PANEL: EXECUTION AND PROGRAMMING MODELS – EXTREME SCALE AND BEYOND

WORKSHOP ON DATA FLOW MODELS AND EXTREME-SCALE COMPUTING

CJ Newburn, Principal Architect, NVIDIA Compute SW July 19, 2019

PROGRAMMING MODEL VS. EXECUTION MODEL

The talk vs. the walk

- Expose what matters, hide what doesn't
- Focus expressivity to minimize time to performance

- Facilitate asynchrony for increasing uncertainty
  - Factor work into tasks
  - Make dependencies explicit or discernable vs. implicitly blocking
- Facilitate locality for increasingly-distributed topologies
  - Hierarchical decomposability and retargetability

GRANULARITY

Either multiple abstractions or less-brittle abstractions

- Fine
  - O(10K) nodes, super low overhead wrt data movement, sync
  - Kokkos with persistent kernels
- Medium-fine
  - Execution times in O(us), amortize HW allocations
  - CUDA Graphs, newer Kokkos efforts
- Programmer influence
  - May hard code
  - May make flexibly hierarchical, e.g. AMR
  - Explored hierarchy in HiHAT forum (HIHAT, AMR, ExaTensor, SLATE, PNNL)

ABSTRACTION

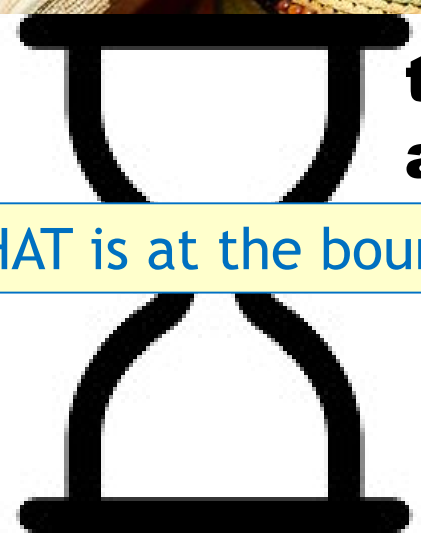
Generality

- Continuation, work creation
- Don’t over-tune for breakable assumptions

Layering

- Tasking runtimes beneath other runtimes
- Enable portability
- MPI/OpenMP/streams/HIHAT Graphs/ CUDA Graphs
- C++ lambdas/executors/HIHAT Graphs/ CUDA Graphs

HiHAT.modelado.org



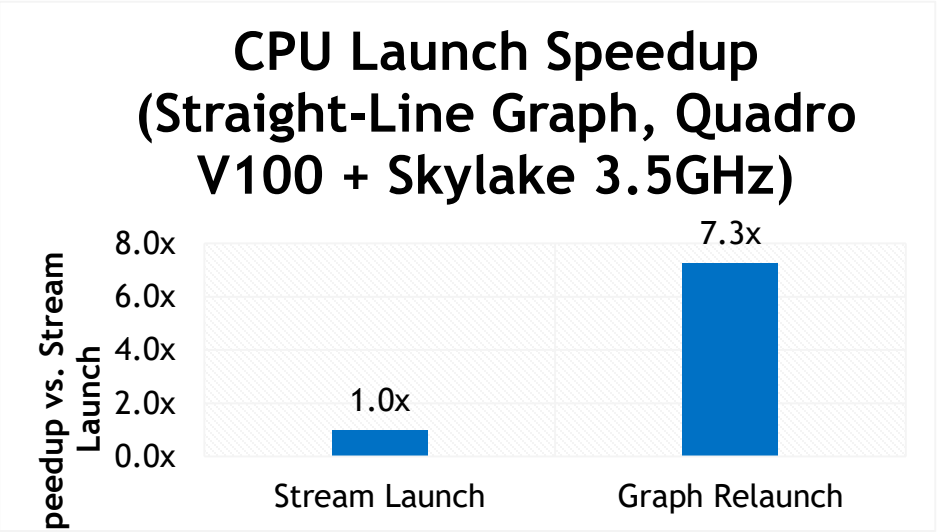
target agnostic

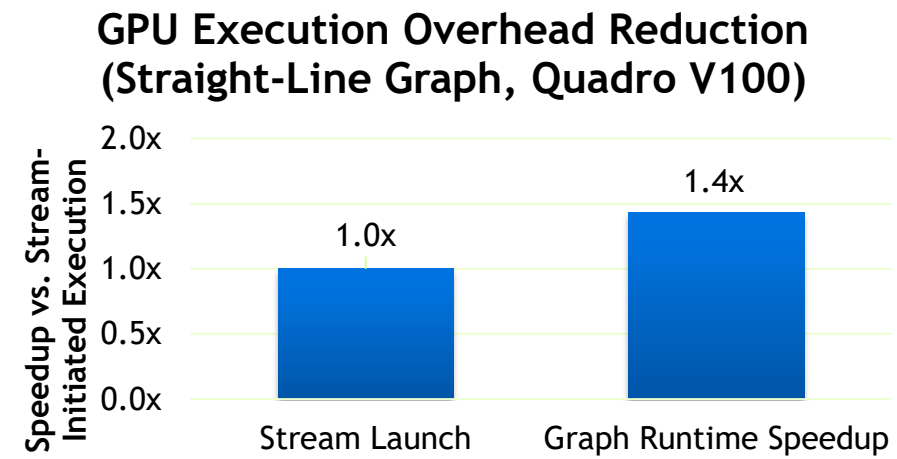
HiHAT is at the boundary

target specific

EFFECTIVENESS OF TASKING

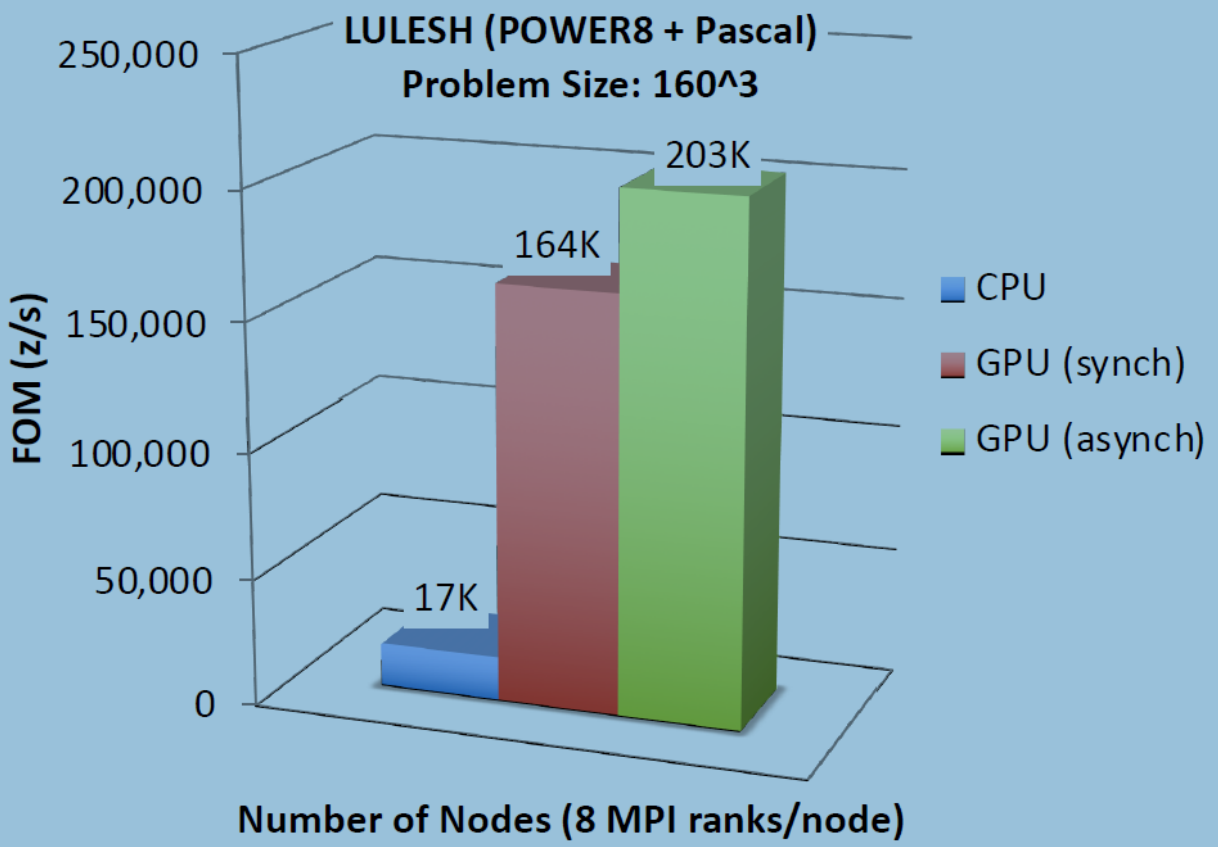
- Productivity = time to enduring performance
- Inertia to refactor codes, build infrastructure
- Infrastructure must be minimally designed with all requirements in mind, esp perf
- Sample benefits with CUDA Graphs (CUDA 10.x)
  - graph launch 7-8x faster on CPU vs. indiv kernels
  - 1.4x faster on GPU from bulk scheduling
  - Coming: control flow in graphs
- Greater generality of HiHAT Graphs (prototype)
  - Broader applicability to more targets
  - Dependencies among graphs





ADDITIONAL CONTENT

LULESH Performance: Asynchronous Execution Improvements



Comparison of the application throughput (Zones/Second Figure-of-Merit metric) on:

- **2 POWER8 CPUs**
- **4 Pascal P100 GPUs – *Synchronous* kernels dispatch**
- **4 Pascal P100 GPUs – *Asynchronous* kernels dispatch**

Offloading computation *asynchronously* to the GPUs yields a **~12x** performance improvement (vs running OpenMP on the CPUs)

~24% better performance with asynch. offloading of target regions (vs synch. offloading)

Courtesy of Kelvin Li, IBM

MPI-OPENMP-STREAMS INTEGRATION (WIP PROPOSAL ONLY)

Dependence chaining, asynchronous copy, with interop

```
#pragma omp task depend(out:ptr1)
    MPI_Recv(ptr1, … MPI_COMM_WORLD)
#pragma omp interop(backend) depend(in:ptr1,out:dev_ptr) device(devno) {
    omp_native_handle_t e;
    stream = omp_get_native_property(e, OMP_BACKEND_QUEUE);
    cudaMemCpyAsync(dev_ptr, ptr1, …, stream);
}
#pragma omp target device(devno) nowait depend(out:ptr2)
map(out:ptr2[0:n]) is_device_ptr(dev_ptr)
    f(dev_ptr, ptr2); // complete f(), copy ptr2 back before dependence is satisfied
#pragma omp task depend(in:ptr2) device(devno)
    MPI_Send(ptr2, … MPI_COMM_WORLD)
```