Performance of CUDA Unified Memory in CMS Heterogeneous Pixel Reconstruction

Martin Kwok, Matti Kortelainen (FNAL)

vCHEP 2021

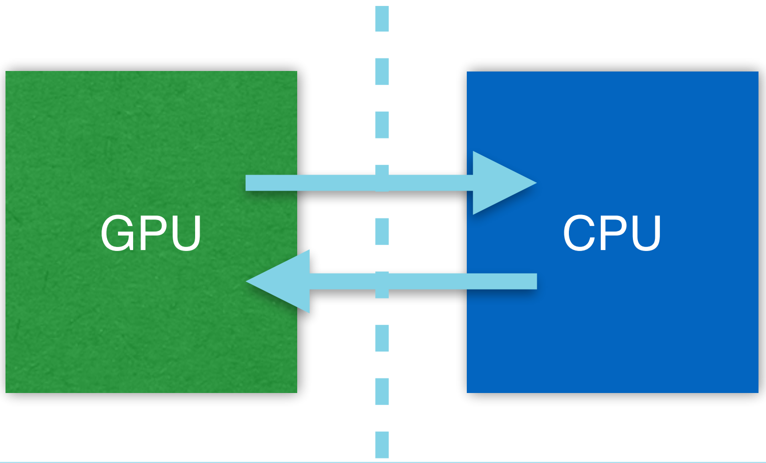
18 May, 2021

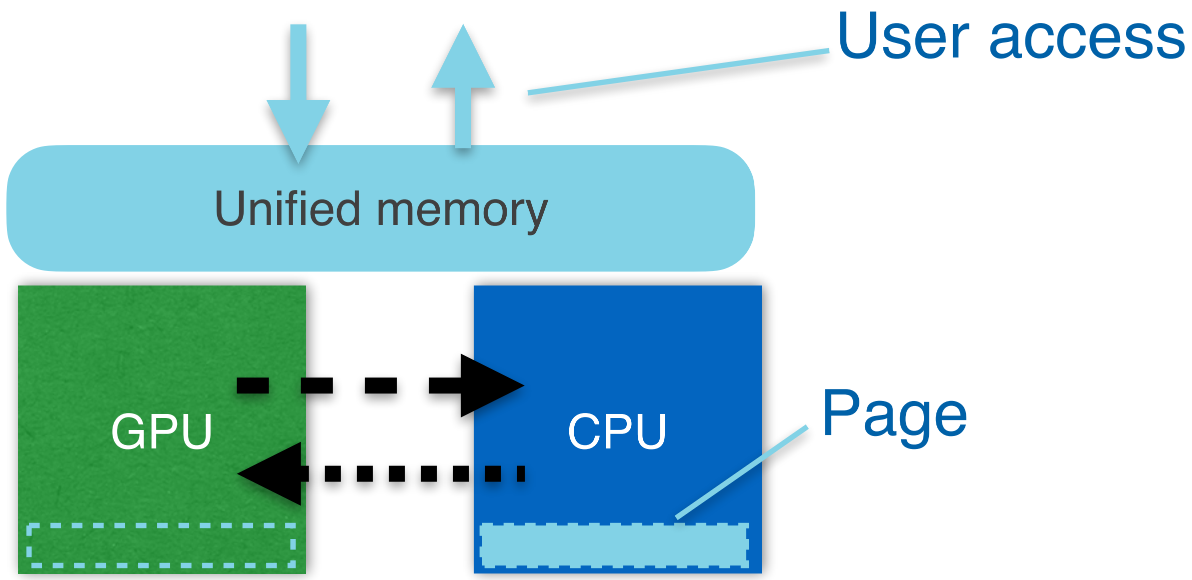
HEP-CCE

Unified Memory

- Managing memory transfer between separate memory space could be a burden in GPU programming
  - Especially when it involves complicated data structures
- In CUDA programming, unified memory aims to provide a single memory space
  - Memory transfers are hidden to programmers, and are done *on-demand* via page faults
- Pros: Easier to write code
- Cons: Performance penalties, e.g. overhead caused by the page faults
  - Can be mitigated via data prefetching

- Use CMS heterogenous pixel reconstruction as a realistic use case to evaluate the performance impact
  - Original code is fully integrated in CMSSW
  - To be run in HLT farm in 2022

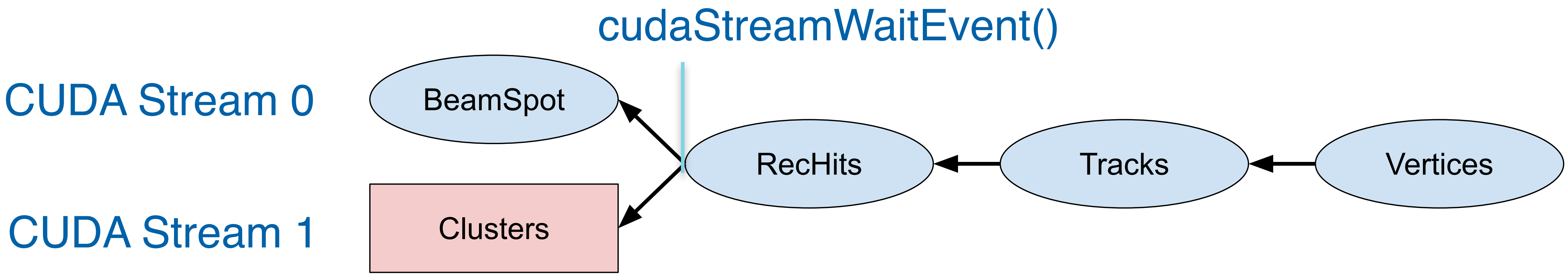




CMS Heterogeneous Pixel Reconstruction

- About 40 CUDA kernels organized in 5 modules
  - arXiv:2008.13461
- Extracted from CMSSW into a *standalone* application for flexibility
- Input: Raw data in pixel detector (~250 kB/event)
  Output: pixel tracks and vertices (~ 4MB for tracks, ~90 kB for vertices)
- Test data: Recycled 1000 $t\bar{t}$ events + pileup 50 simulation from CMS Open Data

- BeamSpot/Clusters/RecHits transfer data from host to device
  - *Clusters* module is only modules that transfer data from device to host
- Events are processed concurrently using TBB Tasks
  - On the device, BeamSpot/Clusters get separate CUDA streams
- What we include in the time measurement: H2D transfer + kernel time
- Not included: disk I/O, transfer of output



Programming Experience with Unified Memory

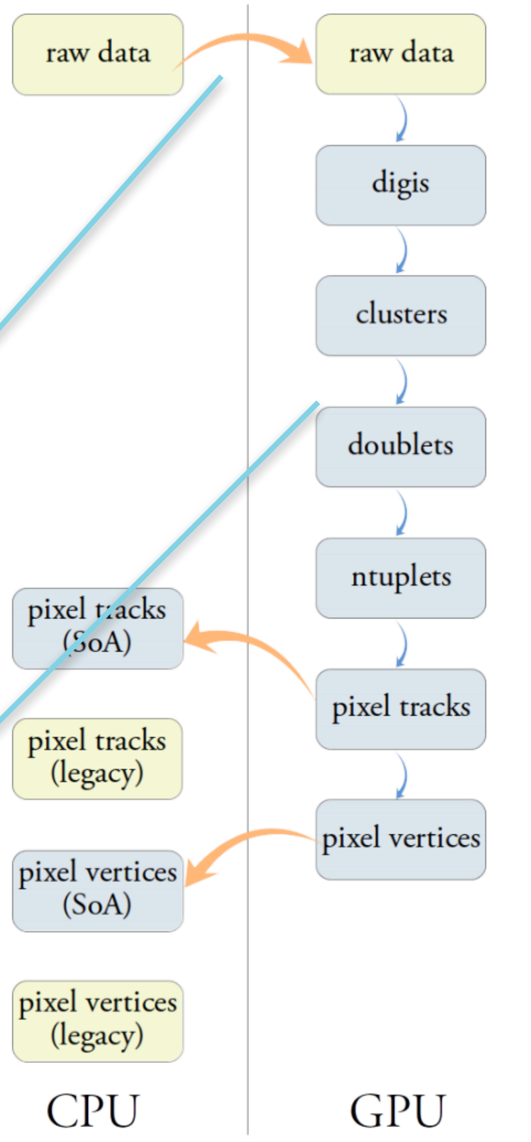
- The benefit brought by unified memory depends on the application

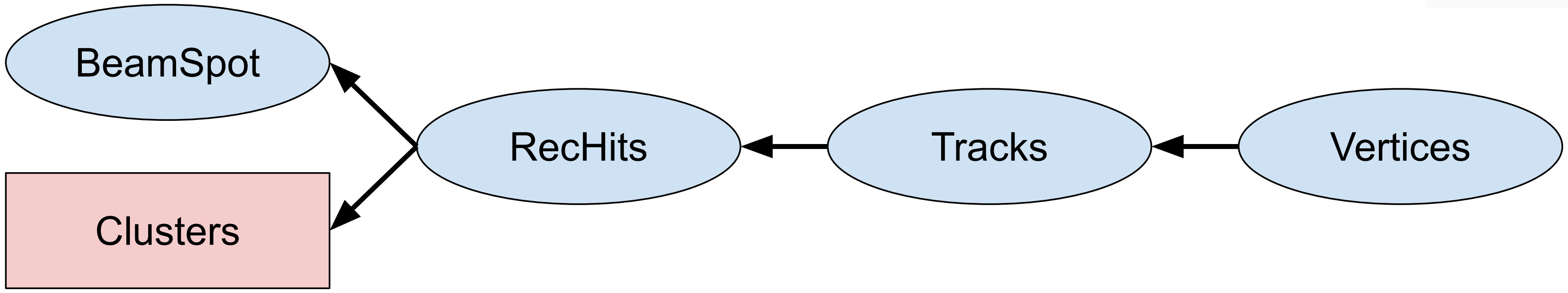
- Significantly easier to use unified memory on condition data
  - Transfer only once in the beginning of the job
  - Otherwise need to allocate and transfer memory to *each* GPU device while keeping host pinned memory alive

- Not much benefit for applying unified memory on event data
  - Would be helpful for data structure using pointers of pointers
  - Not heavily used in Patatrack

Condition data

Event data



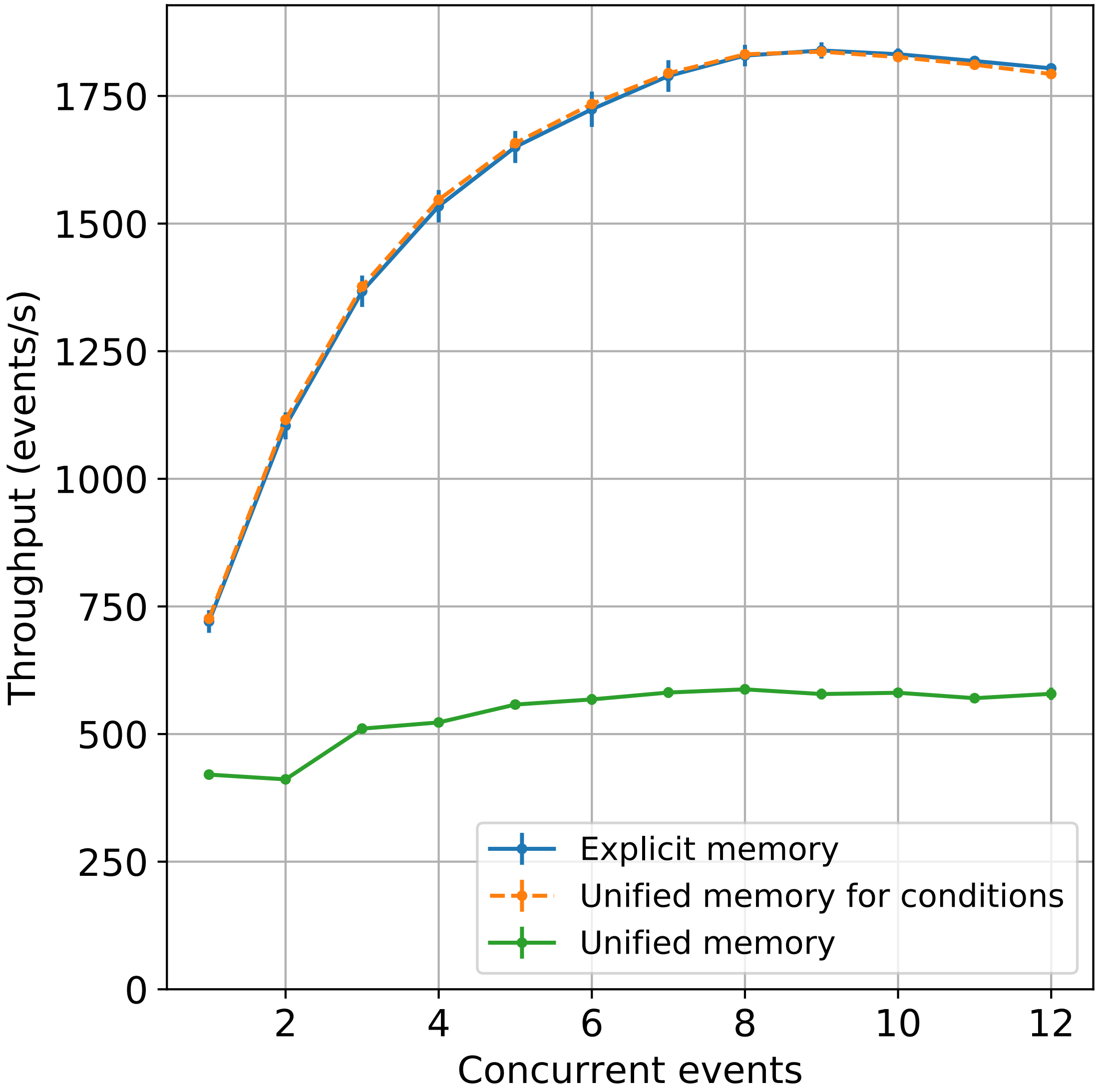


Performance

- Measured on Cori GPU nodes at NERSC using a single GPU (NVIDIA V100)
  - Intel Xeon Gold 6148 ("Skylake") processors with 20 cores, 2 threads per core
  - No other activities on the CPU, all threads are pinned to a single socket
  - Repeated 8 times on random nodes (shown as the uncertainty)
  - Each job takes around 5 min, processing the set of 1000 events multiple times
  - Use explicit memory result as reference

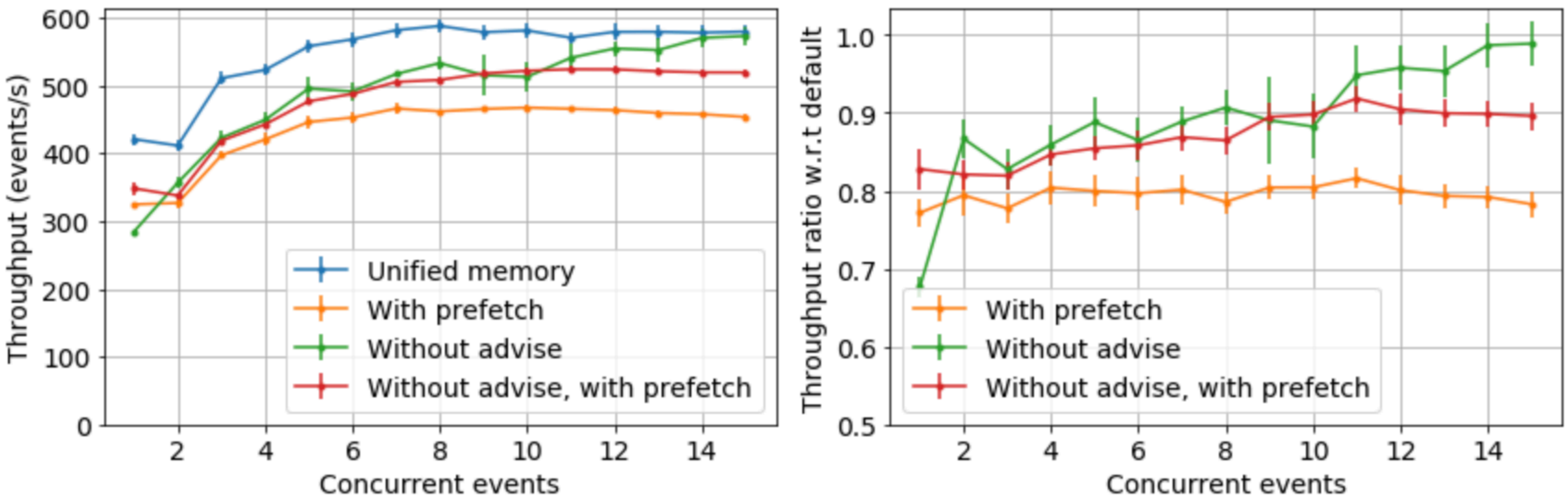
- When unified memory on condition data is used, throughput is within 1% of explicits memory result.
  - This is expected as the memory transfer is only done once.
- When unified memory is used, drop to 33-50% of explicit memory throughput

- General trend of lower throughput with more modules using unified memory



Data Prefetching / Memory advise

- We tried to use two features designed to reduce the performance penalty
  - **Data prefetching**: Intended to avoid page faults by prefetching the data before access
  - **Memory advise**(read-only): Provide hints for CUDA that specific memory ranges are read-only
    - Use on condition data & data transfer from host to device
- 4 possible combinations: (with/without advise) x (with/without prefetch)
  - Best performance: With advise, but ***without*** data prefetch (blue)
  - Memory advise only gives better performance (~15-20%) (blue/green) when it's done without data prefetching
  - Data prefetching only gives better performance (~10%) (red/orange) when it's done without advise



Summary

- We tested the performance of CUDA unified memory with CMS pixel reconstruction Patatrack as a realistic HEP use case
- Performance penalty from unified memory could be very significant (~50-70%)
  - Contrary to expectation, enabling data prefetching could decrease the performance
- Benefit brought by unified memory is *less* attractive if heavy-fine tuning is needed to avoid steep performance penalty
- What we found could be applicable to more than just CUDA:
  - Other GPU programming models have similar ideas as unified memory
  - For example, NVidia's compiler support for portable code via C++ between CPU and GPU relies on unified memory
- Performance reduction could be related to lock contention of the global mutex within CUDA runtime
  - To be confirmed with detailed profiling