HEP-CCE: Event Generators Report

J. Taylor Childers (ANL), Stefan Höche (FNAL)

Review of FY20-FY22

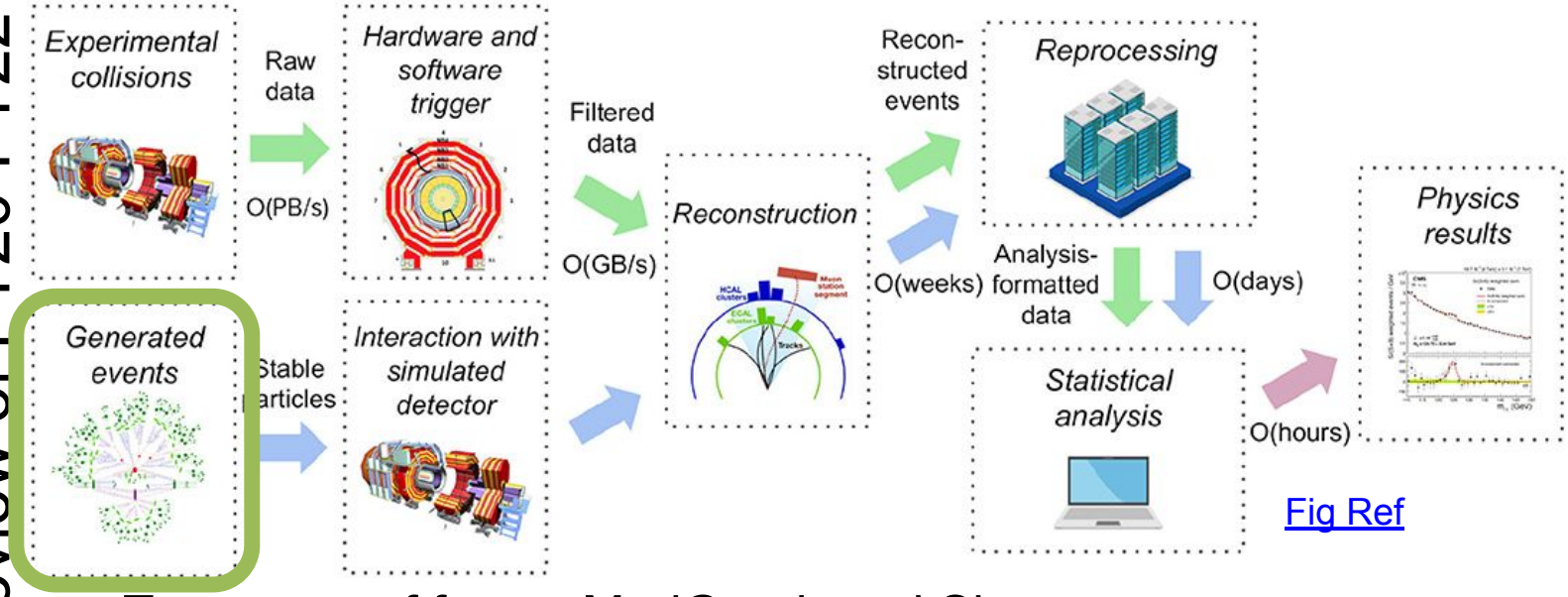
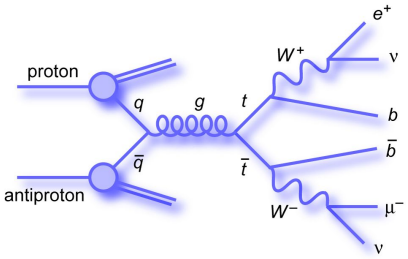


Fig Ref

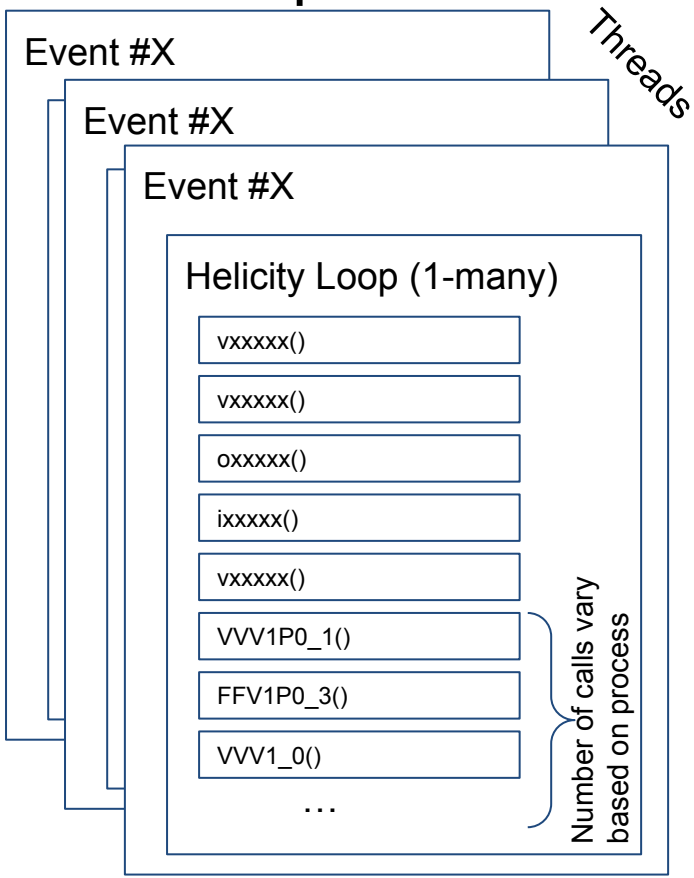
- Two areas of focus: MadGraph and Sherpa
- Both at Leading Order (LO) in perturbative QCD event generators that simulate the particle collisions observed by LHC detectors
- Madgraph included extensive porting to multiple frameworks allowing comparisons in portability
- Sherpa focused on algorithm improvement on GPUs with a focus on using Kokkos for portability

MadGraph: FY20 - FY22

- Collaboration with MG-team, ANL, & CERN:
  - UC-Louvain (FR): Olivier Mattelaer (MG team member)
  - CERN: Andrea Valassi, Stefan Roiser, David Smith, Laurence Field (all CERN-IT)
  - ANL: Walter Hopkins (HEP), Nathan Nichols (ALCF), Taylor Childers (ALCF)
- CERN team leads CUDA development
- ANL team leads Kokkos and Sycl development
- Historically MadGraph uses a python user interface, which generates, compiles, and runs FORTRAN code to generate events.
- Process began with a CUDA implementation of the Matrix Element (ME) calculations which are used to calculate the interaction cross-section for a process and are the biggest computational piece.
- Kokkos port followed, then Sycl, then Alpaka, and the CUDA port was extended to also offer OpenMP

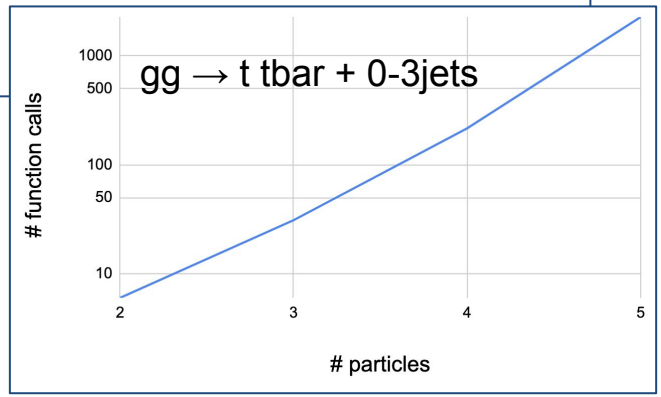


MadGraph: FY20 - FY22




```
// Wavefunction(s) for diagram number 7
FFV1_2( w_sv[3], w_sv[0], cxmake( cIPC[2], cIPC[3] ), cIPD[0], cIPD[1], w_sv[5] );
FFV1_1( w_sv[2], w_sv[1], cxmake( cIPC[2], cIPC[3] ), cIPD[0], cIPD[1], w_sv[11] );

// Amplitude(s) for diagram number 7
FFV1_0( w_sv[5], w_sv[11], w_sv[4], cxmake( cIPC[2], cIPC[3] ), &amp_sv[0] );
jamp_sv[0] -= amp_sv[0];
jamp_sv[1] -= amp_sv[0];
jamp_sv[2] += +amp_sv[0];
jamp_sv[3] -= amp_sv[0];
jamp_sv[4] += +amp_sv[0];
jamp_sv[5] -= amp_sv[0];
```



MadGraph: FY20 - FY22

Event #X

Event #X

Event #X

Helicity

vxxxxx()

vxxxxx()

oxxxxx()

ixxxxx()

vxxxxx()

VVV1P0_

FFV1P0_

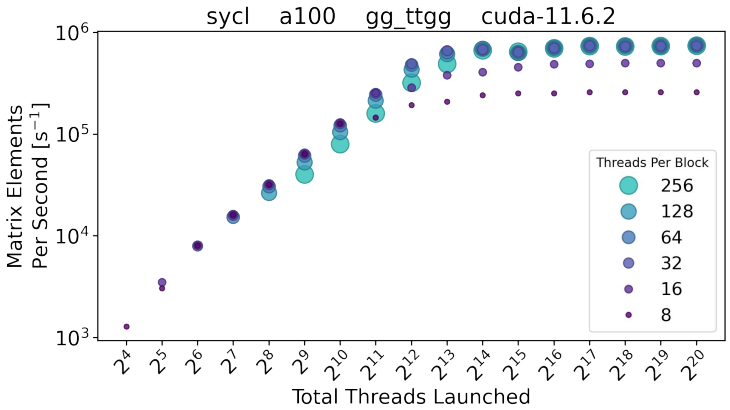
VVV1_0()

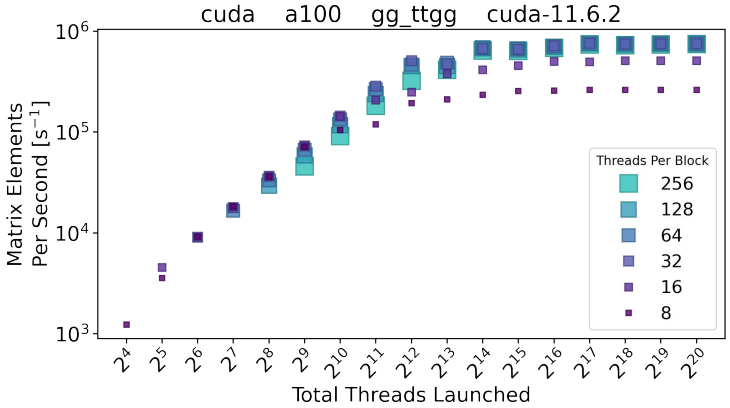
```
// Compute the output wavefunction 'V1[6]' from the input wavefunctions V2[6], V3[6]

KOKKOS_INLINE_FUNCTION
void VVV1P0_1( const cxtype V2[],
               const cxtype V3[],
               const cxtype COUP,
               const fptype M1,
               const fptype W1,
               cxtype V1[] )
{
  const cxtype cI = cxmake( 0., 1. );
  const fptype_sv P2[4] = { +cxreal( V2[0] ), +cxreal( V2[1] ), +cximag( V2[1] ), +cximag( V2[0] ) };
  const fptype_sv P3[4] = { +cxreal( V3[0] ), +cxreal( V3[1] ), +cximag( V3[1] ), +cximag( V3[0] ) };
  V1[0] = + V2[0] + V3[0];
  V1[1] = + V2[1] + V3[1];
  const fptype_sv P1[4] = { -cxreal( V1[0] ), -cxreal( V1[1] ), -cximag( V1[1] ), -cximag( V1[0] ) };
  const cxtype_sv TMP0 = ( V3[2] * P1[0] - V3[3] * P1[1] - V3[4] * P1[2] - V3[5] * P1[3] );
  const cxtype_sv TMP2 = ( V3[2] * P2[0] - V3[3] * P2[1] - V3[4] * P2[2] - V3[5] * P2[3] );
  const cxtype_sv TMP4 = ( P1[0] * V2[2] - P1[1] * V2[3] - P1[2] * V2[4] - P1[3] * V2[5] );
  const cxtype_sv TMP5 = ( V2[2] * P3[0] - V2[3] * P3[1] - V2[4] * P3[2] - V2[5] * P3[3] );
  const cxtype_sv TMP6 = ( V3[2] * V2[2] - V3[3] * V2[3] - V3[4] * V2[4] - V3[5] * V2[5] );
  const cxtype_sv denom = COUP / ( (P1[0] * P1[0] ) - ( P1[1] * P1[1] ) - ( P1[2] * P1[2] ) - ( P1[3] * P1[3] ) - M1 * ( M1 - cI * W1 ) );
  V1[2] = denom * ( TMP6 * ( - cI * ( P2[0] ) + cI * ( P3[0] ) ) + ( V2[2] * ( - cI * ( TMP0 ) + cI * ( TMP2 ) ) + V3[2] * ( +cI * ( TMP4 )- cI * ( TMP5 ) ) )
  V1[3] = denom * ( TMP6 * ( - cI * ( P2[1] ) + cI * ( P3[1] ) ) + ( V2[3] * ( - cI * ( TMP0 ) + cI * ( TMP2 ) ) + V3[3] * ( +cI * ( TMP4 )- cI * ( TMP5 ) ) )
  V1[4] = denom * ( TMP6 * ( - cI * ( P2[2] ) + cI * ( P3[2] ) ) + ( V2[4] * ( - cI * ( TMP0 ) + cI * ( TMP2 ) ) + V3[4] * ( +cI * ( TMP4 )- cI * ( TMP5 ) ) )
  V1[5] = denom * ( TMP6 * ( - cI * ( P2[3] ) + cI * ( P3[3] ) ) + ( V2[5] * ( - cI * ( TMP0 ) + cI * ( TMP2 ) ) + V3[5] * ( +cI * ( TMP4 )- cI * ( TMP5 ) ) )
  return;
}
```

MadGraph: FY20 - FY22

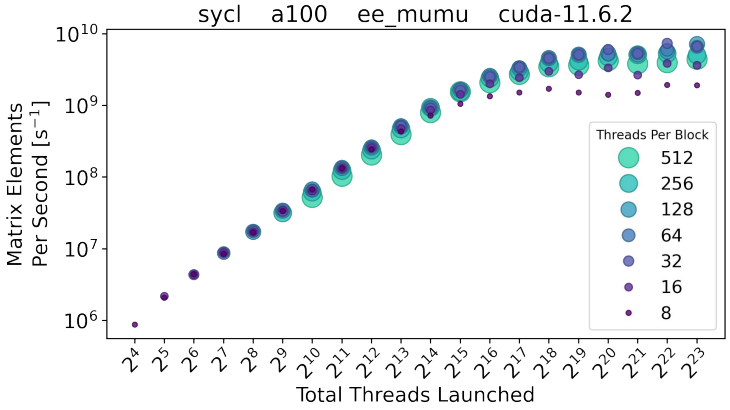
- Repo Link
- Our figure of merit is the rate of calculating the ME of one interaction. (ME/s)
- We vary the total number of threads run, find the maximum for each framework on each system, then compare.
- We use ee→μμ, and gg→tt+0,1,2,3jets as test cases to measure effects of increasing computational complexity
- Only ee→μμ (easy) and gg→tt+2jets (hard) will be presented due to time.

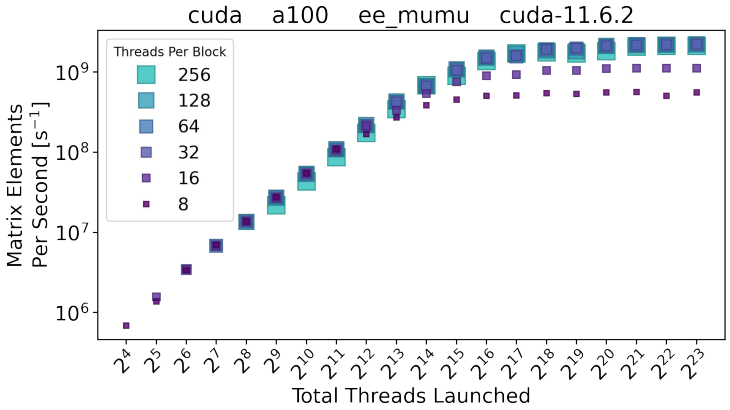




MadGraph: FY20 - FY22

- Repo Link
- Our figure of merit is the rate of calculating the ME of one interaction. (ME/s)
- We vary the total number of threads run, find the maximum for each framework on each system, then compare.
- We use ee→μμ, and gg→tt+0,1,2,3jets as test cases to measure effects of increasing computational complexity
- Only ee→μμ (easy) and gg→tt+2jets (hard) will be presented due to time.

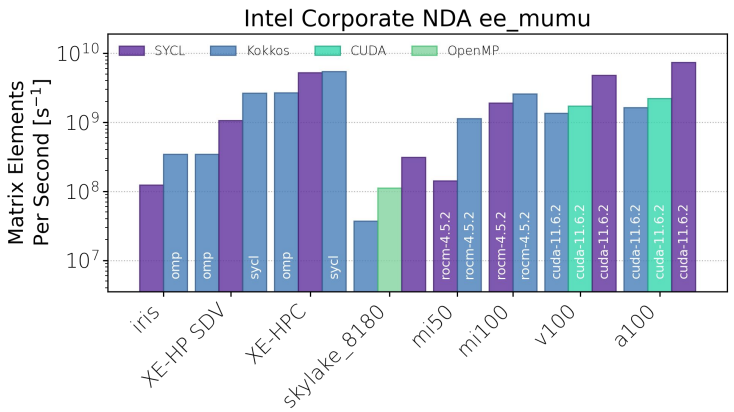


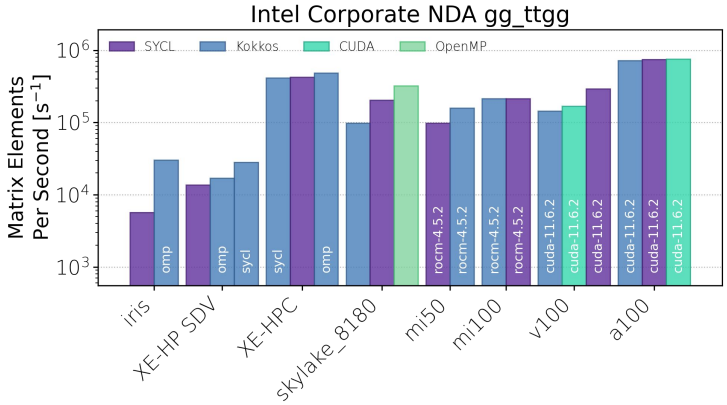


MadGraph: FY20 - FY22

ICHEP2022 SLIDES

- These plots are hot off the press.
- Here we have 3 intel GPUs, 2 AMD GPUs, and 2 NVidia GPUs compared running two processes.
- **Iris** is the gen-9 integrated GPU from Intel in many laptops.
- **Xe-HP SDV** is a software development vehicle for functional testing only.
- **Xe-HPC** is an early version of the Aurora Ponte-Vecchio GPU and software.
- We expect further performance improvements before the final Aurora silicon hits the floor in the coming months.





MadGraph: FY20 - FY22

- This work has benefitted from an ATLAS Early Science Project on Aurora which covered additional effort from myself and funded a post-doc.
- All our data came from JLSE systems.

https://www.alcf.anl.gov/science/early-science-program

https://www.jlse.anl.gov/

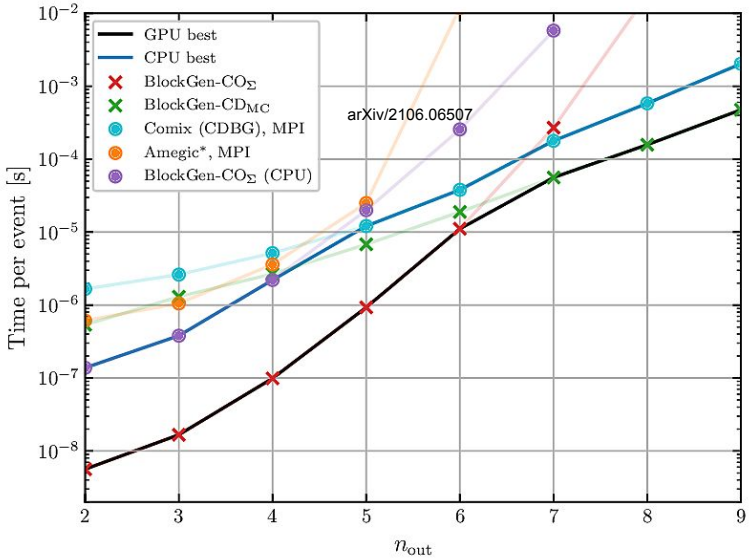
- Acknowledgement:
- We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

Sherpa: FY20 - FY22

- Collaboration between Sherpa team & ANL:
  - Uni Goettingen: Enrico Bothmann, Max Knobbe
  - Fermilab: Walter Giele, Stefan Hoeche, Joshua Isaacson
  - ANL: Taylor Childers, Rui Wang
- Goettingen / Fermilab teams lead algorithm / CUDA development
- ANL team leads Kokkos development
- Historically Sherpa is a fully integrated, general-purpose event generator, which handles all aspects of the simulation (ME, PS, hadronization, decays, …)
- Project began with an implementation from scratch of only the ME generator
  - Reduce overhead & improve portability
  - Re-evaluate choice of algorithms
- Prototyping in C++/CUDA, Kokkos as a generalization

Sherpa: FY20 - FY22

- Matrix element generators are responsible for most of the computing time spent in precision event simulation for LHC
- *Scalable algorithms & big O analysis are crucial*
- Project provided first prototype ME generator (all-gluon) for high multiplicity calculations on GPUs, using multiple alternative algorithms to achieve optimal performance
- Timing studies in comparison to existing CPU codes (MadGraph, Comix, Amegic) show factor 10+ speedup at low particle multiplicity, factor ~4 at high multiplicity. (fully loaded CPU vs GPU, for details see 2106.06507)
- https://arxiv.org/abs/2106.06507
- https://www.gitlab.com/ebothmann/blockgen-archive

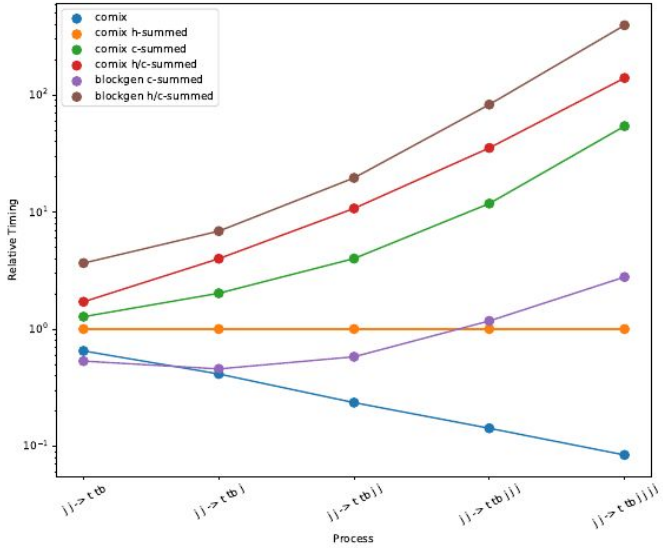


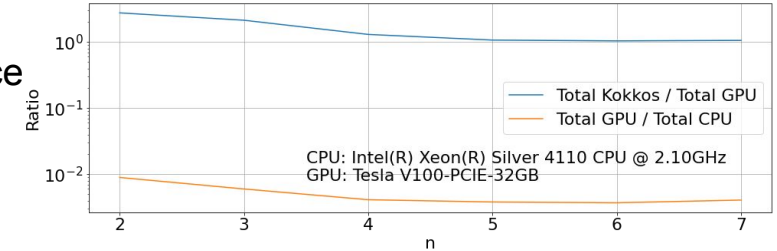
Sherpa: FY20 - FY22

- Initial code for all-gluon processes, now extended to most critical background processes at LHC:
  - W/Z+jets
  - tt+jets
  - pure jets
- Minimal basis for color summation gives significant speedup for quark amplitudes (1304.7809)

- Ported ME calculation from C++ to Kokkos
- Good performance compared to dedicated CUDA code

- Currently being interfaced to new phase-space generator and equipped with realistic parton-level cuts
- Next step: Add output to store events (HDF5)





Stop Here…
Start of FY24-26

Plans for FY24 - FY26

- Continue LO EvGen on HPC:
  - Work on scaling out on DOE resources
  - How to integrate outputs with existing experiments
    - Workflow
    - Physics validation
- Explore NLO EvGen on GPUs
  - Tree-like pieces (real/integrated subtraction terms)
  - Dedicated analytic loop matrix elements (by relevance)
  - There is an existing code for up to 4 or 5 jets at NLO that can run on CPU-only, but requires lots of Mem/core. Could run on Polaris with 16GB/core.
  - Reach out Collier group to gauge interest in GPU friendly algs?
    - Fast and stable, could be ported to Kokkos, but ideally want author involvement.
- New SciDAC-5 for EvGen development
  - Find complementary activities, e.g. HPC build out, portability
  - Using MCFM which could be utilized for Sherpa or MadGraph in the future for NLO calculations