#### 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 nearterm development items, that are planned to be released with 10.5 or in the next couple of years.

Sub-event parallelism	Makoto Asai 0	8
	14:00 - 14:12	2
Unified tracking mechanism for exotic particles (ions, muonic atom	Makoto Asai et al. 🤞	2
Revisit / retreat production thresholds physics process framework	s and Marc Verderi 🥝	2
Refactoring transportation	Makoto Asai et al.	0
	14:36 - 14:48	3
Use of HPC	Jonathan Madsen	0
	14:48 - 15:00	C
Extending crystal structure capabilitie	es Enrico Bagli 🥝	9
	15:00 - 15:12	2
TiMemory performance monitoring	Jonathan Madsen	8
	15:12 - 15:24	4
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
eî—	Ionization	Heavy production (limited by energy binding to atoms). These are actually "recoil electrons". Threshold needed to limit the production.
eî+	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 + 68core/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
  - o e.g., https://hub.docker.com/r/geant4
  - Tags for compilers, versions, etc.
  - $\circ~$  Use this as standard for per-machine performance comparison
    - ▷ No virtualization layer
    - Portable to any HPC/cluster capable of running Docker containers
    - $\triangleright$  Exact same compiler, configuration, etc.  $\Rightarrow$  hardware-vs-hardware
  - Reproducibility
    - $\triangleright$  (A) associate Dockerfile with Git revision or (B) docker save ...
- Performance characterization with respect to:
  - HPC system Compiler  $\circ$  OS (*i.e.*, kernel) 0 Cori KNL ▷ GNU ▷ openSUSE Cori Haswell ▷ Intel  $\triangleright$ ▷ RHEL Edison ▷ Clang ▷ Any others?  $\triangleright$ Any others? ▷ 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

TiMemory + CDashPlotting



Figure 1: Plot of timing/memory in CDash





ASCII

Login All Dashboards											Friday, /	April 13 2018 15	:43:19	5
Dashboard	TiMemo Back Previous	Y Current	Project											
Site: travis Build Name: [master][Linu Stamp: 20180413-0910-C	x86_64][Clang 5.0.2][Release][P intinuous	/thon 3.6][C++11][i	MP[]											
2018-04-13 09:15:00 - 2018-04-13 09:15:00 -	tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou tmp/TiMemory/cdash/Continuou	s/build-TiMemory/ s/build-TiMemory/ s/build-TiMemory/ s/build-TiMemory/ s/build-TiMemory/ s/build-TiMemory/ is/build-TiMemory/ is/build-TiMemory/	test_output/timing_erray_test.out test_output/nested_report.out test_output/timing_report.out test_output/timing_report.out test_output/timing_depth.out test_output/timing_depth.out test_output/timing_toggle.out /test_output/timing_toggle.out	1										
			2018-04-	13 09:15:00 /tmp/TiMemory/cda	dash/Continuous/build-T	"iMemory/test_outpu	ut/timing_array_t	est.out						
> [pyc] ma > [pyc]  _run@'array	.n@'array_test.py':205 _test.py':174		: 4.1 : 3.924 wall,	42 wall, 1.170 user + 2.970 1.160 user + 2.760 system =	70 system = 4.140 CP = 3.920 CPU [sec] (	U [sec] (100.0%) 99.9%) : RSS {tot	: RSS {tot,sel ,self}_{curr,p	f}_{curr,peak} : ( eak} : (381.6 572	0.1 572.1)   1)   (190.7 381	( 0.1 572.0) [ .3) [MB] (total	[MB] (total # o l # of laps: 15	of laps: 3) 5)		

<pre>&gt; [pyc]  _create@'array_test.py':177</pre>	: 1.569 wall,	0.320 user + 1.230 system =	1.550 CPU [sec] ( 98.8%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   (190.7 190.7) [MB] (total # of laps: 15)
> [pyc]  array_finalize_[auto_disk_array]@'array_test.py':156	: 0.001 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.2  0.0) [MB] (total # of laps: 15)
> [pyc]init[auto_array_weakref]@'array_test.py':125	: 0.004 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  _enable_auto_rollback[auto_array_weakref]@'array_test.py':102</pre>	: 0.003 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  _bind_finalizer[auto_array_weakref]@'array_test.py':68</pre>	: 0.001 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  call[auto_disk_array]@'array_test.py':167</pre>	: 0.001 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  finalize[auto_array_weakref]@'array_test.py':120</pre>	: 0.003 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  _do_rollback[auto_array_weakref]@'array_test.py':134</pre>	: 0.001 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc]  del[auto_disk_array]@'array_test.py':163</pre>	: 0.001 wall,	0.000 user + 0.010 system =	0.010 CPU [sec] (1261.0%) : RSS {tot,self}_{curr,peak} : (381.6 572.1)   ( 0.0  0.0) [MB] (total # of laps: 15)
<pre>&gt; [pyc] dummy@'array_test.py':217</pre>	: 1.002 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : ( 0.1 572.1)   ( 0.0  0.0) [MB]
<pre>&gt; [pyc]  _do_sleep@'array_test.py':220</pre>	: 1.001 wall,	0.000 user + 0.000 system =	0.000 CPU [sec] ( 0.0%) : RSS {tot,self}_{curr,peak} : ( 0.1 572.1)   ( 0.0  0.0) [MB]

#### 2018-04-13 09:15:00 -- /tmp/TiMemory/cdash/Continuous/build-TiMemory/test\_output/nested\_report.out

	> [pyc] main'AUTO_TIMER_FOR_NESTED_TEST':104[int]@'nested	test.py':	104 : 8.	640 wall,	2.410 use	r + 0.220	system = 2.630	OCPU [sec]	30.4%) : RSS {tot,self}_{curr,peak} : ( )	.2 10.7)   ( 5.2 10.7) [MB]	
> [pyc]	_nested_func_1[int](15)@'nested_test.py':69	: 0.	453 wall,	0.400 use	r + 0.040	system =	0.440 CPU [sec]	( 97.2%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.4  0.4) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>[_fibonacci(15)@'nested_test.py':64</pre>	: 0.	441 wall,	0.390 use	r + 0.040	system =	0.430 CPU [sec]	(97.4%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.4  0.4) [MB] (total # of lar	ups: 2)
> [pyc]	[_fibonacci(14)@'nested_test.py':64	: 0.	283 wall,	0.250 use	r + 0.040	system =	0.290 CPU [sec]	(102.3%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.2  0.2) [MB] (total # of la	ips: 2)
> [pyc]	<pre>[_fibonacci(13)@'nested_test.py':64</pre>	: 0.	174 wall,	0.160 use	r + 0.010	system =	0.170 CPU [sec]	(97.7%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.1  0.1) [MB] (total # of lar	ips: 2)
> [pyc]	[_fibonacci(12)@'nested_test.py':64	: 0.	106 wall,	0.100 use	r + 0.000	system =	0.100 CPU [sec]	(94.7%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.1  0.1) [MB] (total # of la	ups: 2)
> [pyc]	<pre>fibonacci(11)@'nested_test.py':64</pre>	: 0.	065 wall,	0.060 use	r + 0.000	system =	0.060 CPU [sec]	(92.7%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>[_fibonacci(10)@'nested_test.py':64</pre>	: 0.	039 wall,	0.040 use	r + 0.000	system =	0.040 CPU [sec]	(101.9%)	RSS {tot, self}_{curr, peak} : ( 0.4  0.4)	( 0.0   0.0) [MB] (total # of la	ups: 2)
> [pyc]	[_fibonacci(9)@'nested_test.py':64	: 0.	023 wall,	0.020 use	r + 0.000	system =	0.020 CPU [sec]	(86.9%)	RSS {tot, self}_{curr, peak} : ( 0.4  0.4)	( 0.0   0.0) [MB] (total # of la	ips: 2)
> [pyc]	<pre>fibonacci(8)@'nested_test.py':64</pre>	: 0.	014 wall,	0.020 use	r + 0.000	system =	0.020 CPU [sec]	(147.3%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>[_fibonacci(7)@'nested_test.py':64</pre>	: 0.	008 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(130.1%)	RSS {tot, self}_{curr, peak} : ( 0.4  0.4)	( 0.0   0.0) [MB] (total # of la	ips: 2)
> [pyc]	<pre>fibonacci(6)@'nested_test.py':64</pre>	: 0.	004 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(249.2%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>fibonacci(6)@'nested_test.py':64</pre>	: 0.	003 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(297.7%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ups: 2)
> [pyc]	<pre>[_fibonacci(8)@'nested_test.py':64</pre>	: 0.	014 wall,	0.020 use	r + 0.000	system =	0.020 CPU [sec]	(145.3%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	fibonacci(7)@'nested_test.py':64	: 0.	007 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(139.5%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>fibonacci(6)@'nested_test.py':64</pre>	: 0.	003 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(291.7%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of la	ips: 2)
> [pyc]	_fibonacci(6)@'nested_test.py':64	: 0.	004 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(243.1%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ups: 2)
> [pyc]	<pre>fibonacci(5)@'nested_test.py':64</pre>	: 0.	001 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(691.1%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0   0.0) [MB] (total # of la	ups: 2)
> [pyc]	<pre>[_fibonacci(9)@'nested_test.py':64</pre>	: 0.	023 wall,	0.020 use	r + 0.000	system =	0.020 CPU [sec]	(87.9%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	<pre>fibonacci(8)@'nested_test.py':64</pre>	: 0.	014 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	( 72.8%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of lar	ips: 2)
> [pyc]	_fibonacci(6)@'nested_test.py':64	: 0.	004 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(257.1%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of la	ups: 2)
> [pyc]	[_fibonacci(7)@'nested_test.py':64	: 0.	007 wall,	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(150.2%)	RSS {tot,self}_{curr,peak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of la	ips: 2)
> [ovc]	<pre>1 fibonacci(6)@'nested test.ov':64</pre>	: 0.	003 wall.	0.010 use	r + 0.000	system =	0.010 CPU [sec]	(299.1%)	RSS {tot.self} {curr.neak} : ( 0.4  0.4)	( 0.0  0.0) [MB] (total # of la	ins: 2)

#### Figure 2: ASCII output of timing + memory in CDash

