Geant4 11.1-p02

Heterogeneous Computing G4Tasking

10th International Geant4 Tutorial in Korea

Dennis Wright

6-10 November 2023

Slides based on those of Soon Jung Jun (Fermilab) and Makoto Asai (Jefferson Lab)

Outline

Introduction

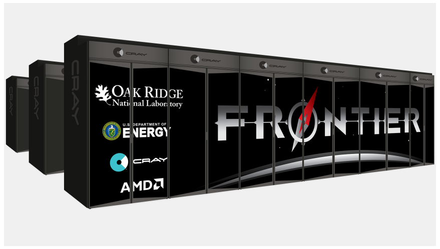
Heterogeneous Computing

- Definition: computing which uses more than just CPUs
  - move from heavily CPU-centric to combinations of CPUs, accelerators, GPUs,
  - TPU (tensor processing unit)
  - NPU (neural processing unit)
  - smartNIC (smart network interface card)
  - FPGA, ASIC, …
- Goal: optimize three things simultaneously
  - power, performance, cost
  - maximize workload by efficient software stacks
- Take advantage of verticalization
  - software, hardware designed and organized to perform specific tasks, rather than general

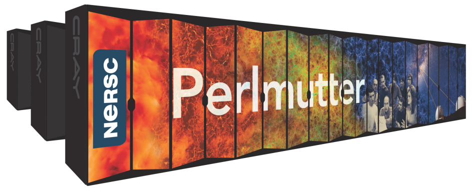
Computing Hardware

- Taking advantage of increasing heterogeneity in computing resources
  - e.g. exa-scale computers at DOE facilities with GPUs

    - AMD



NVIDIA



    - Intel

U.S. Department of Energy
NNSA
Lawrence Livermore National Laboratory
CRAY
EL CAPITAN

AMP

- Much HEP-ESC related R&D
  - Coprocessors as a service
  - PPS (portable parallel strategies)
  - Sonic (services for optimized network interference on coprocessor) and Adios (advanced data I/O service)
  - High-performance GPU-based algorithms for multi-dimensional numerical integration

Tasking

Task Computing

- Task: a unit of execution, or unit of work
  - can mean process, light-weight process, thread, step, request or query
  - in Geant4, each G4DynamicParticle or G4Track can be a task

- Task-based parallelism: distributes tasks across different processors in parallel computing environment
  - distributed tasks run on the same data
  - task-based, event-level parallelism has been in Geant4 since 11.0

Geant4 Tasking

- Geant4 supports a task-based framework (G4Tasking)
  - Based on PTL (parallel tasking library by J. Madsen, lightweight C++11 multi-threading tasking system featuring thread pool, task-groups, and task queue) or TBB (Thread Building Blocks) backend
  - Code and examples:
    - source/tasking
    - example/extended/parallel/ThreadsafeScores
- G4Tasking opens opportunities for heterogeneous computing
  - sub-event level parallelism (from events to tracks)
    - a group of particles can be handled by a thread-pool
    - a group of special tasks can be a task group
  - concurrent simulation workflow with coprocessors
    - off-loading specialized tasks to GPUs (e.g. optical photons)
    - hybrid computing model: e^-, e^+, γ, n on GPUs, all else on CPUs

Monte Carlo Simulation on GPUs

- Challenging to port fully detailed detector to GPU
  - Each GPU process should have strictly limited scope
    - limited physics coverage
    - particular particle types
    - specific geometry/material
  - Examples:
    - optical photon transport in Cerenkov detector
    - EM shower in a calorimeter

- GPU task should behave like a black box
  - the blacker the box (less output), the better the performance
  - individual steps or tracks should not be taken out of a task
  - don’t re-shuffle tracks over tasks
  - minimize output information

- Sub-event parallelism is the only solution that allows various tasks running on GPUs in parallel while conducting the full event simulation

Heterogeneous Simulation with Geant4

- Simulation throughput needs to increase by 10 - 100 times
    - Lots of simulation needed to maintain small systematic uncertainty
    - New detector hardware -> greater demands on simulation

- Strategy: heterogeneous computing
    - Master process manages sub-tasks
    - Each sub-task has strictly limited scope (particle type, physics, detector)

- Use CPUs when
    - sustainable or no alternative
    - creativity required: stack management, event biasing, …
    - management or integration of granular sub-tasks required

- Use GPUs (or co-processors) when
    - there is a variety of single-purpose, optimized sub-tasks
    - output must be minimized

Sub-event Parallelism in Geant4

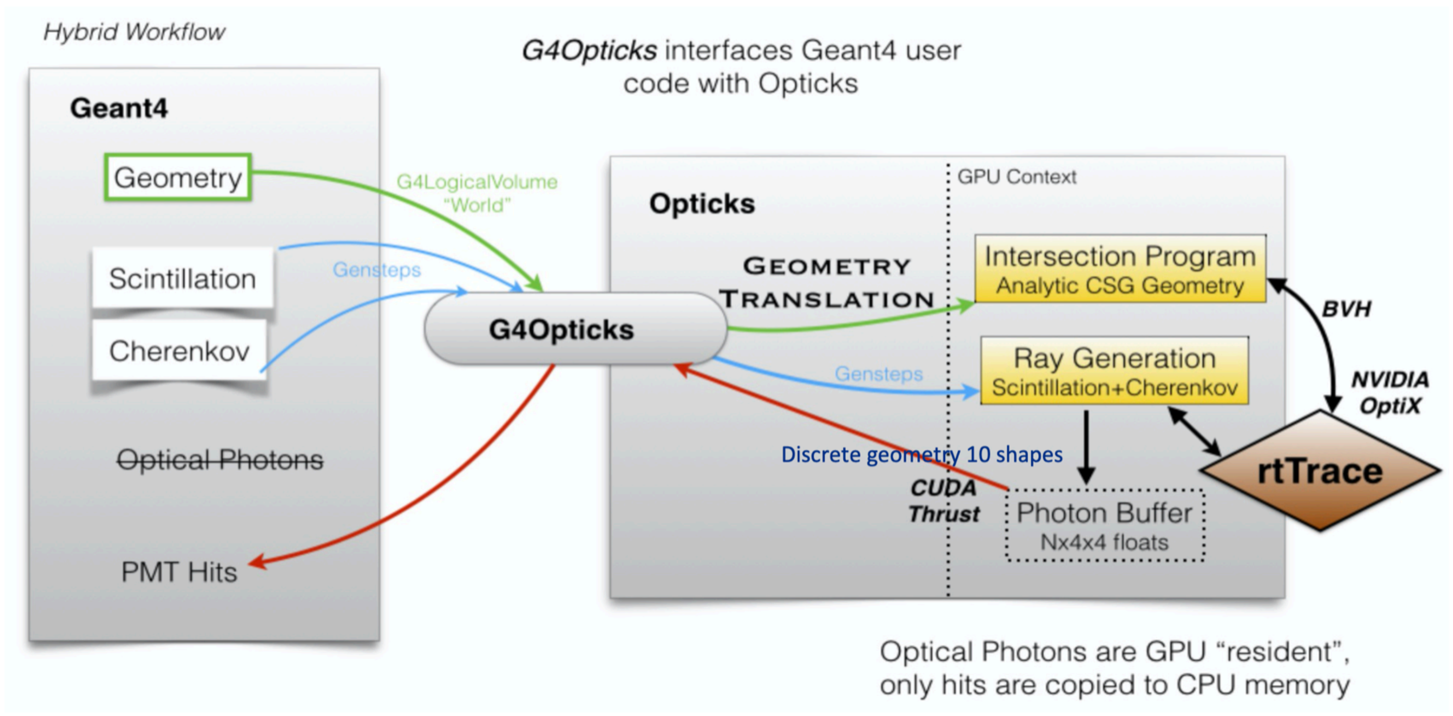
- Sequential, threading and tasking modes will continue to work fine
- Using sub-event parallelism (for phase I)
  - Use a newly introduced RunManager
  - Implement UserStackingAction to sort tracks into sub-events
  - Implement merge() method in G4Event if special merging treatment is required
    - Ordinary HitCollection and HitVector will be automatically merged
- Using sub-event parallelism (for phase II):
  - In addition to above
  - Physics list and/or detector construction dedicated to each task if needed
    - Example: G4HepEM (designed to be compatible with GPUs)

Examples

Opticks/OptiX

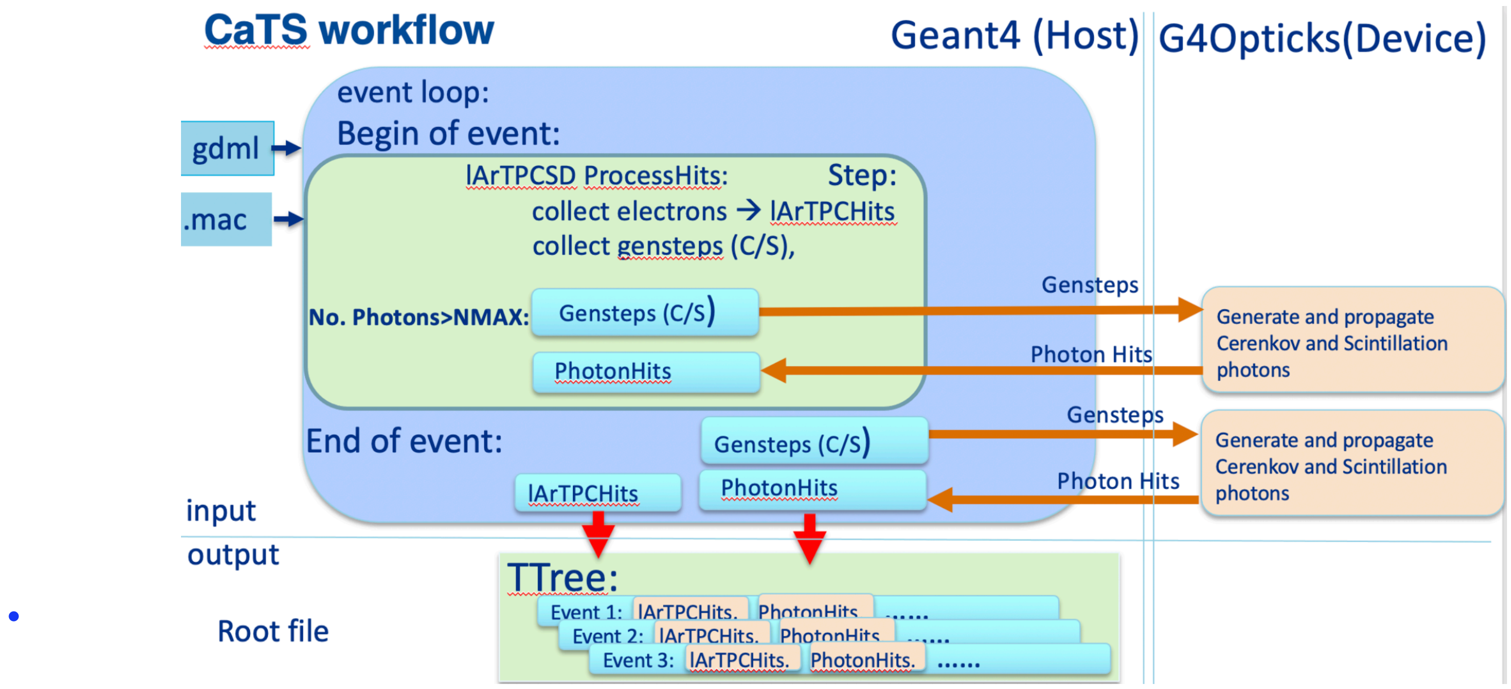
- Opticks: open-source project that accelerates optical photon simulation by integrating NVIDIA GPU ray-tracing, accessed by NVIDIA OptiX
  - developed by Simon Blyth for Daya Bay and JUNO experiments
  - https://bitbucket.org/simoncblyth/opticks

Figure from Simon's presentation



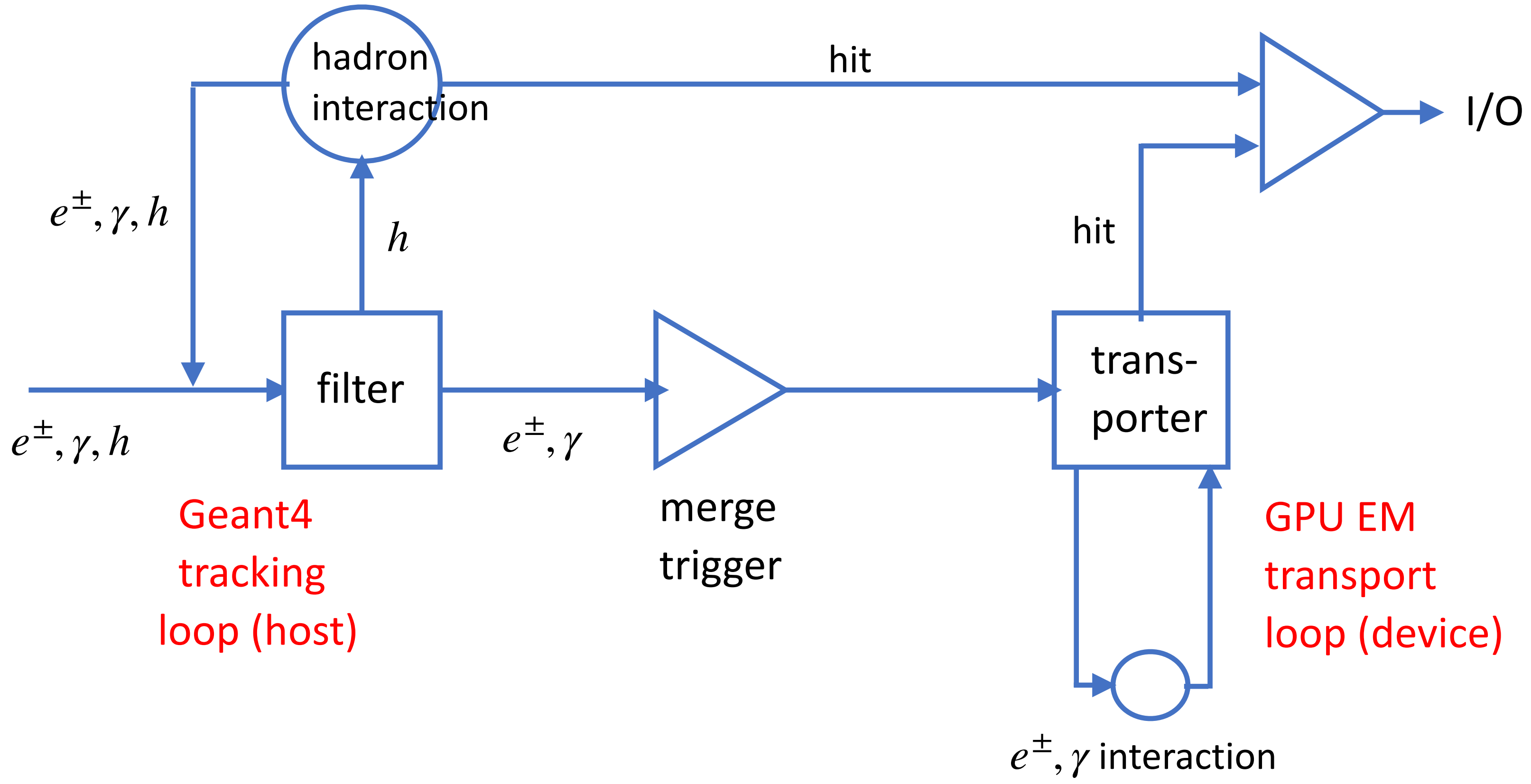
Example 1: CaTS

- Advanced example using Opticks/OptiX (examples/advanced/CaTS)
- Uses Geant4 to collect GenSteps from G4Scintillation and G4Cerenkov
    - Genstep contains all data required to generate scintillation/cerenkov photons on GPUs
- Opticks generates optical photons on GPU and traces them using NVIDIA Optix while Geant4 handles transport of all other particles on CPUs



Example 2: EM Particle Transport on GPUs

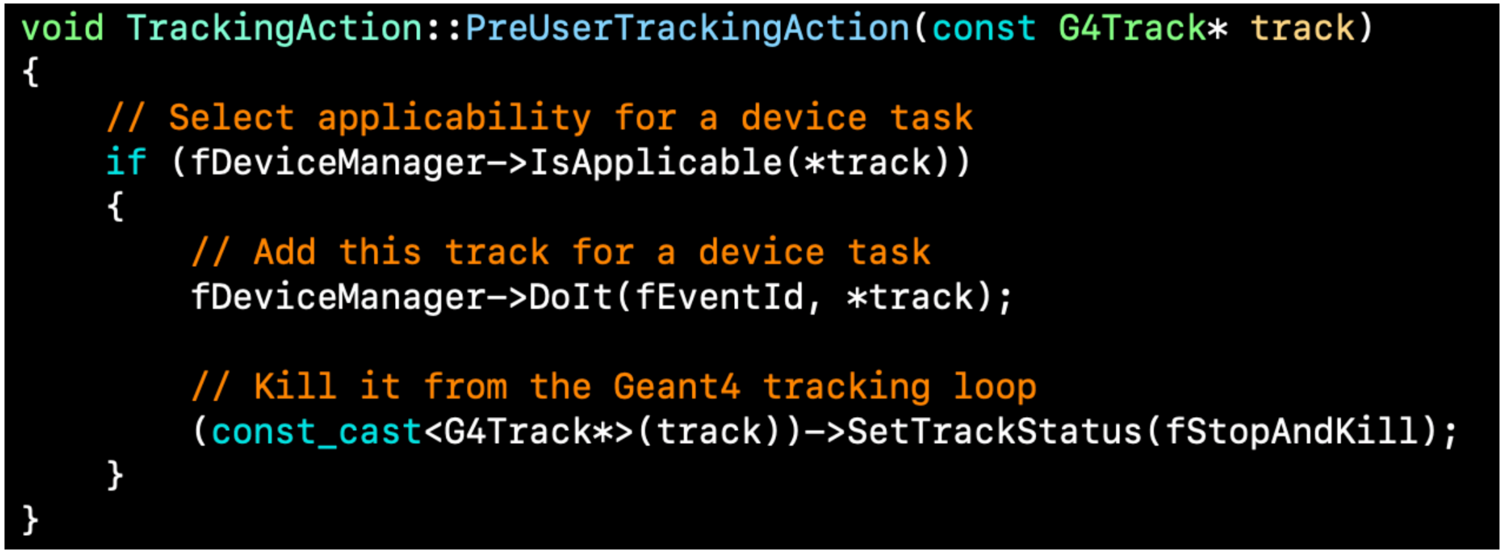
- Hybrid workflow with selected tasks executed on GPUs
  - Example: a Geant4 application which offloads EM particle transport to GPU using user actions. $e^{\pm}, \gamma$



Strategy for Offloading EM Simulation

- Use G4UserTrackingAction and G4TaskRunManager
  - collect specific particles (e.g. EM particles) for device tasks from PreUserTrackingAction
  - execute tasks in G4TaskGroup with a set of stored tracks

- Code example: TrackingAction
  - PreUserTrackingAction
    - select candidate tracks for offloading
    - store tracks in a stack and trigger tasks within the same task group
    - kill the Geant4 track from the Geant4 tracking loop

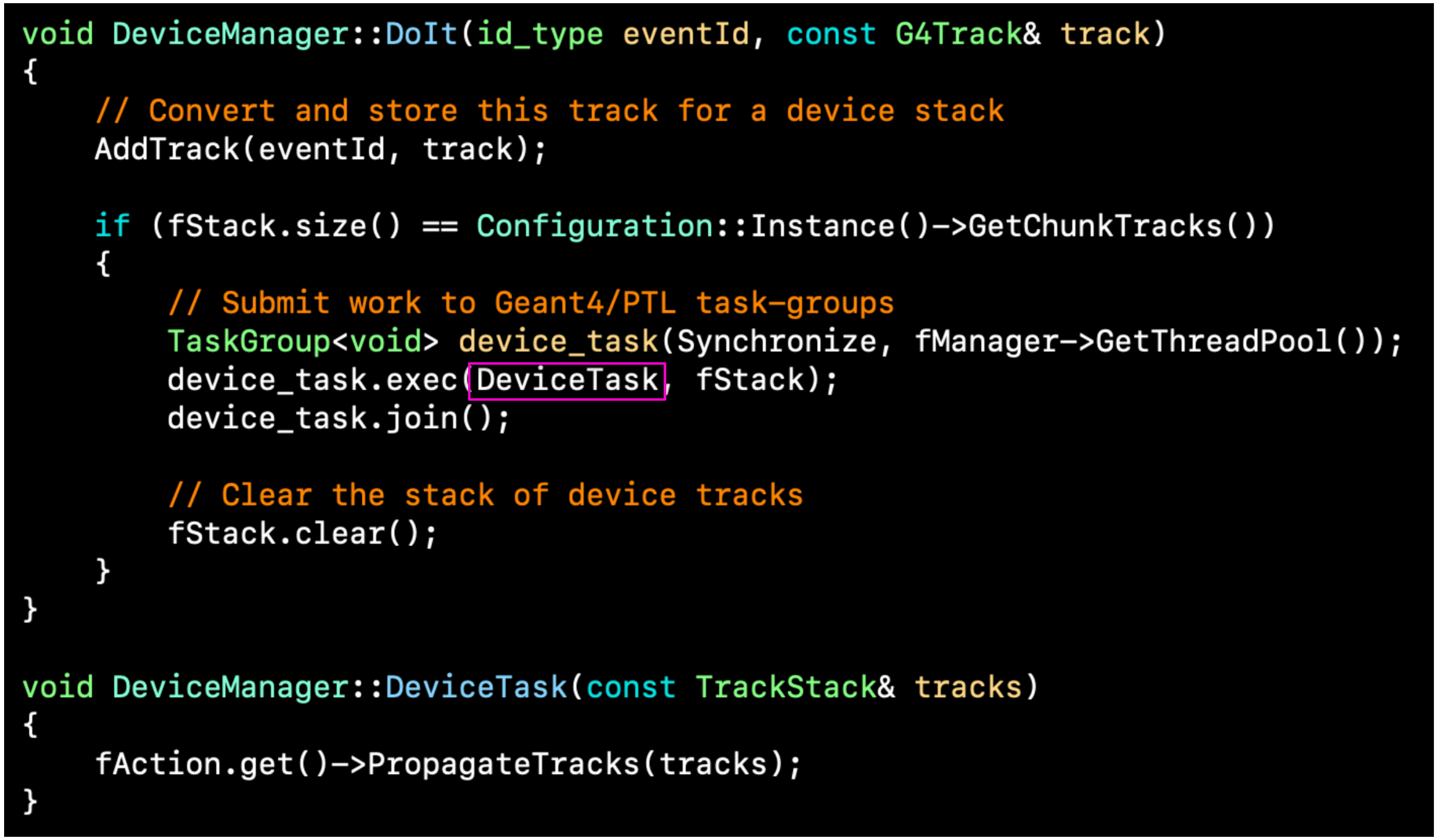


```cpp
void TrackingAction::PreUserTrackingAction(const G4Track* track)
{
    // Select applicability for a device task
    if (fDeviceManager->IsApplicable(*track))
    {
        // Add this track for a device task
        fDeviceManager->DoIt(fEventId, *track);

        // Kill it from the Geant4 tracking loop
        (const_cast<G4Track*>(track))->SetTrackStatus(fStopAndKill);
    }
}
```

-

Example: Device Manager

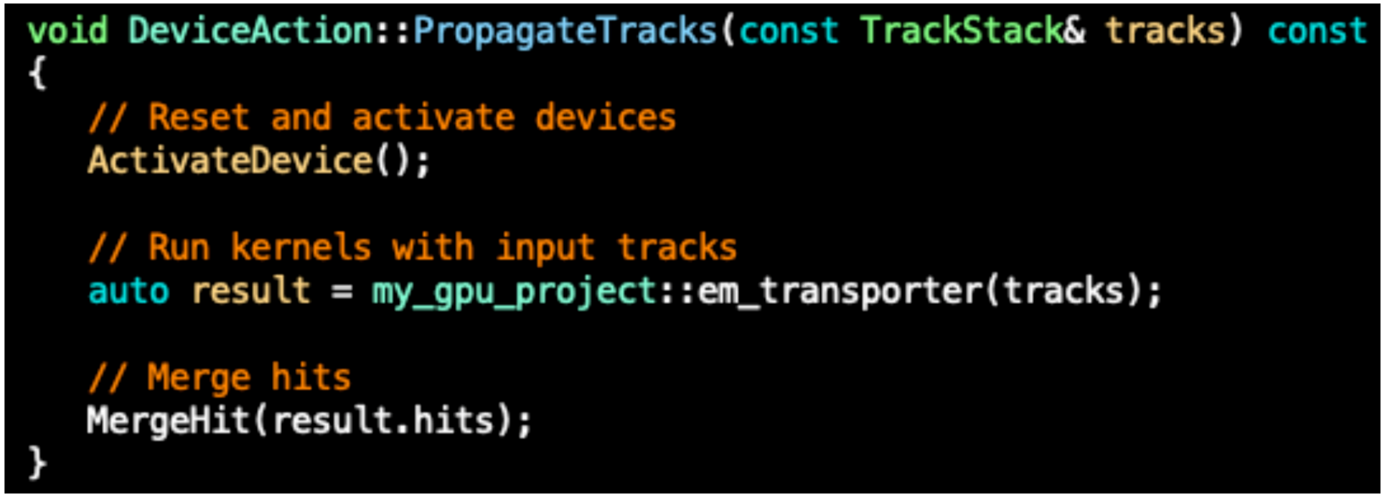
- DoIt actions: add tracks and trigger a device task if criteria are met



```
void DeviceManager::DoIt(id_type eventId, const G4Track& track)
{
    // Convert and store this track for a device stack
    AddTrack(eventId, track);

    if (fStack.size() == Configuration::Instance()->GetChunkTracks())
    {
        // Submit work to Geant4/PTL task-groups
        TaskGroup<void> device_task(Synchronize, fManager->GetThreadPool());
        device_task.exec(DeviceTask, fStack);
        device_task.join();

        // Clear the stack of device tracks
        fStack.clear();
    }
}

void DeviceManager::DeviceTask(const TrackStack& tracks)
{
    fAction.get()->PropagateTracks(tracks);
}
```

Example: Device Action

- Connectivity to an external software suitable for GPU

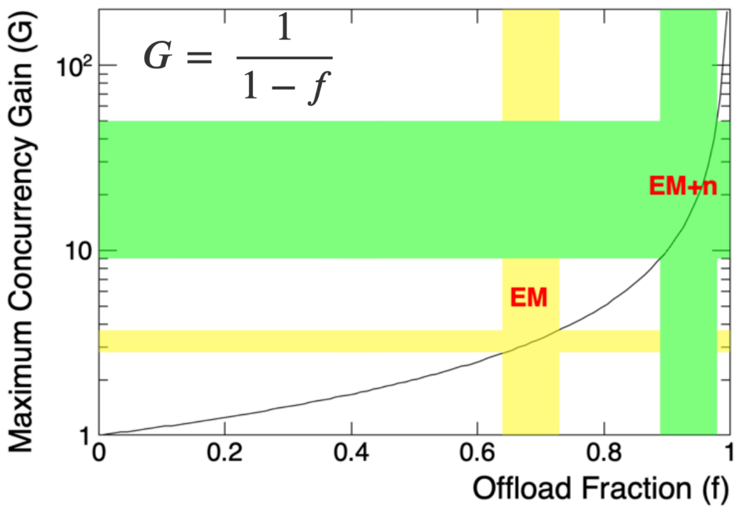


```
void DeviceAction::PropagateTracks(const TrackStack& tracks) const
{
   // Reset and activate devices
   ActivateDevice();

   // Run kernels with input tracks
   auto result = my_gpu_project::em_transporter(tracks);

   // Merge hits
   MergeHit(result.hits);
}
```

- em_transporter(tracks) launches GPU kernels with set of device data (geometry, physics data, input stack of tracks)
- transfer sensitive hits from devices and merge them with Geant4 hits created on host
- process left-over tracks at end of event or run action, depending on how taskings are organized and scheduled
- execute CPU task (currently event-level multithreading), device task, I/O task groups as concurrently as possible (latency hiding)

Expected Performance Gain and Limit

- Limit on the maximum gain (G) by the offloading fraction (f, in terms of time)
  - 1/(1-f) when $time_{GPU} \times f \leq time_{CPU} \times (1-f)$ with full concurrency (assumes no overhead)
  - Example of offloading EM physics processes with cms2018 @LHC



Max gain by GPU

Process	EM	EM+n
QCD(p_T^{min}=1TeV)	2.9	21.1
W+Jets	2.8	22.2
Z+Jets	3.0	22.7
$t\bar{t}$	2.8	21.0
Higgs	2.8	21.1
MSSM	3.4	20.2
Stop	2.8	20.9

  - GPU performance may not be a limiting factor for overall gain if the offloading fraction is relatively low

Summary

- Advanced software tools use a range of architectures to take advantage of increasing heterogeneity in computing resources

- Geant4 supports a task-based framework (G4Tasking) suitable for heterogeneous workflows
    - event-level task parallelism
    - sub-event-level task parallelism
    - (track-level parallelism)

- Examples of ongoing HEP detector simulation projects using GPUs within Geant4:
    - optical photon simulation (Opticks/NVIDIA OptiX) -> CaTS
    - offloading EM particle transport to G4HepEM/Adept and Celeritas/Acceleritas

- GPUs are everywhere even though there are many challenges for fully taking advantage of modern coprocessors for general detector simulation