Developments in Performance and Portability of BlockGen

J. Taylor Childers[1], Rui Wang[1]

Enrico Bothmann[2], Max Knobbe[2]

Stefan Hoeche[3], Joshua Isaacson[3]

1. Argonne National Laboratory
2. University of Göttingen
3. Fermilab

Work Done in association with the DOE High Energy Physics Center for Computational Excellence

21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research

October 24, 2022

Event Generators for LHC Simulation

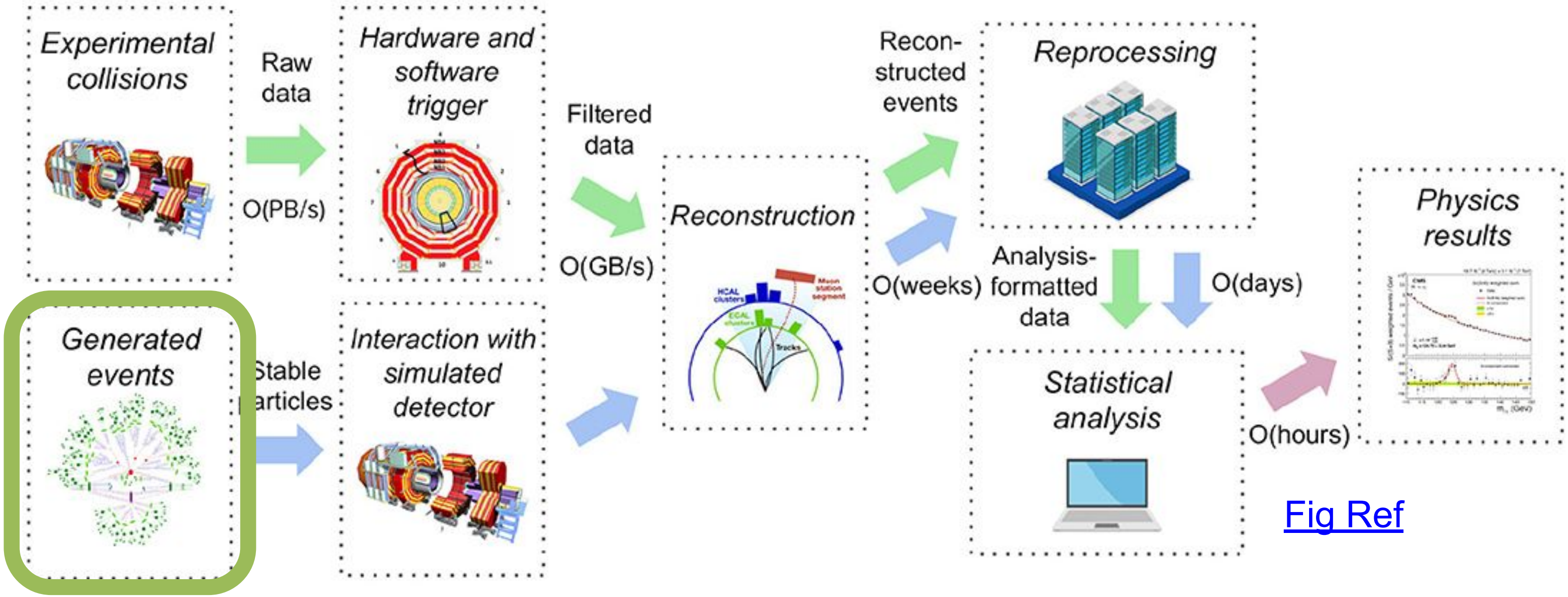
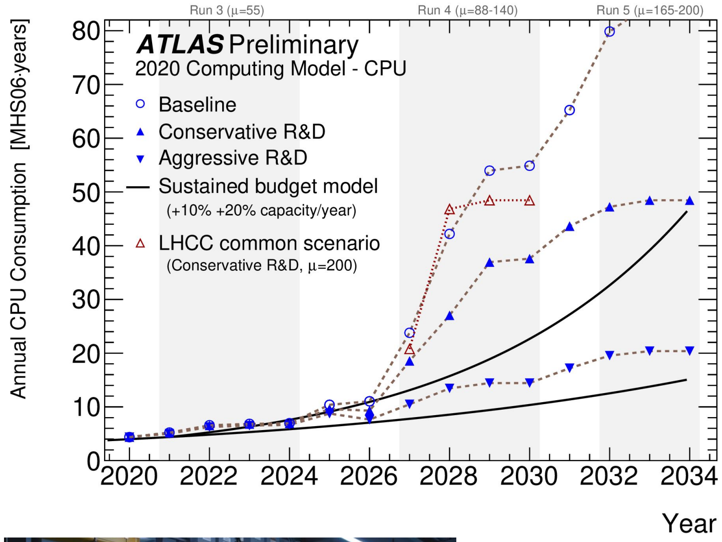


Fig Ref

- Event Generators implement the perturbative QCD calculations and use Monte Carlo methods to generate particle interactions.
- They are the first step in the simulation chain for collider experiments.

Motivation

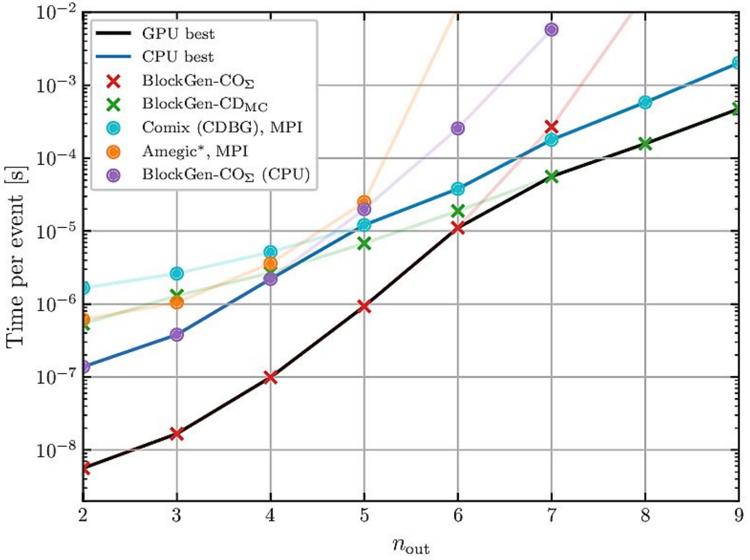
- The needs of the computing resources grows rapidly when moving into the HL-LHC era
- Could only stay within the budget under the **Aggressive R&D scenario** – [ATLAS HL-LHC Computing CDR]
  - Reduce the per event generation time
  - Utilize GPU resources besides the CPUs



- Modern architecture environment includes many different configurations of hardware, portable software helps alleviate the need for rewriting algorithms for each.
- In the future every geographic will have their own custom chips.

Brief Intro to *Blockgen* Algorithm

- Matrix Element (ME) calculation represents most of the computing time spent in precision event generation.
- We studied a new set of fast algorithms for ME calculation
  - Helicity sum, amplitudes and color sum
  - Details in Max's talk in this session
- Implemented in CUDA for early tests
- Here the improvements can be seen in one of the costliest processes for LHC event generation: **V + N$_{out}$ jets**
- Compares with existing CPU codes (Comix, Amegic)
- Shows factor ~10 speedup at low particle multiplicity, factor ~4 at high multiplicity. (with fully loaded CPU and GPU)



arXiv:2106.0650

git repo

Portability with Kokkos

- Writing code in CUDA only runs on NVidia GPUs
- Abstraction libraries like **Kokkos**, Alpaka, Sycl [Intel] provides portability for the
- Same code to be run on both CPU and GPU with Reasonable performance compared to the native language

As of March 2022

	OpenMP Offload	Kokkos	dpc++ / SYCL	HIP	CUDA	Alpaka	Python	std::par
NVidia GPU			codeplay and intel/llvm				numba	nvc++
AMD GPU		feature complete for select GPUs	via hipSYCL and intel/llvm			hip 4.0.1 / clang	numba	
Intel GPU		native and via OpenMP target offload		HIPLZ: early prototype		prototype	numba-dppy	via oneapi::dpl
CPU single-core								
CPU multi-core								nvc++ g++ & tbb
FPGA						possibly via SYCL		

Supported
Under Development
3rd Party
Not Supported

- Kokkos was used largely based on experience and evidence of achievable performance and portability
- Kokkos offers abstracted, templated memory management, and parallel kernel launching
- It is an Exascale project funded by US-DOE, not aligned to any particular industry hardware.
- Our codes are written by physics theorists, not software engineers, making readability very important.
- DOE HEP-CCE has presented on portability frameworks [ref] Poster at ACAT

Example of Kokkos Abstraction

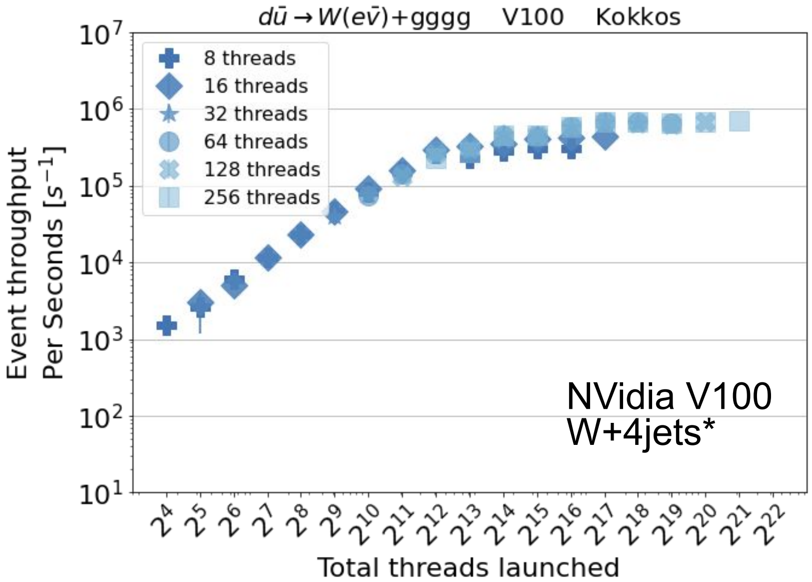
- Kokkos offers a memory management abstraction called a View class:

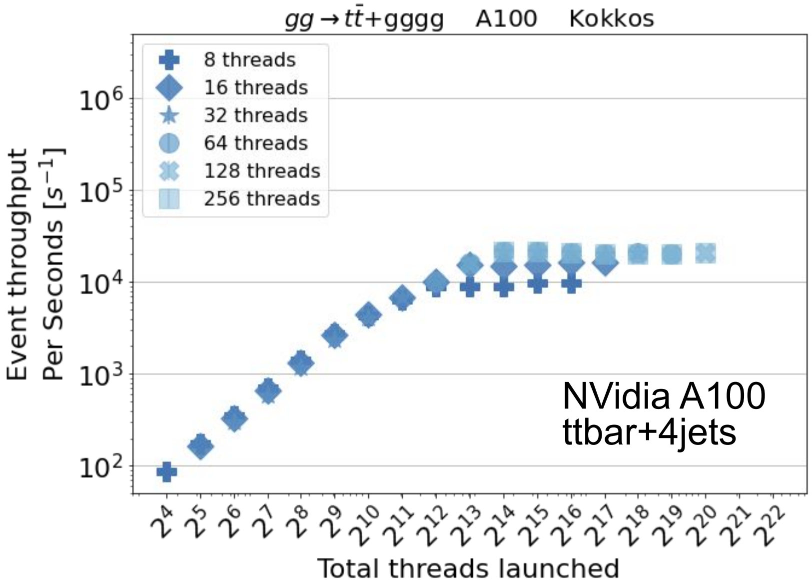
```
Kokkos::View<int*> d_array(10);  // Device-side array
auto h_array = Kokkos::create_mirror_view(d_array) // host-side array
Kokkos::deep_copy(h_array,d_array); // copy from device to host (swap for inverse)
```

- Kokkos offers methods for running parallel algorithms:

```
int team_size = 128; // like threads per block in CUDA or OMP_NUM_THREADS in OpenMP
int league_size = 1000; // like number of blocks in CUDA

using member_type = typename Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace>::member_type;
Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace> policy(league_size,team_size);

Kokkos::parallel_for("helicity_loop",policy,
  KOKKOS_LAMBDA(const member_type& team_member){
    int ievt = team_member.league_rank() * team_member.team_size() + team_member.team_rank();

    // some algorithm that runs in parallel

}); // end of parallel code
```

Event Throughput vs Parallel Threads

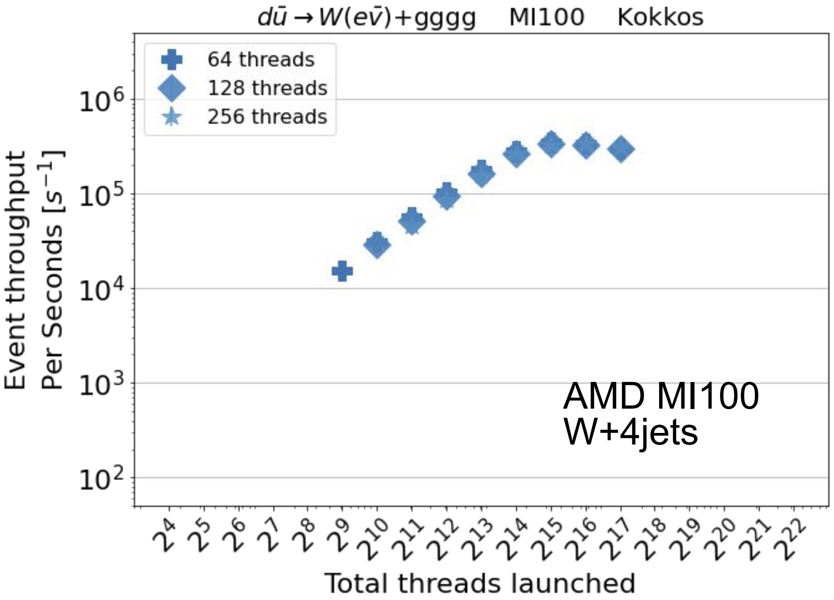


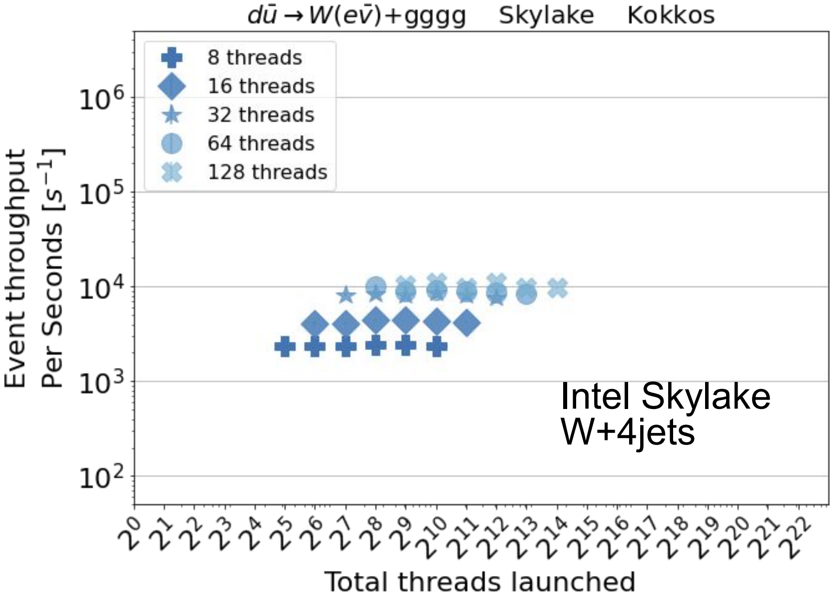


- Measure the event throughput with various combinations of *threads***blocks* on different hardware.
- The plateau is easily identified on the NVidia V100 (left) and A100 (right)
  - GPU is fully filled at this point

** Note: all jets mentioned in this talk are gluon jets*

Event Throughput vs Parallel Threads

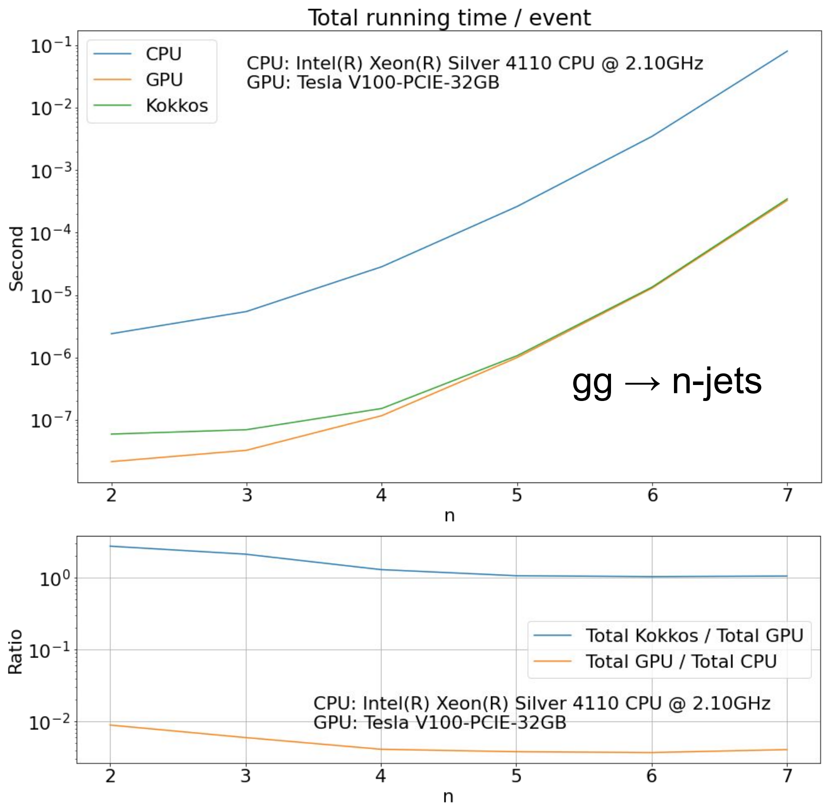




- Measure the event throughput with various combinations of *threads***blocks* on different hardware.
- The plateau is easily identified on the AMD MI100 (left) and Intel Skylake (right) as well
    - The plateau is majorly based on the No. of threads/block rather than the total threads in case of the Skylake
- The peak varies based on the process and hardware

Performance of Kokkos

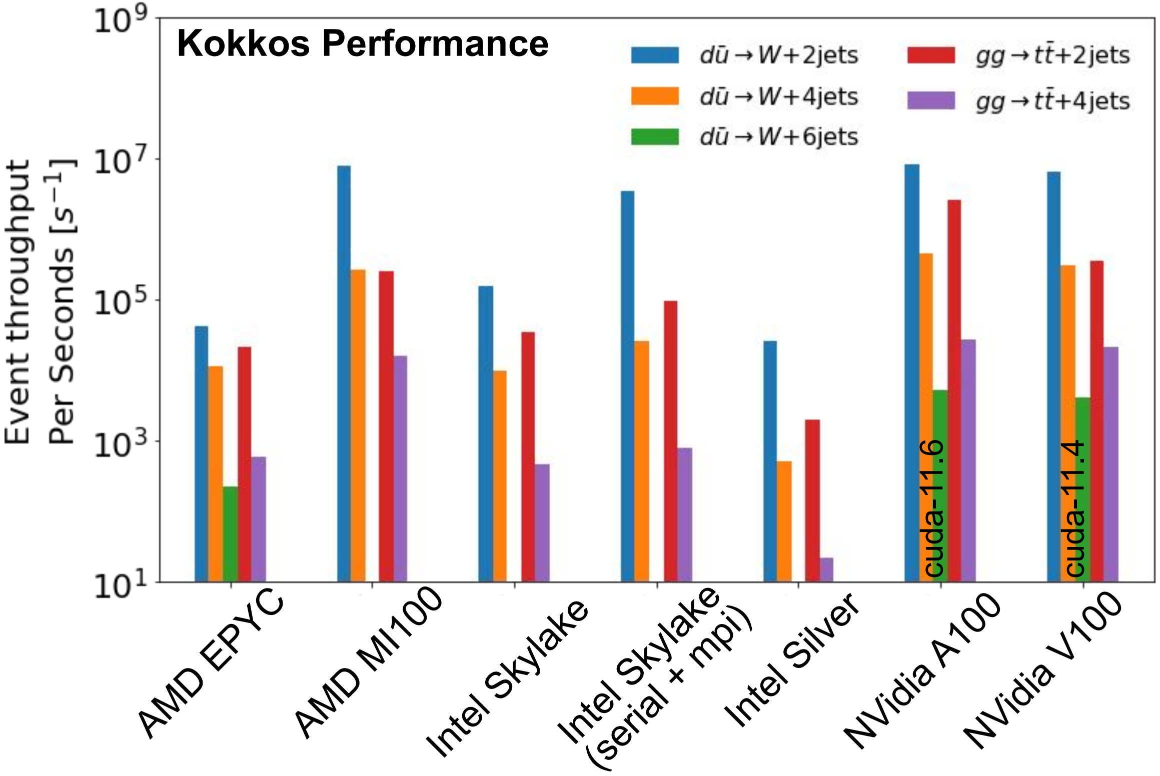
- So, does Kokkos provide equivalent performance?
- Plot shows early versions of BlockGen calculating the process: gg→njets
- Time per Event on y-axis, number of outgoing partons on x-axis
- Compare CPU with C++, GPU with CUDA, and GPU with Kokkos
- Can see the CUDA is 100x faster than the CPU for this example
- Kokkos is slightly less performant than CUDA at low multiplicity (low computational complexity), but reaches comparable performance as multiplicity increases.



Argonne NATIONAL LABORATORY

Hardware Comparison

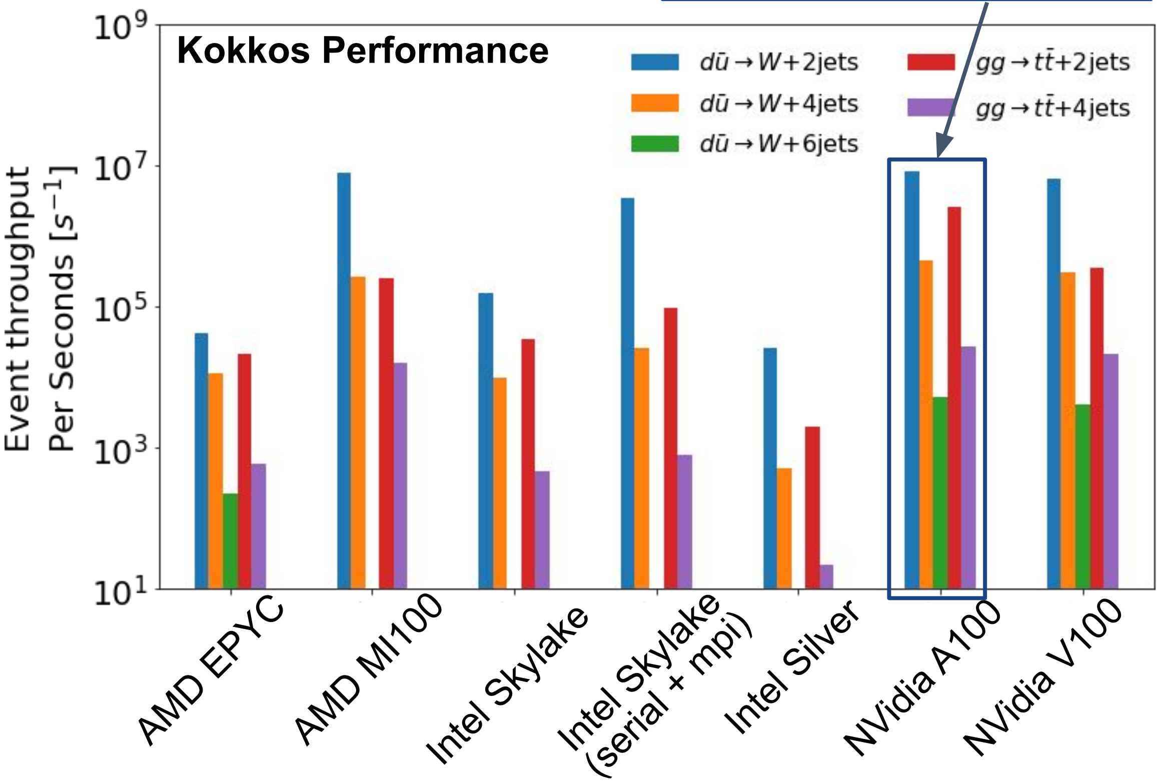
- Tested Kokkos implementation with benchmarking processes
  - ttbar + jets & W+jets
- Kokkos with CUDA backend on Nvidia GPUs
- Kokkos with OpenMP backend on AMD and Intel CPUs
- Kokkos with ROCM/HIP backend on AMD GPUs
- Used the serial C++ algorithm run with MPI to fill a Skylake for comparison to original
- Caveat: no time has yet been spent studying differences



Hardware Comparison

- Tested Kokkos implementation with benchmarking processes
  - ttbar + jets & W+jets
- Kokkos with CUDA backend on Nvidia GPUs
- Kokkos with OpenMP backend on AMD and Intel CPUs
- Kokkos with ROCM/HIP backend on AMD GPUs
- Used the serial C++ algorithm run with MPI to fill a Skylake for comparison to original
- Caveat: no time has yet been spent studying differences

Dependence on the complexity of the process

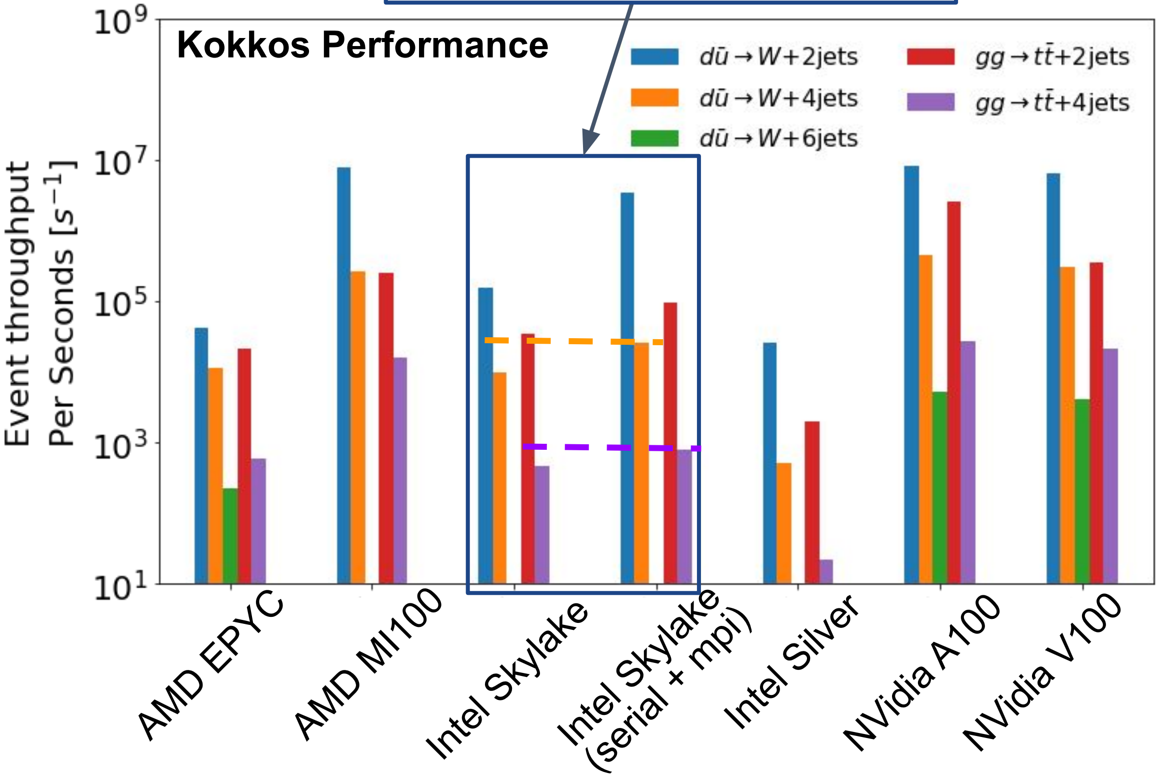


Argonne NATIONAL LABORATORY

Hardware Comparison

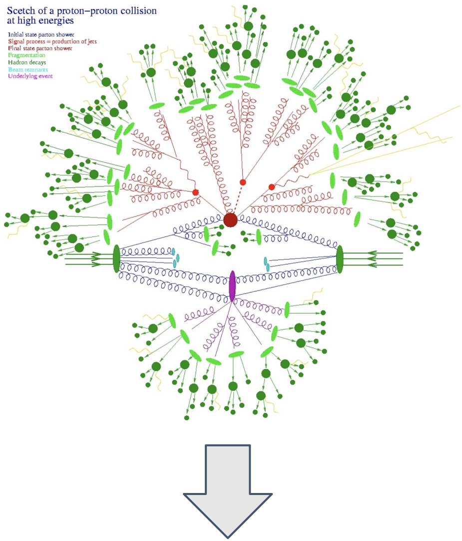
- Tested Kokkos implementation with benchmarking processes
  - ttbar + jets & W+jets
- Kokkos with CUDA backend on Nvidia GPUs
- Kokkos with OpenMP backend on AMD and Intel CPUs
- Kokkos with ROCM/HIP backend on AMD GPUs
- Used the serial C++ algorithm run with MPI to fill a Skylake for comparison to original
- Caveat: no time has yet been spent studying differences

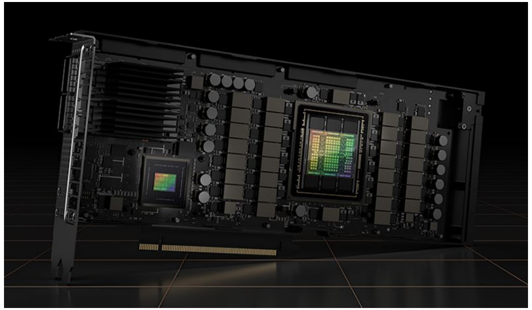
Comparable performance when moving to high jet multiplicity



Summary

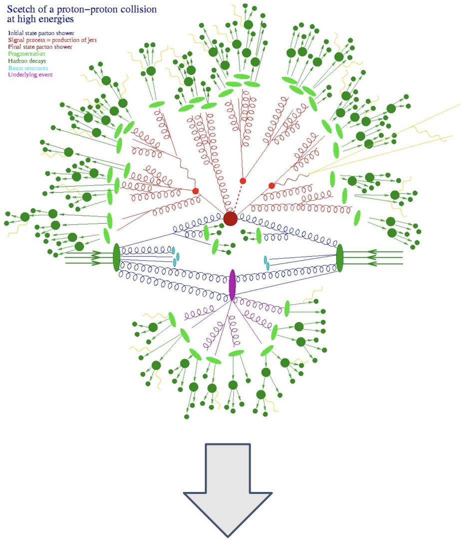
- New Algorithms that can utilize accelerators for HEP simulation are needed to achieve the scientific goals of the LHC in a timely fashion.
- *Blockgen* is such an algorithm.
- Kokkos offered a relatively pain free method for writing physics algorithms that run on multiple architectures and maintain reability.
- While the performance of Kokkos may not be equal to native frameworks, it gets within 10% in the computationally intensive algorithms.
- Kokkos is not an industry product, it comes from the HPC scientific community and can be extended as needed to support future architectures.

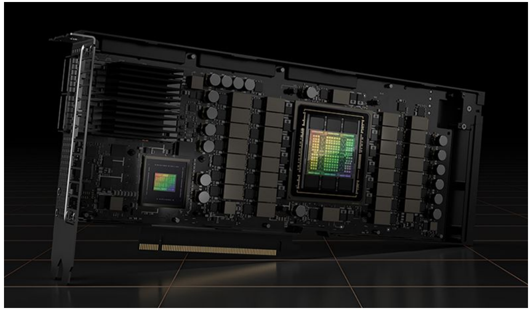




Next Steps

- The Matrix Element calculations have been ported to Kokkos
- The performance will be investigated in collaboration with the Kokkos developers who are very helpful.
- Now we need to integrate these into a proper Leading-Order Event Generator.
    - This work is largely done on the C++ side.
- Physics validation





Acknowledgements

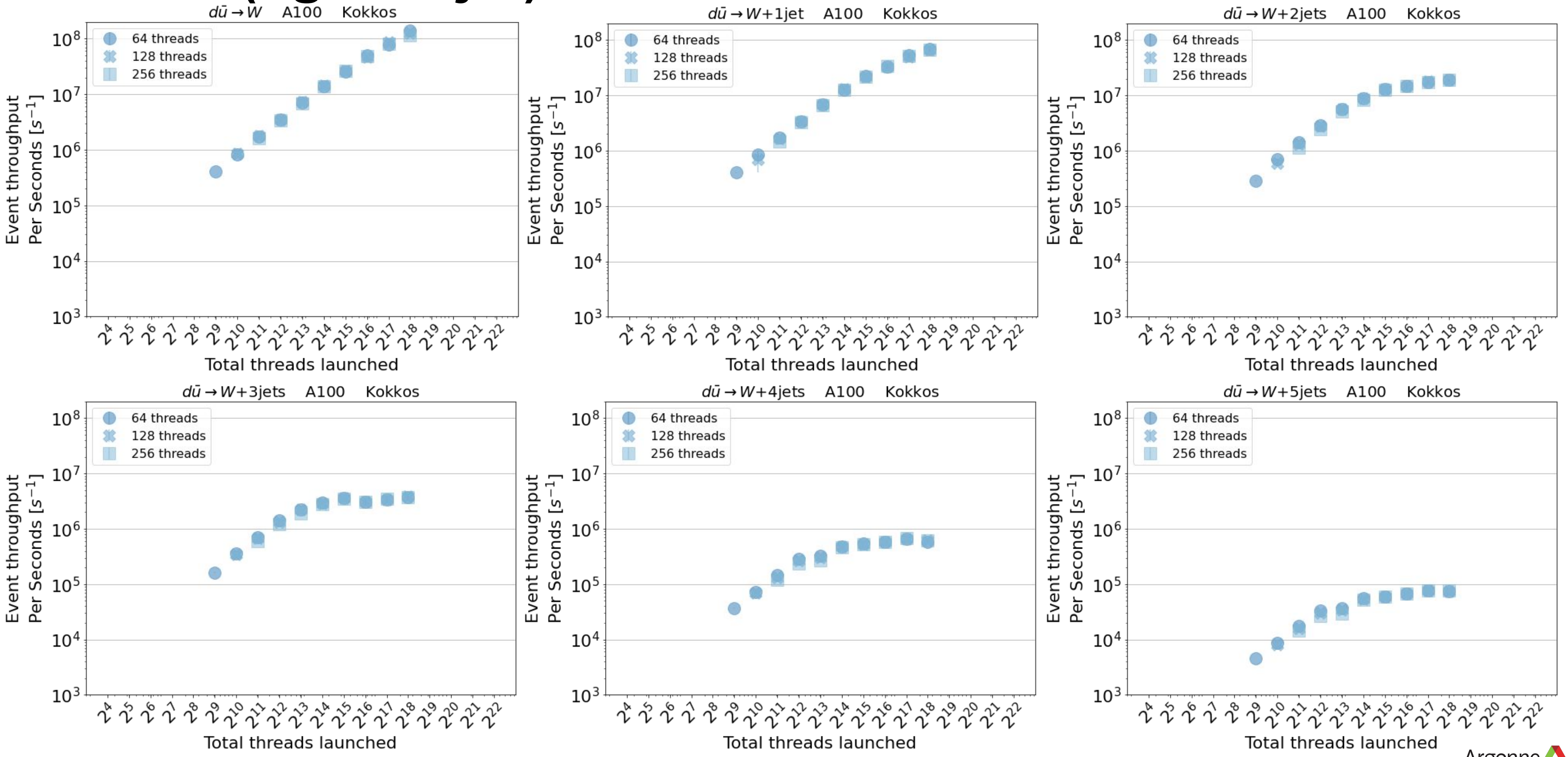
We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

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.

This work was supported by the US Dept. of Energy's Office of HEP Center for Computational Excellence.

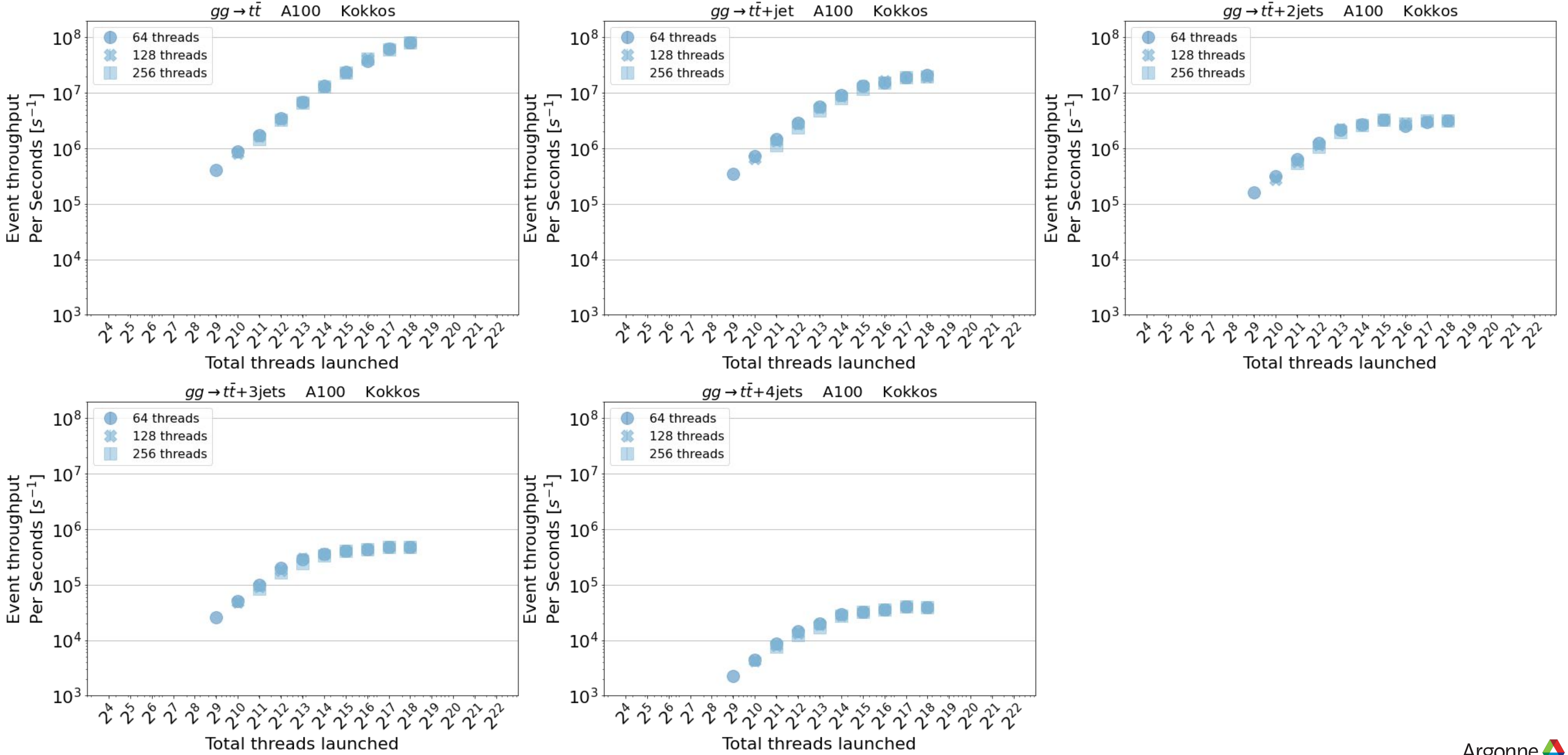
Backups

Z→ee (+gluon jet)



Argonne NATIONAL LABORATORY

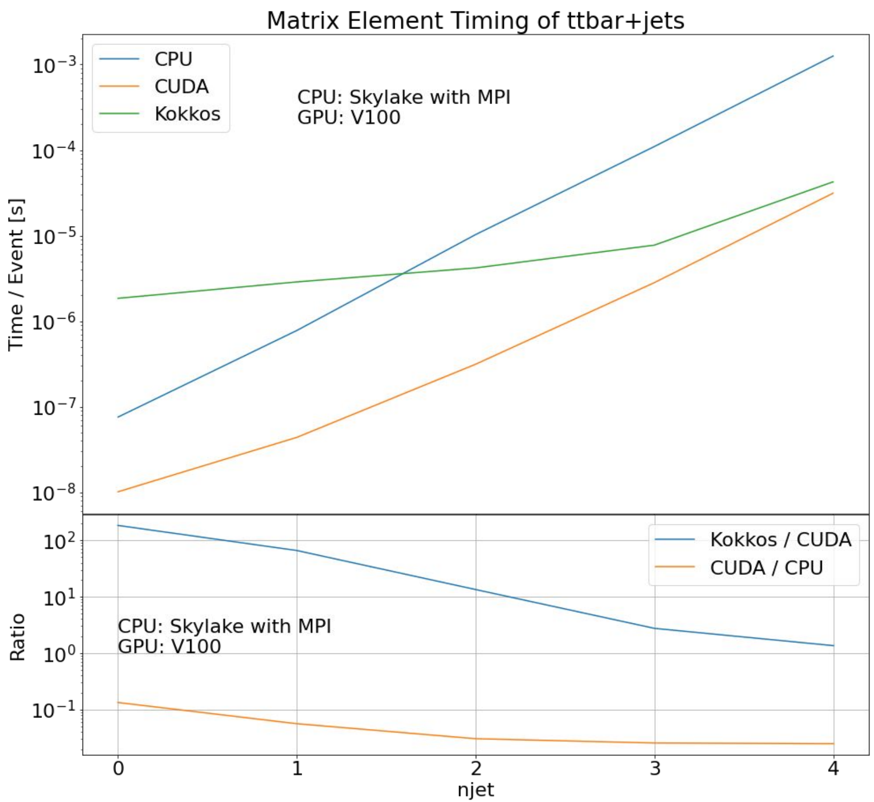
ttbar (+gluon jets)



Argonne NATIONAL LABORATORY

Kokkos vs CUDA vs CPU – latest BlockGen version

- Compare CPU with C++, GPU with CUDA, and GPU with Kokkos
- Can see the CUDA is ~10 – 100 times faster than the CPU for this example
- Kokkos has a negligible overhead comparing to CUDA at low multiplicity (low computational complexity), but reaches comparable performance as multiplicity increases.



Argonne NATIONAL LABORATORY