spend a week getting it working again.
- Profiled using Score-P showing the fraction of run time spent in each function.
- This was taken on the ALCF Cooley cluster.
  - Two 2.4 GHz Intel Haswell E5-2620 v3 (12 cores total).
- This is for Rank 0, which is the manager of other ranks in Sherpa and always has the smallest time spent in MPI_Allreduce.
- 7% of the time for this rank 0 was spent in MPI_Allreduce, most likely waiting on other ranks to reach the same MPI call.
- 12% of the time is spent converting things to strings.
- The values quickly fall to sub-percent levels exhibiting the challenge of Sherpa, i.e. no one performance blocker.

The point? Sherpa has no single computation where time is focused, Mira is inefficient at program flow so we see this in our scaling measurements.
Some of the Sherpa Developments

- Stefan Hoche (SLAC) has been very supportive of our efforts to run Sherpa at large scales, providing many updates and patches to improve performance, remove old inefficiencies.
- Many Framework improvements
- Reduced ‘fstat’ calls and file system crawling which is slow with thousands of processes
- Reducing number of shared libraries to load
- Removing system specific code
- Reductions in memory consumption
- Instead of each process calculating N phase space points between MPI_Allreduce, a time period can be specified, allowing process to accomplish as much as they can between communication steps.
Code improvements enable scaling on KNL

New results from two days ago… next test on Mira to see if we see similar improvements
Future Plans

- Continue testing on KNL going to multiple nodes
- Update scaling tests on Mira using this time period control of the length of the phase space point calculations
- Update profiling to see how these changes have changed the distribution of work and communication
- From this decide how to move forward with Sherpa

- In parallel working with Olivier Mattelaer from MadGraph5_aMC@NLO to parallelize MadEvent binaries.
  - This involves having the MG5 framework generate MPI wrappers for these FORTRAN binaries
  - Then taking advantage of the cluster mode to run many processes in parallel
Future Plans

- Continue testing on KNL going to multiple nodes
- Update scaling tests on Mira using this time period control of the length of the phase space point calculations
- Update profiling to see how these changes have changed the distribution of work and communication
- From this decide how to move forward with Sherpa

- In parallel working with Olivier Mattelaer from MadGraph5_aMC@NLO to parallelize MadEvent binaries.
  - This involves having the MG5 framework generate MPI wrappers for these FORTRAN binaries
  - Then taking advantage of the cluster mode to run many processes in parallel
Baseline Sherpa Test Job

- Process: $W \rightarrow e\nu + \text{up to 4-jets}$
- Most testing has been focused on scaling integration which at NLO/NNLO is the computationally intensive piece
- Start with 50% integration precision to get data for a fast job, with this precision integration takes
  - $\sim 3\text{hrs}$ on 16 Xeon cores
  - $\sim 1.5\text{hrs}$ on 512 nodes with 8 nodes per core on Mira.