Heterogeneous Computing G4Tasking

Soon Yung Jun (Fermilab)

9th International Geant4 Tutorial in Korea

Dec 5-9, 2022@KISTI

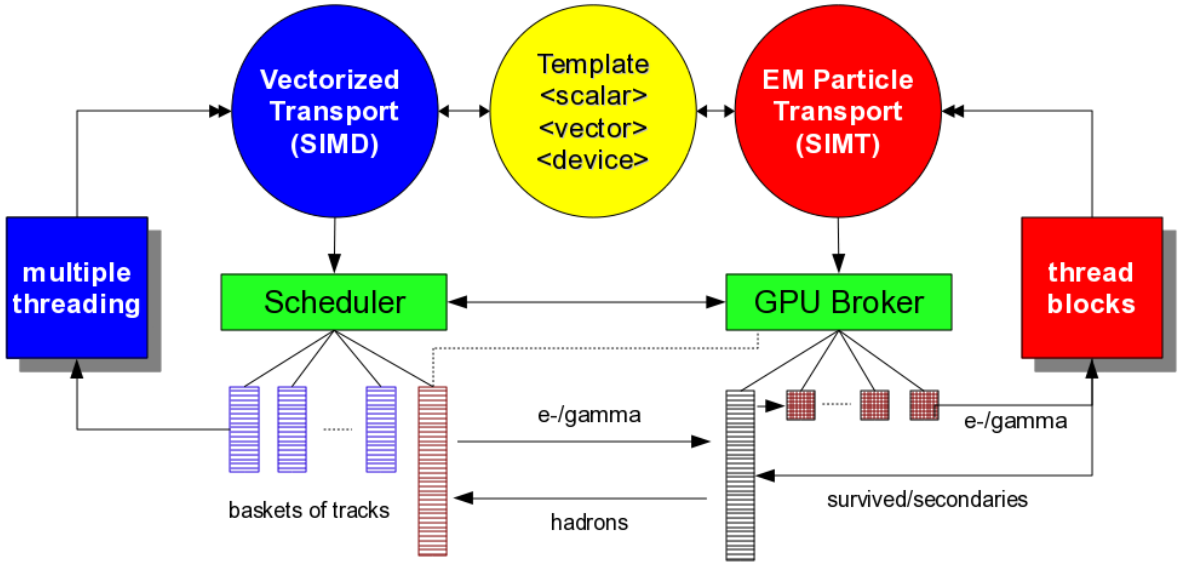
Contents

- Introduction
  - What is heterogeneous computing?
  - Hardware landscape: exascale computing facilities
- Geant4 Tasking
- Examples of Geant4 applications using GPUs
  - Optical photon transport
  - EM particle transport
- Summary

Introduction

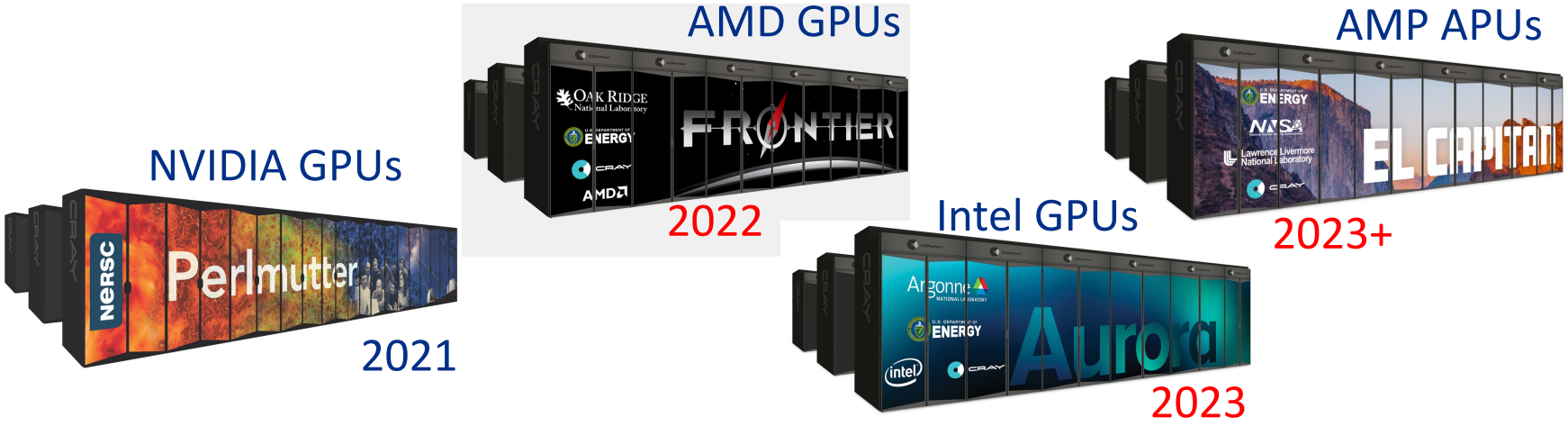
What is Heterogeneous Computing

- Computing with more diverse processing elements other than CPU
  - Heavy CPU centric (HTC) → combination of CPU + Accelerators (GPUs, TPUs, NPU, smart NIC, FPGA etc.)
  - Optimize for very specific domain applications for the triplet (power, performance, cost)
  - Maximize workload by efficient software stacks
  - Verticalization(hardware designs ←→ software stacks)
- Example of task decomposition: CPU + GPU combination



Ever-Changing Hardware Landscape

- Take advantages of increasing heterogeneity in compute resources
- E.g., Exa-scale computers under DOE flagship facilities with GPUs



- Many HEP-ECC related R&D
  - Coprocessors as-a-service (leverage industry hardware and tools)
  - PPS (Portable Parallel Strategies)
  - Sonic (Services for Optimized Network Inference on Coprocessor) and Adios (Advance data I/O service)
  - High-performance GPU-based algorithms for multidimensional numerical integration

G4Tasking

Geant4 Tasking

- Geant4 supports a task-based framework (*G4Tasking*)
  - based on PTL (parallel tasking library by J. Madsen, lightweight C++11 multithreading tasking system featuring thread-pool, task-groups, and task-queue) https://indico.cern.ch/event/809383/ or TBB backend
  - Task-based event-level parallelism since Geant4 v11.0 (Dec. 2021)
    - source/tasking
    - example/extended/parallel/ThreadsafeScores
- G4Tasking opens opportunities for heterogeneous computing
  - Sub-event level parallelism (from events to tracks)
    - Each G4PrimaryParticle or G4Track can be a task
    - A group of a particles can be executed in a thread-pool
    - A group of a special task can be a task-group
  - Concurrent simulation workflow with co-processors
    - Offloading specialized tasks to GPUs (ex: optical photons)
    - Hybrid computing model: $e^{\pm}$, γ or neutrons on GPUs and the rest on CPU

Monte Carlo simulation on GPU (Makoto A.)

- It is challenging to port the full fidelity detector simulation on GPU
  - Each GPU process should have strictly limited scope
  - Physics coverage, particle type, geometry/material
  - E.g., optical photon transport in Cerenkov detector, EM shower in calorimeter (w/o back splash or punch through)
- A task on GPU should behave like a blackhole
  - The darker (less output) a task is, the better performance it has
  - Individual step/track should not be taken out from a task
  - Reshuffling tracks over tasks is no a good thing to do
  - Minimize output information
- Sub-event parallelism is the only solution that allows various tasks running on GPU in parallel while conducting the full event simulation

Heterogeneous simulation with Geant4 (Makoto)

- Simulation throughput needs to increase by O(10-100)
  - Contribution of simulation to maintain the systematic uncertainty
  - New detector hardware comes with higher demands (ex. EIC)
- Strategy: Heterogeneous computing
  - The main (master) process manages sub-tasks.
  - Each sub-task has strictly limited scope with only the limited kind of physics processes per particle type or detector geometry
- On CPU
  - When sustainable or when no alternative
  - With creativity (e.g., stack management, event biasing, ...)
  - Managing / integrating granular sub-tasks
- On GPU (or co-processors, ...)
  - Variety of single-purpose, optimized sub-tasks
  - Output must be minimized

Sub-event parallelism in Geant4 (Makoto A.)

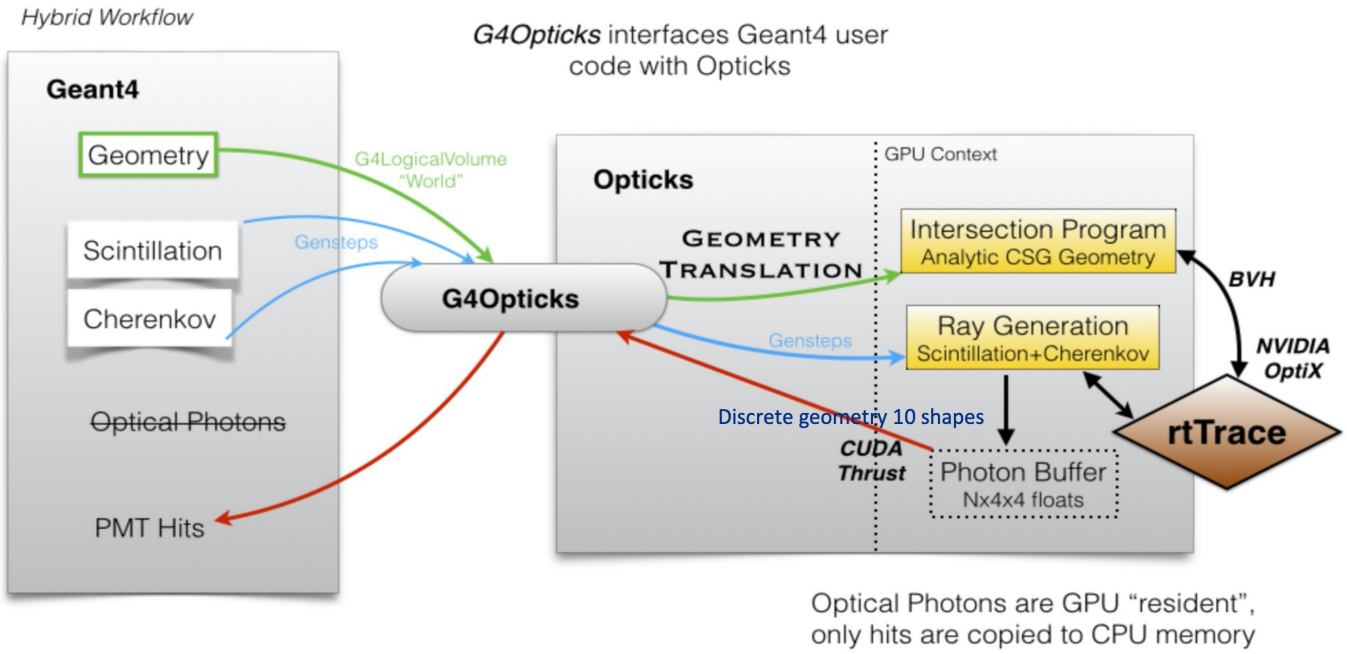
- Sequential, threading and tasking modes will work fine as they do now.
- To use sub-event parallelism (for Phase-I)
  - Use a newly introducing *RunManager* for sub-event parallelism
  - Implement *UserStackingAction* to sort tracks into sub-events
  - Implement merge() method in *G4Event* if special merging treatment is required
    - Ordinary HitCollection and HitVector (for scoring) will be automatically merged.
- To use sub-event parallelism (for Phase-II)
  - In addition to above
  - Physics list and/or detector construction dedicated to each task if needed
    - For an example with G4HepEM (designed to be compatible with GPUs)

Examples

Example 1: Opticks/OptiX + CaTS

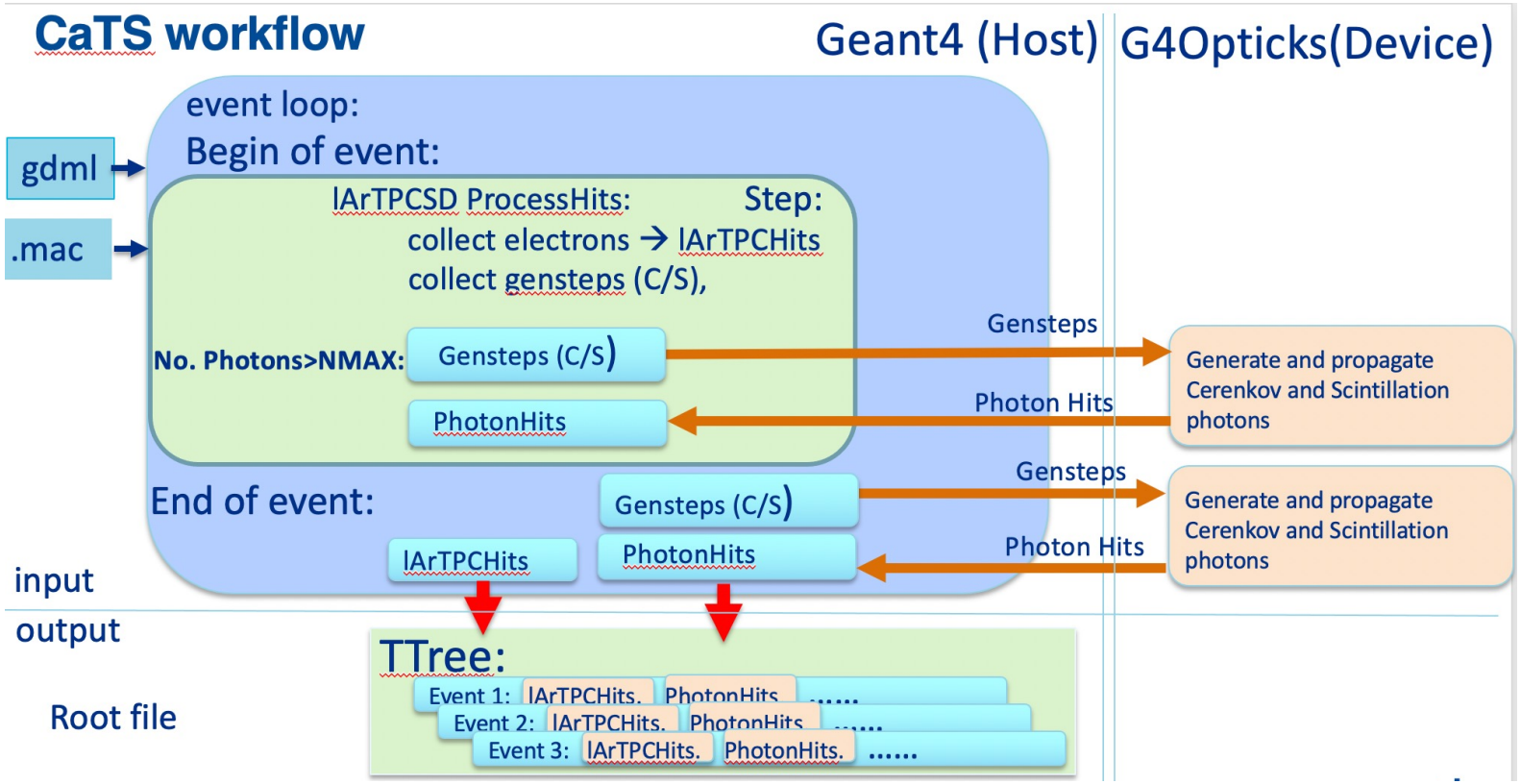
- Opticks is an open-source project that accelerates optical photon simulation by integrating NVIDIA GPU ray tracing, accessed via NVIDIA OptiX (developed by Simon Blyth for DayaBay and JUNO experiments, https://bitbucket.org/simoncblyth/opticks/).
- CaTS is an advance Geant4 example with Opticks/OptiX
- examples/advanced/CaTS

Figure from Simon's presentation



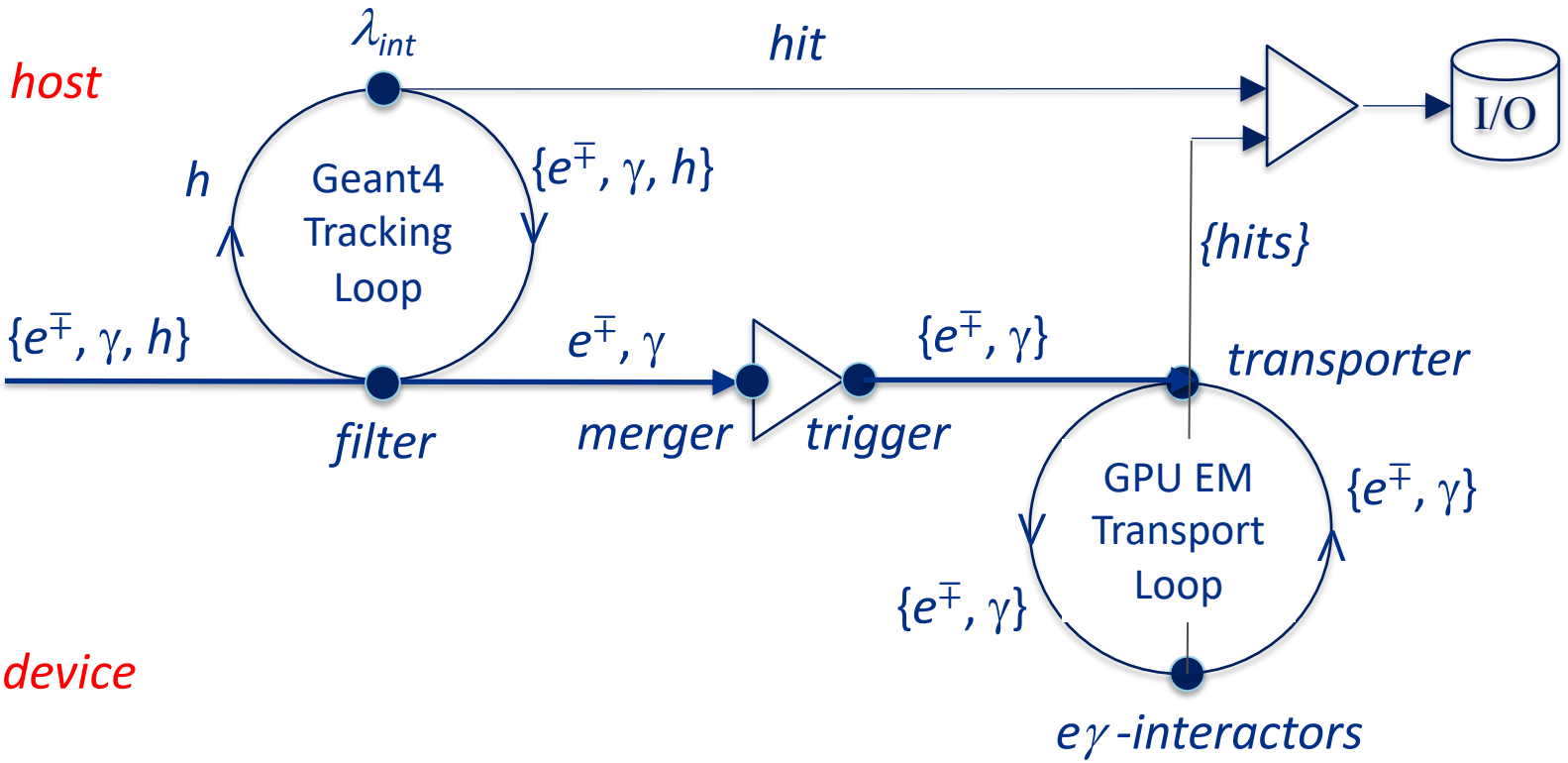
CaTS

- Uses Geant4 to collect Gensteps from Scintillation/Cerenkov. A Genstep contains all data to generate Cerenkov/Scintillation photons on GPUs
- Opticks generates optical photons on GPU and trace them using NVIDIA OptiX while Geant4 handles transportation of all other particles on CPU



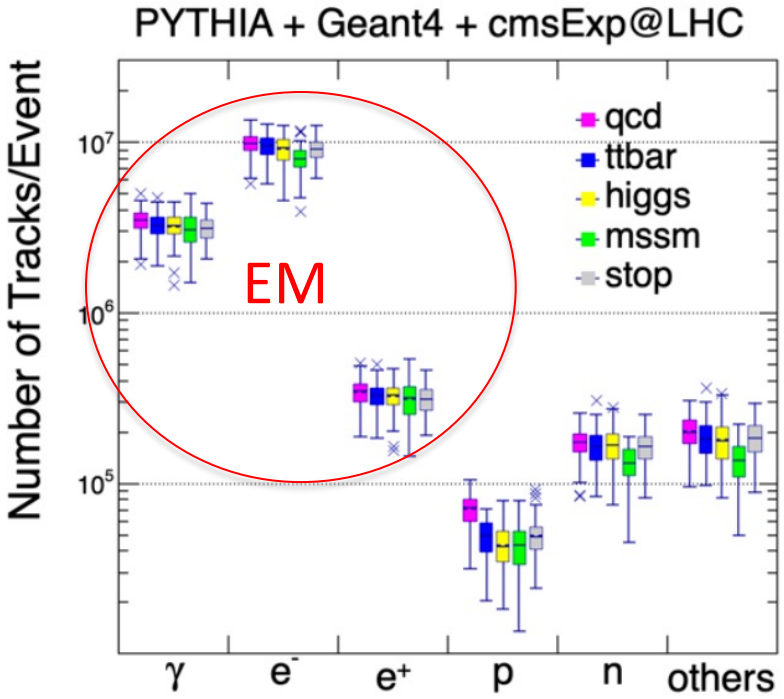
Example 2: EM Particle Transport on GPU

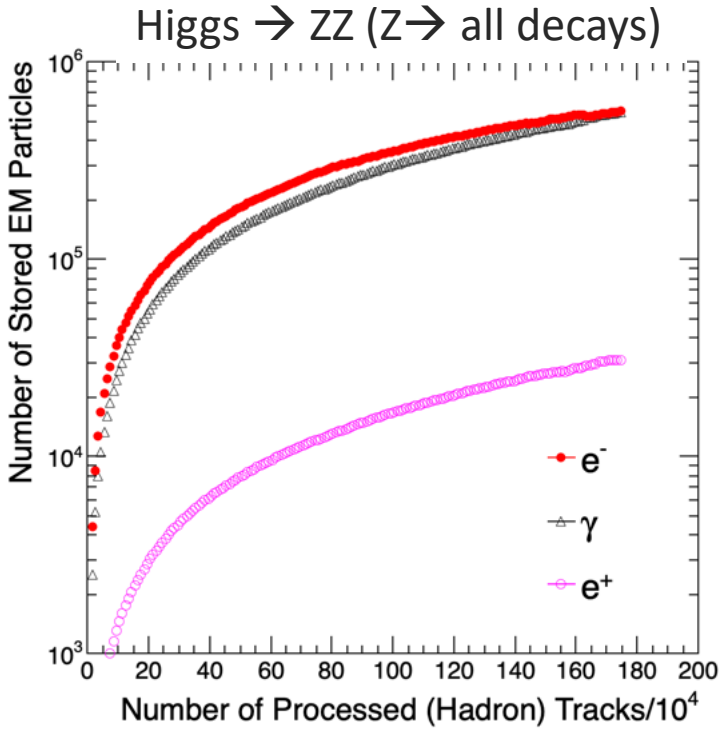
- A hybrid workflow with selected tasks executed on GPUs
  - Example: a Geant4 application which offloads EM particle transport on GPU using user actions.



Strategy for Offloading EM Simulation

- Use *G4UserTrackingAction* and *G4TaskRunManager*
  - Collect specific particles (ex. EM particles) for device tasks from *PreUserTrackingAction*
  - Execute tasks in *G4TaskGroup* with a chunk of stored tracks
- Number of EM tracks/event (cms2018@LHC, default Geant4 cut)





Code Example: Tracking Action

- *PreUserTrackingAction*
  - Select candidate tracks (e.g., $e^{\pm}$, γ) for offloading
  - Store tracks in a stack and trigger tasks within a same task group when the number of stored tracks reaches a pre-defined work unit (configurable)
  - Kill the Geant4 track from the Geant4 tracking loop

```
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);
    }
}
```

Code 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);
}
```

Code Example: Device Action

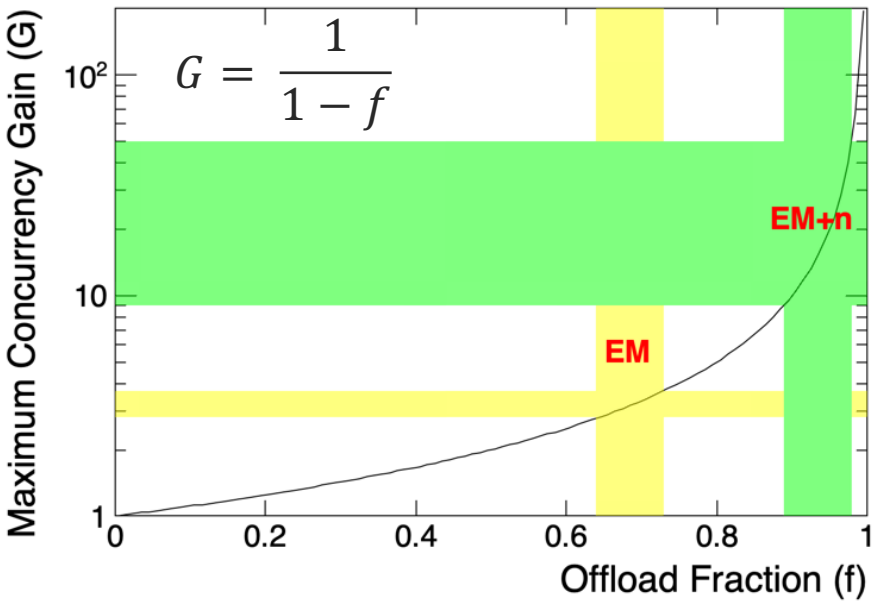
- Connectivity to an external software suitable for GPU

```
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 a set of device data {geometry, physics data, input stack of tracks}
  - Transfer sensitive hits from devices and merge them with Geant4 hits created on the host
  - Process left-over tracks at the end of event or run action (depending on how taskings are organized and scheduled)
  - Execute CPU task (currently event-level-multi-threading), device task, I/O task groups as concurrent 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 TimeGPU (f) ⩽ TimeCPU (1- f) with full concurrency (assuming no overhead)
- Example of offloading EM physics processes with cms2018@LHC



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, f is relatively low

Summary

- Advance software tools to use range of architectures to take advantages of increasing heterogeneity in compute resources
- Geant4 supports a task-based framework (*G4Tasking*) suitable for heterogeneous workflows
  - Event level task parallelism
  - Sub-event level parallelism
  - (Track-level parallelism)
- Examples of on-going HEP detection simulation projects using GPUs within the Geant4 R&D task force
  - Optical photon simulation (Opticks/Nivida OptiX) → CaTS
  - Offloading EM particle transport to (G4HepEM/Adept and Celeritas/Acceleritas)
- GPUs are everywhere even though there are many challenges for fully taking advantages of modern coprocessors for general detect simulation