Session 3B
Near-term Kernel Developments

Makoto Asai (SLAC)
Gabriele Cosmo (CERN)

Session Topics

- Discuss and review the current status and work plan of the near-term development items, that are planned to be released with 10.5 or in the next couple of years.

Sub-event parallelism	*Makoto Asai*	14:00 - 14:12
Unified tracking mechanism for exotic particles (ions, muonic atom...	*Makoto Asai et al.*	
Revisit / retreat production thresholds and physics process framework	*Marc Verderi*	
Refactoring transportation	*Makoto Asai et al.*	14:36 - 14:48
Use of HPC	*Jonathan Madsen*	14:48 - 15:00
Extending crystal structure capabilities	*Enrico Bagli*	15:00 - 15:12
TiMemory performance monitoring	*Jonathan Madsen*	15:12 - 15:24
Open discussion		15:24 - 15:30

Sub-event parallelism

- Sub-event parallelism generalizes Geant4-MT event parallelism approach to serve the case of applications requesting large memory per event.
- One event is split into “sub-events”, e.g. each few primary tracks = a sub-event
  - Split method is obviously user dependent.
- Each sub-event is sent to a worker thread, and merged back to the original full event later.
- Constraint – all the current API’s must be preserved.
- Time scale of the development :
  - Source code : to be finished by May 2019
  - An example : to be finished by September 2019

Unified tracking mechanism for exotic particles (ions, muonic atoms, radicals, hyper-nuclear, phonon, e/h)

- There are at least four different particle families that share the same mechanism of G4GenericIon and shared G4ProcessManager.
  - Ions, muonic ions, Hyper-nucleous, radicals
- Uniform, transparent and extendable treatment is required.
- G4SteppingManager is already cleaned up. There is no longer #ifdef, there is no longer indirect access to G4PrcessManager via G4GenericIon (or G4GenericMuonicAtom, etc.).
  - G4IonTable is updated to ensure this.
- Remaining work – Split G4IonTable class
  - G4IonTable::GetMuonicAtom() will be moved to new G4MuonicAtomTable.
  - Hyper-nucleon, radicals, will have their own “table” class.
  - Clean way to enable extendibility for yet another particle family without overhead
  - Plan to finish for ion and muonic atom by 10.5 release, while hyper-nucleon and radical tables will be in the coming year.

Revisit / retreat production thresholds and physics process framework

- Production thresholds ("cuts") initially considered as an issue fundamental enough to be taken care at the kernel class level.

Particle produced	Production process	Motivation
$e^{\uparrow}-$	Ionization	Heavy production (limited by energy binding to atoms). These are actually "recoil electrons". Threshold needed to limit the production.
$e^{\uparrow}+$	Conversion	No divergence nor heavy production. Use case : production cut in mountain rock for, e.g., dark matter experiments.
γ	Bremsstrahlung	Cross-section divergence (actually limited by dielectric effects at very low energies). Threshold needed to limit the production.
p (ions)	Hadron elastic	Threshold on recoil proton, e.g. n scattering on proton, ejecting it. Mechanism adapted for ions. Threshold defines the "visibility" cut.

Proposal

- Kernel classes are offloaded from cuts control
  - Including control at tracking time
  - Classes involved: `G4ParticleDefinition, G4SteppingManager`
- Processes are given the full responsibility to manage their production thresholds
  - Whatever if this is due to divergences or not
  - With possibly, and preferably, common tools, shared among packages, to expose the configuration to the user
- The machinery for material-cut couple becomes extendable:
  - It has the set {, and } by default
  - But is extendable to any other type of particle
- Dedicated tests are added to check for conformance of secondary production
    - A test using a simple user stepping action could do it
- Backward compatibility should be considered as well
  - At least for some time

Refactoring transportation

- Currently, only one transportation object exists in the memory:
  - Either G4Transportation, G4CoupledTransportation or G4ITTransportation
  - It deals with all particle types:
    - neutral and charged particles, optical photons, phonons, etc
- Idea is to provide at least two flavors of transportation that co-exist:
  - One for charged particles, one for neutral particles
  - Eventually one also for optical photons
    - As velocity calculations differ from other particles
- Further extensions/specializations to be also considered:
  - VecGeom navigation: optimized/vectorized, implementation with modern C++
  - À la DagMC: direct and efficient navigation in CAD geometries
  - DNA navigation: better serve the case of radicals
- Revision will be extended to “Coupled Transportation”:
- Study in progress. Plan to deliver first implementations in 2019 as an option in G4VModularPhysicsList::AddTransportation().

Use of HPC – short-term developments

- Existing supercomputers
  - Cori (32-core/node Haswell @ 2.3 GHz + 68-core/node KNL @ 1.4 GHz)
  - Edison (24-core/node Ivy-Bridge @ 2.4 GHz)
- Upcoming supercomputer
  - NERSC 9 supercomputer (2019)
  - Accelerators
    - Not next-gen KNL => no AVX-512 instruction set

- Create "official" Geant4 docker repository
  - *e.g.*, https://hub.docker.com/r/geant4
  - Tags for compilers, versions, etc.
  - Use this as standard for per-machine performance comparison
    - No virtualization layer
    - Portable to any HPC/cluster capable of running Docker containers
    - Exact same compiler, configuration, etc. ⇒ hardware-vs-hardware
  - Reproducibility
    - (A) associate Dockerfile with Git revision or (B) `docker save ...`
- Performance characterization with respect to:
  - HPC system
    - Cori KNL
    - Cori Haswell
    - Edison
    - Any others?
  - Compiler
    - GNU
    - Intel
    - Clang
    - Any others?
  - OS (*i.e.*, kernel)
    - openSUSE
    - RHEL
    - Any others?

- Created a public CDash dashboard at NERSC: cdash.nersc.gov
  - Built with Docker and deployed in Spin – a containers-as-service (CaaS) platform at NERSC
- Use Geant4 + TiMemory + CDash for performance characterization
  - Nightly and continuous performance testing
  - Separate from our build testing – avoid adding more clutter
  - Longer-term additions:
    - Valgrind reports
    - Code coverage
    - VTune reports (*e.g.*, attach VTune summary as CTest note)

Extending Crystal Structure Capabilities

- Geant4 crystal structures implemented and working:
  - Very flexible structure thanks to the G4ExtendedMaterial class, updates can be done without any change to existing G4 classes.
- Store the lattice parameters (unit cell) and elasticity tensors (used by phonon)
  - Capability of compute structure factor and other properties (can be updated with more function depending on the future needs)
- Currently used by the G4Channeling process:
  - The process get the crystal lattice orientation from the G4LogicalCrystalVolume and the evaluated data from the G4ChannelingData (subclass of G4MaterialExtension)
- Future updates are related to the needs of the processes
  - Migration of Phonon processes
  - Coherent X-ray scattering, Diffraction
- Support for powders/polycrystals?

TiMemory Performance Monitoring

- TiMemory is a lightweight “in-situ” package for consistent and automated performance reports.
  - Supplements profilers – profilers have overhead and that overhead skews/misrepresents run-time
- Cross-language ⇒ C, C++ and Python implementations can be used independently or simultaneous
- Built-in features (via Python)
  - Timing and memory plotting
  - Support for attaching analysis as CTEST_NOTES and attaching images as <DartMeasurementFile> to CDash dashboard

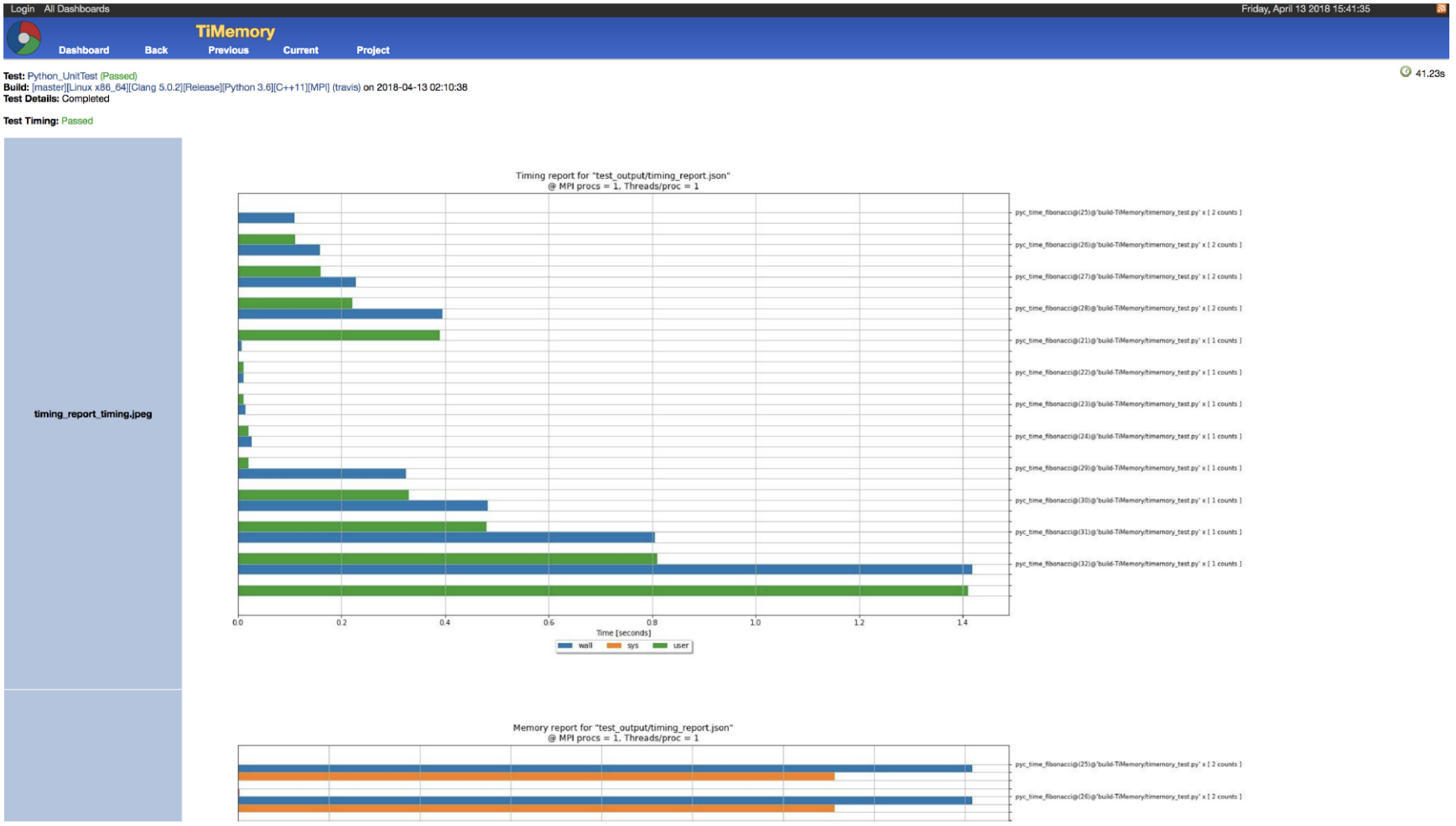

Figure 1: Plot of timing/memory in CDash

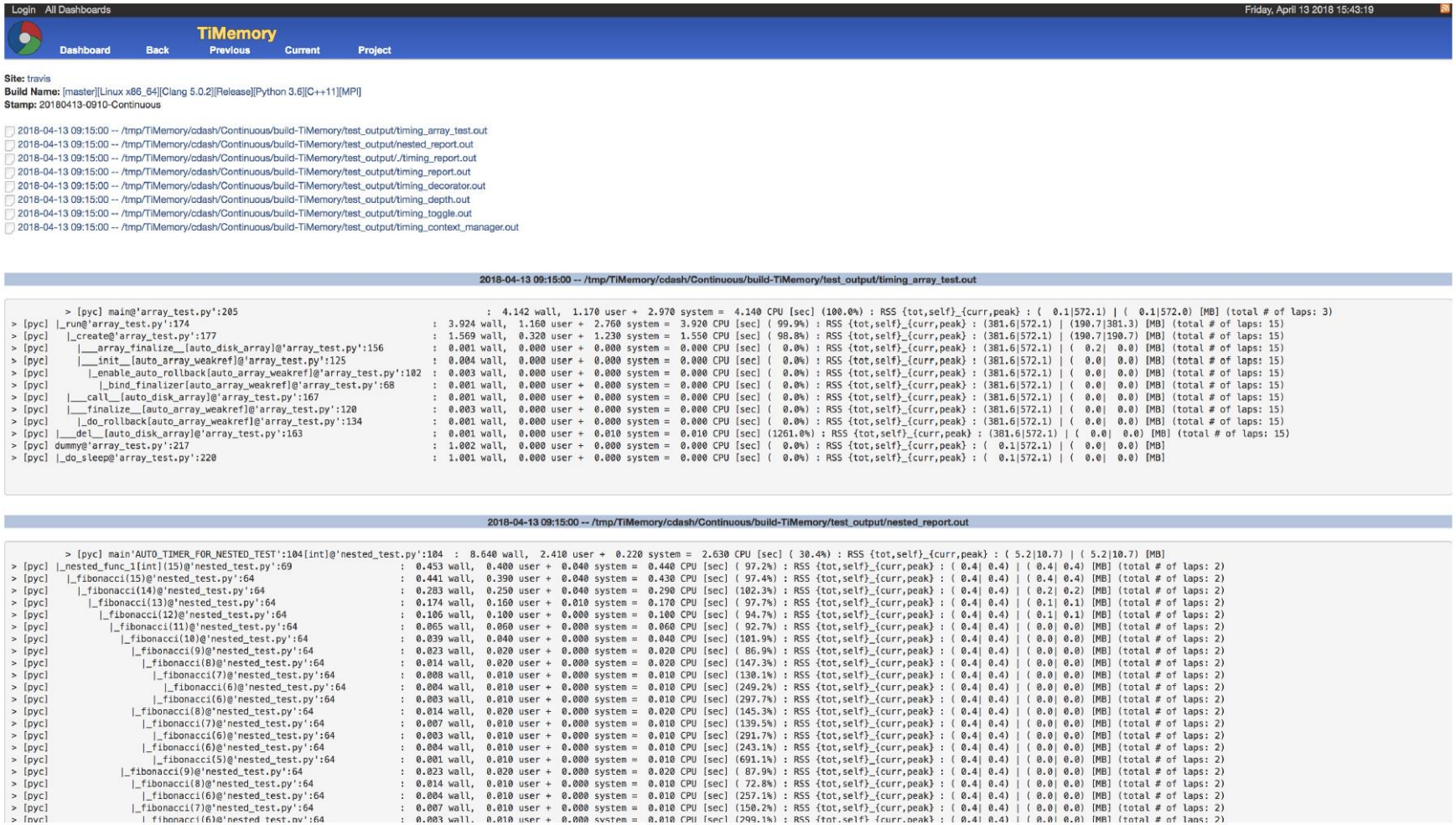

Figure 2: ASCII output of timing + memory in CDash