Geant4 Sub-Event Parallelism

Makoto Asai (JLab/DCS)
asai@jlab.org

Geant4 evolutions in parallelization

1. Sequential mode : original since Geant4 v1.0
   - Single core (thread) does everything
2. Multithreaded event-level parallel mode : since Geant4 v10.0 (Dec.2013)
   - Taking the advantage of independence of events, many cores (threads) process events in parallel (event-level parallelism)
   - Geometry / x-section tables are shared over threads
3. Task-based event-level parallel mode : since Geant4 v11.0 (Dec.2021)
   - Decoupling task (event processing) from thread
   - More flexible load-balancing
4. Task-based sub-event parallel mode : since Geant4 v11.3 (Dec.2024)
   - Split an event into sub-events and task them separately
   - Sub-event : group of tracks that belong to one event
     - e.g. selected kind(s) of particle types, tracks getting into a particular detector component, etc.

- N.B. We made these evolutions without forcing the user to migrate
  - Except for using the new functionalities

Sub-event parallelism

- In the sub-event parallel mode, the concept of “run” does not exist in worker threads. Worker threads receive series of sub-events and process them. “Run” exists only in the master thread.
  - Only the master thread has *RunAction*
  - For each run (of the master), geometry and/or physics can be changed. Workers see these changes. Master does not proceed to the next run until workers complete all sub-events of the current run.
- In the same context, the beginning and end of “event” make sense only in the master thread.
  - Only the master thread has *EventAction* and *PrimaryGeneratorAction*.
  - The master thread may proceed to the next event without waiting workers to complete sub-events of the current event.
  - *EndOfEventAction()* of *EventAction* is invoked when all sub-events of the event have been completed by workers and results (hits, scores, trajectories) are merged back to the master thread.
  - Master thread takes care of sorting / merging results of workers to corresponding events.

Sub-event parallelism (continued)

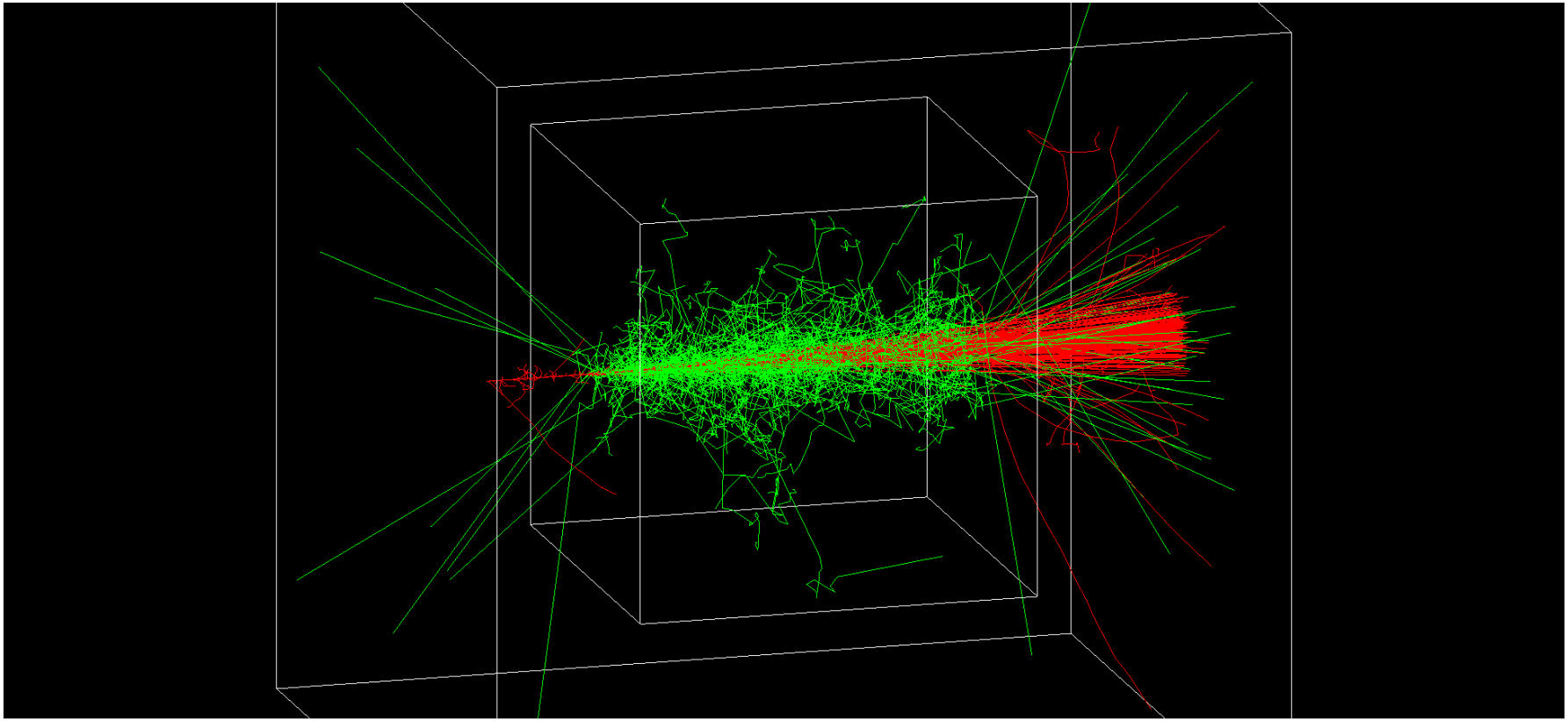
- *G4WorkerSubEvtRunManager* receives a sub-event and creates a local *G4Event* with tracks handed from the master.
  - *G4EventManager* of the worker thread processes this local *G4Event*.
  - *G4TrackingManager*, *G4SteppingManager*, sensitive detectors, scorers of the worker thread work exactly the same as the ones in MT mode. No migration required.
  - The local *G4Event* with hits and (optional) trajectories is sent to the master, so that these hits and trajectories are merged into the master *G4Event*.
- *TrackingAction* and *SteppingAction* make sense in worker threads. You may set them also in the master thread.
  - Note that primary event is generated in the master thread, and tracks that are not sent to worker threads are processed in the master thread.
  - Also note that you can implement several different *TrackingAction* / *SteppingAction* classes specialized for each sub-event types or specialized for master.

Use-cases of sub-event parallel mode

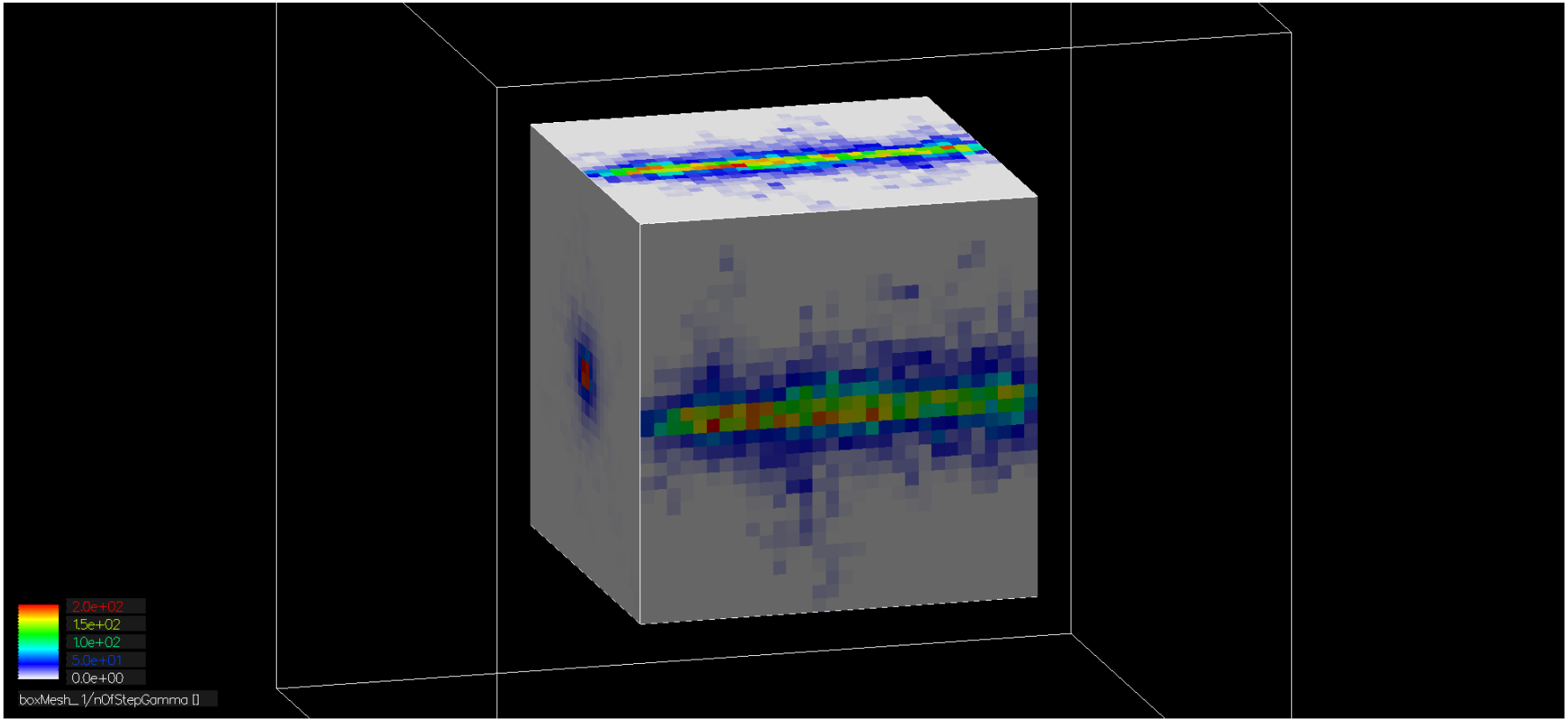
- Without GPUs
  - If processing of one event takes too long, you can boost the simulation by factor of N(threads).
  - CDMS (phonon), Air shower, DNA, …
  - You won't gain if number of events you are simulating is much larger than number of threads you have. Event parallelism is faster / more efficient.
- With GPUs
  - A sub-event is tasked to a worker that is dedicated to a GPU code.
  - Different kinds of GPU codes can be assigned to different workers.
    - GPU code for dedicated physics, or optimized to particular detector component (e.g. barrel, forward and backward calorimeters)
  - Master thread takes care of stacking tracks and sending them as sub-events to responsible workers.
  - All tracks of a sub-event belong to the same event. Whilst there may exist sub-events of more than one events at a given time.
  - A worker may be fed more than one sub-events to make GPU busy.

Event level parallel mode

Shooting muons to a water box
Red : negative particle, green : neutral particle

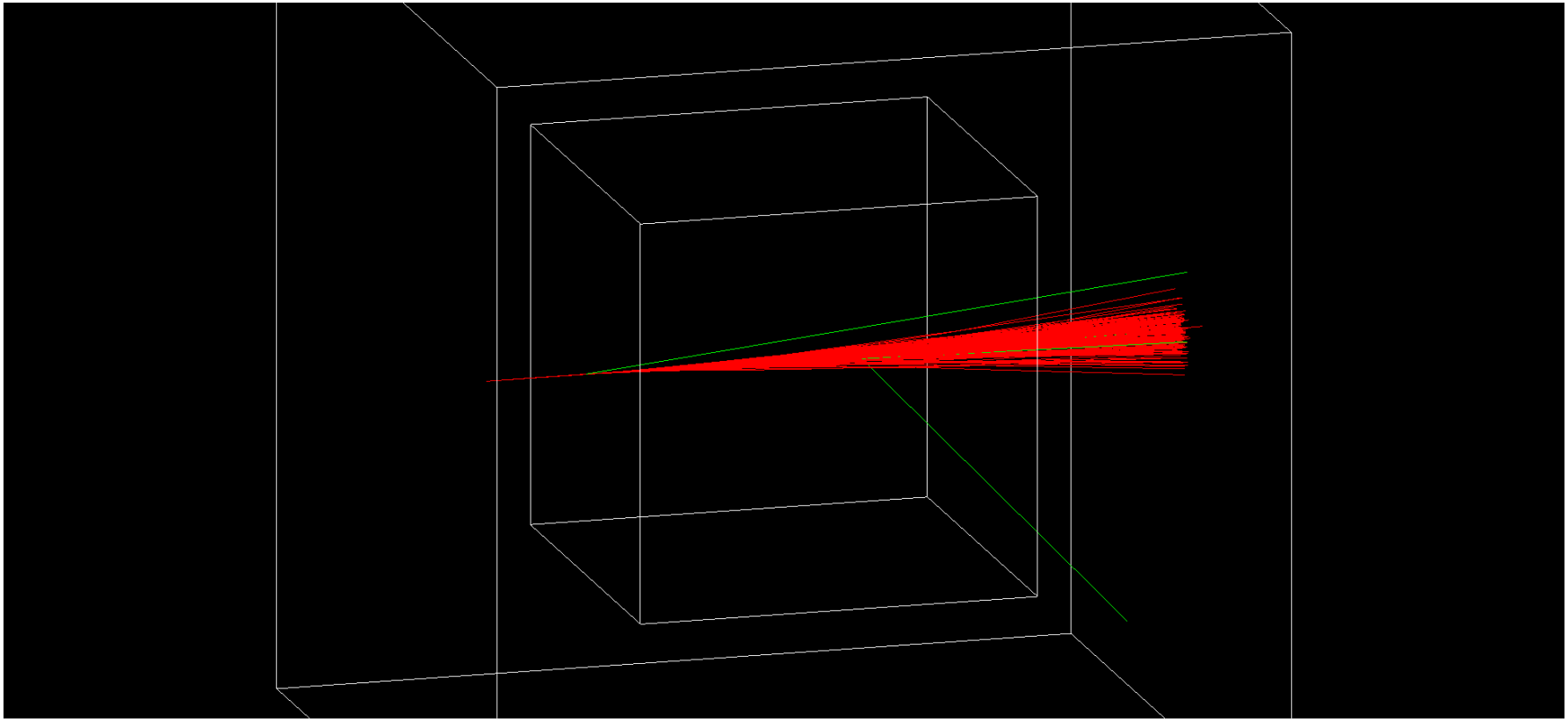


Event level parallel mode



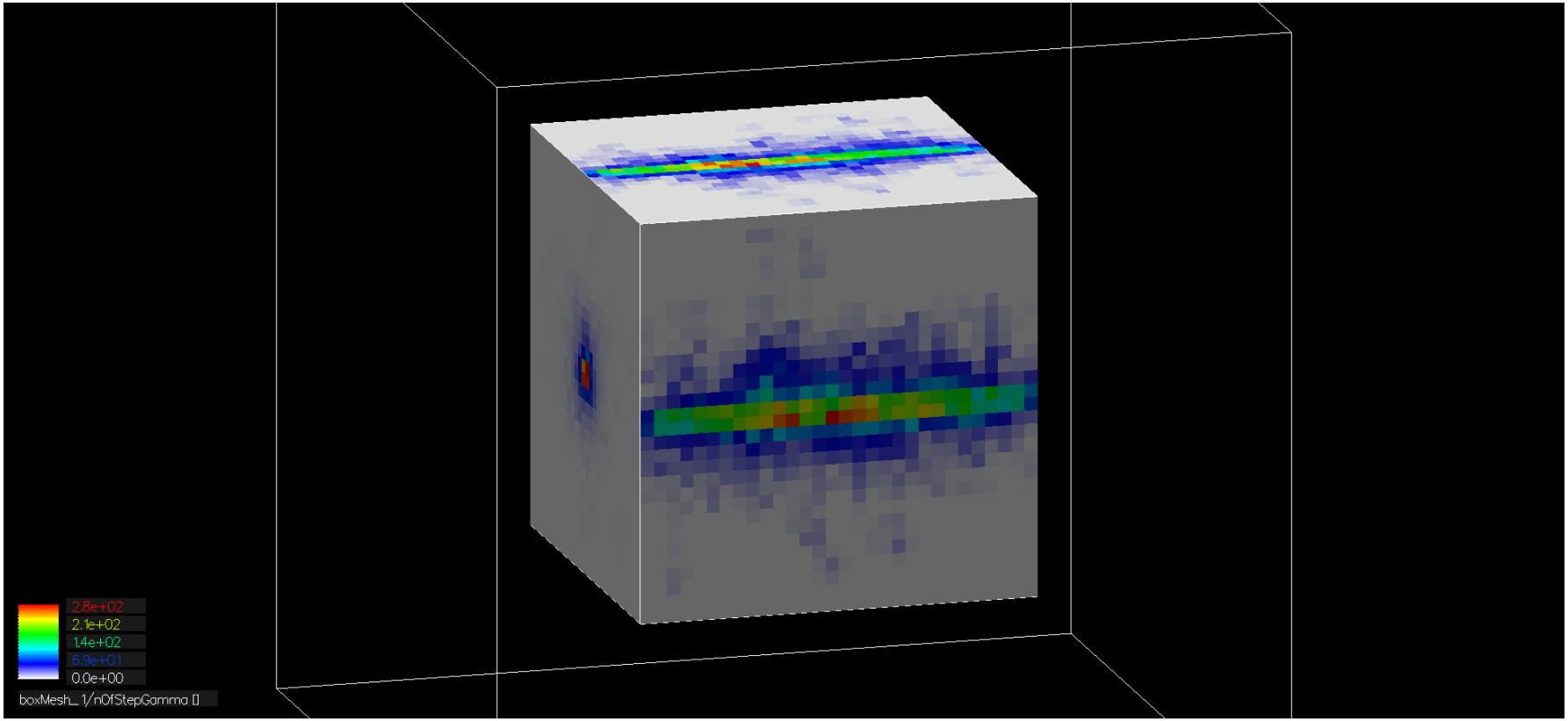
Sub-event level parallel mode

Sending e±, gamma to worker threads
Trajectories of worker threads are not visualized



Sub-event level parallel mode

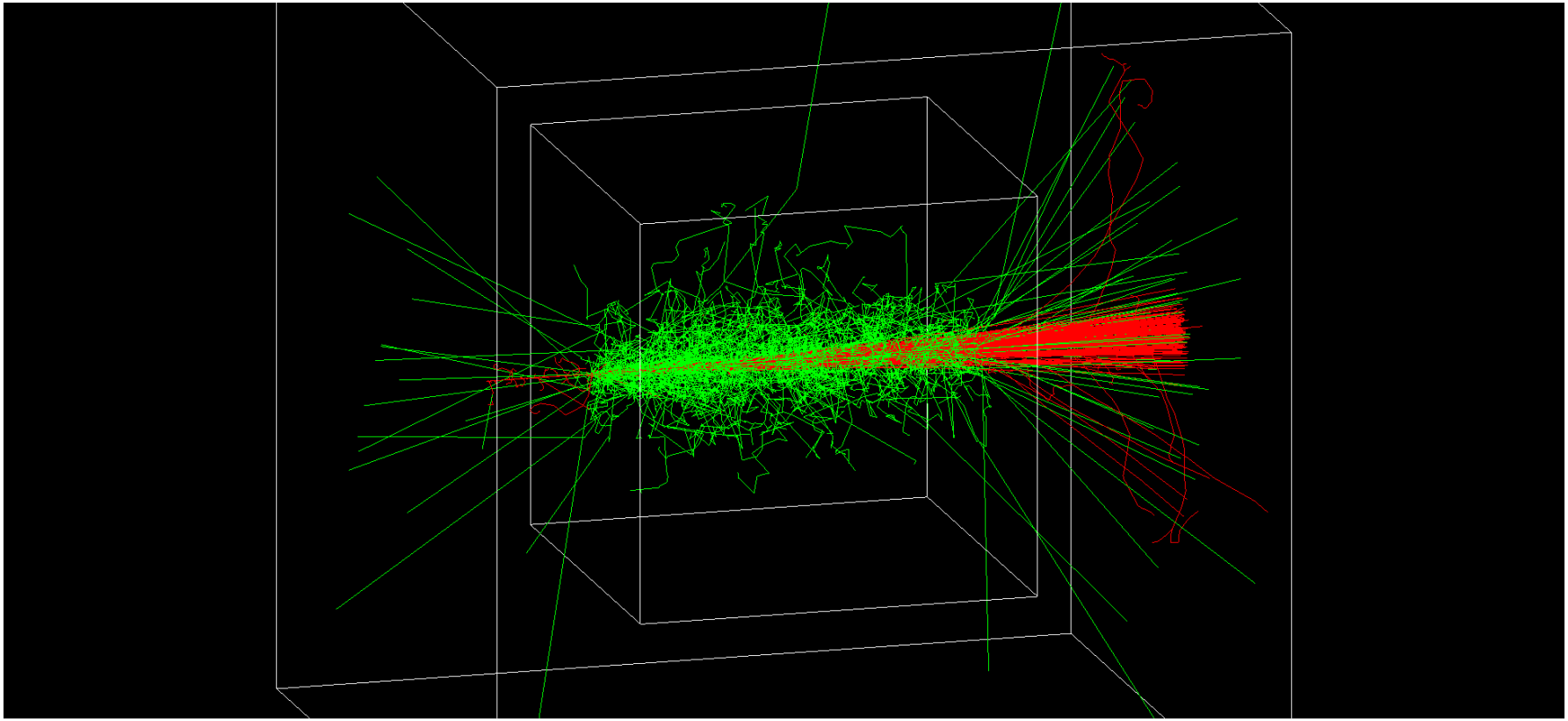
Scores of the worker threads are automatically merged.



Jefferson Lab

Sub-event level parallel mode

/run/trajectoriesToBeMerged true



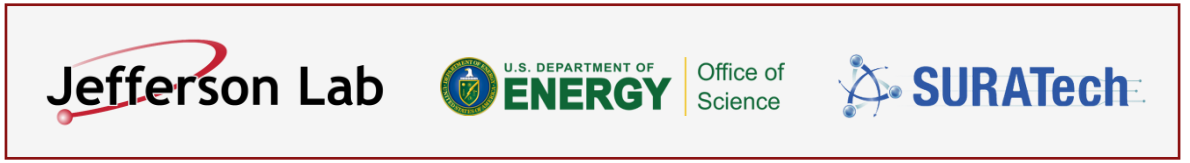
User-classes for sub-event parallelism

- *main()*
  - Use *G4SubEvtRunManager*
  - Or set *G4RUN_MANAGER_TYPE* environment variable to “SubEvt” if you use *G4RunManagerFactory*
- *UserSubEvtThreadInitialization*
  - For more than one sub-event types, instantiate *G4WorkerSubEvtRunManager* with sub-event type id.
  - Set *UserSubEvtThreadInitialization* to *G4SubEvtRunManager*.
  - If you have only one sub-event type, you don’t need this class.
- *ActionInitialization::BuildForMaster()*
  - Register sub-event type(s) with the maximum number of tracks to be packed in a sub-event.
  - *PrimaryGeneratorAction* must be set for the master thread rather than worker threads.
- *UserStackingAction* of the master thread
  - Classify a track with sub-event type.

Status and plan

- Tracks are sent (i.e. copied) unidirectional from master to worker. Hits/scores are sent from worker to master. Trajectories are optionally sent from worker to master.
- Passing tracks between worker threads is not practical, as worker tasks are running asynchronously so that we need to introduce complex Mutex barriers that would destroy performance.
  - For the time being, tracks that cannot be handled by GPU should be processed by the CPU of the corresponding worker thread.
  - If necessary, tracks must be transferred to the other worker through the master. Sending tracks back from worker to master may be considered in the future, if needs are.
- Currently (version 11.4) *examples/extended/runAndEvent/RE03* demonstrates some limited features of the sub-event parallelism.
  - Extending *RE03* as well as introducing new example(s) are planned for the version 11.5 release in December. Documents will be added.

Backup slides



Object ownership

- Passing *G4Track* between master and worker is a deep copy, as *G4Track* object is instantiated by thread-local *G4Allocator* and thus cannot be handed over to other thread. Touchable cannot be passed by the same reason.
- Scores of the worker threads are automatically merged to the master.
- User-defined hit objects created in a worker thread can be referred to sum-up or deep-copied in the master.
  - *Clone()* method required for deep-copying
- Trajectories created in the worker thread are optionally deep-copied.
  - In sub-event parallel mode, trajectories created in worker threads are not merged to the event in the master thread by default due to the performance overhead caused by copying them.
  - */run/trajectoriesToBeMerged* UI command enables the merging.

ActionInitialization

```
void ActionInitialization::BuildForMaster() const {
  if(G4RunManager::GetRunManager()->GetRunManagerType()
                          == G4RunManager::subEventMasterRM) {
    G4RunManager::GetRunManager()->RegisterSubEventType(0,1000);
    G4RunManager::GetRunManager()->SetDefaultClassification(
                          G4Electron::Definition(),fSubEvent_0);
    SetUserAction(new PrimaryGeneratorAction);
    SetUserAction(new StackingActionForMaster);
  }
}
void ActionInitialization::Build() const {
  if(G4RunManager::GetRunManager()->GetRunManagerType()
                           != G4RunManager::subEventWorkerRM)
  { SetUserAction(new PrimaryGeneratorAction); }
}
```