FastCaloSim-GPU

HSF Simulation Working Group Meeting
R&D on accelerators and simulations

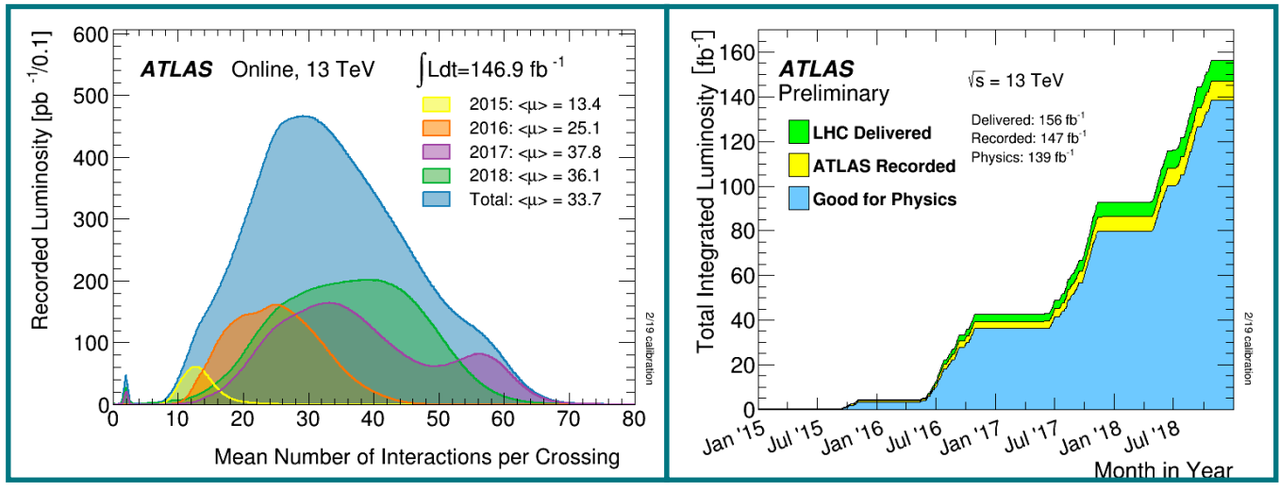
Vincent R. Pascuzzi
June 24, 2020

U.S. DEPARTMENT OF ENERGY

BERKELEY LAB

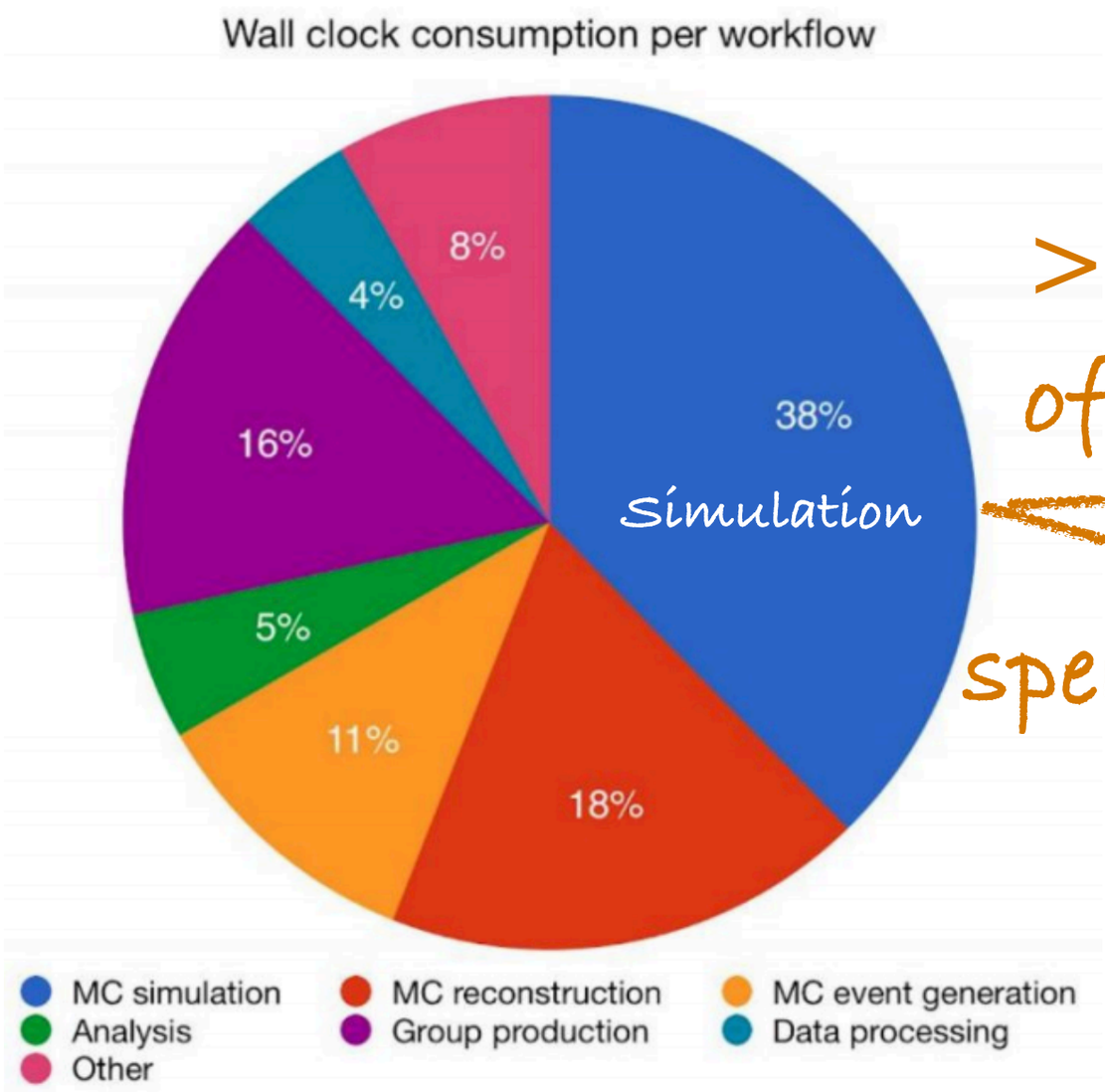
Motivation

In a nutshell



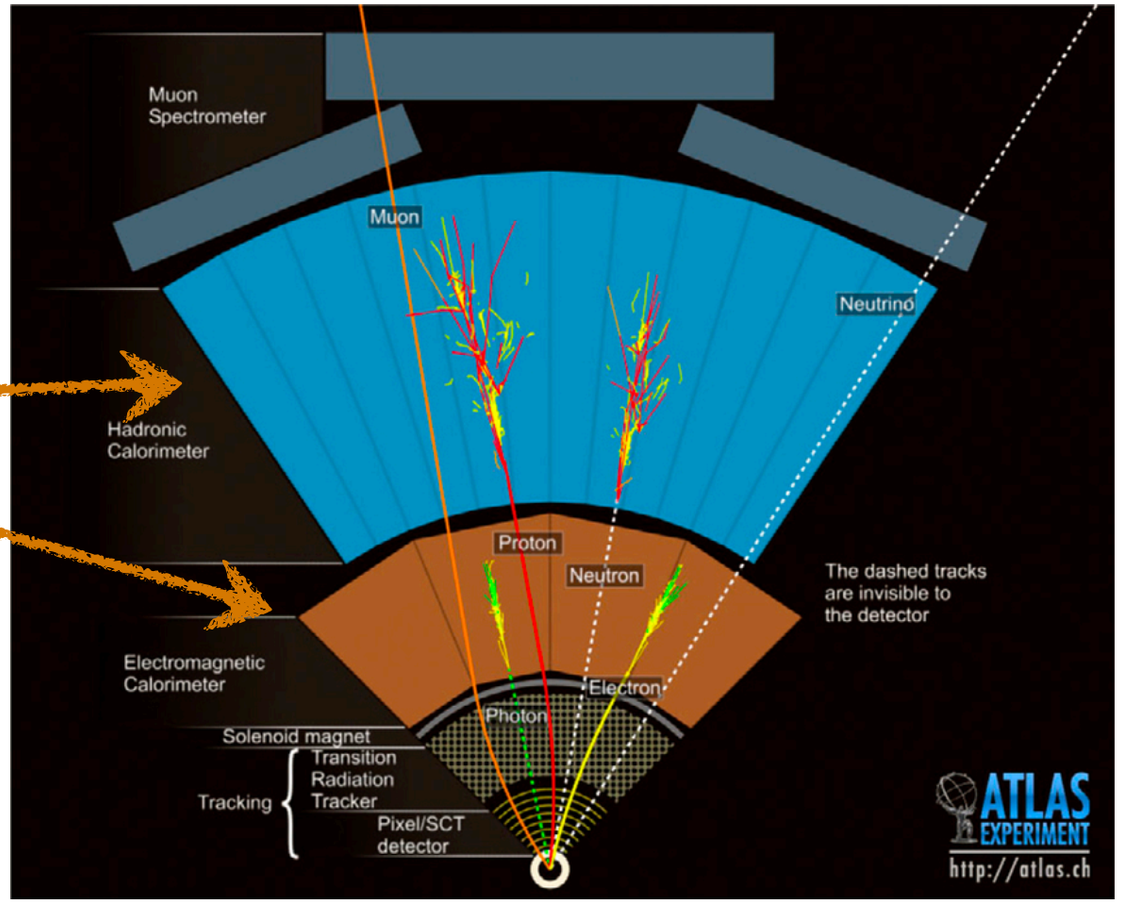
AtlasPublic/LuminosityPublicResultsRun2

$$\sqrt{\frac{\frac{1}{N_{\mathrm{MC}}}}{\frac{1}{\sigma \int \mathcal{L} dt}}} \lesssim 1\% \sim 10^{18}$$ sim'd events needed for Run-2 dataset



> 90% of time

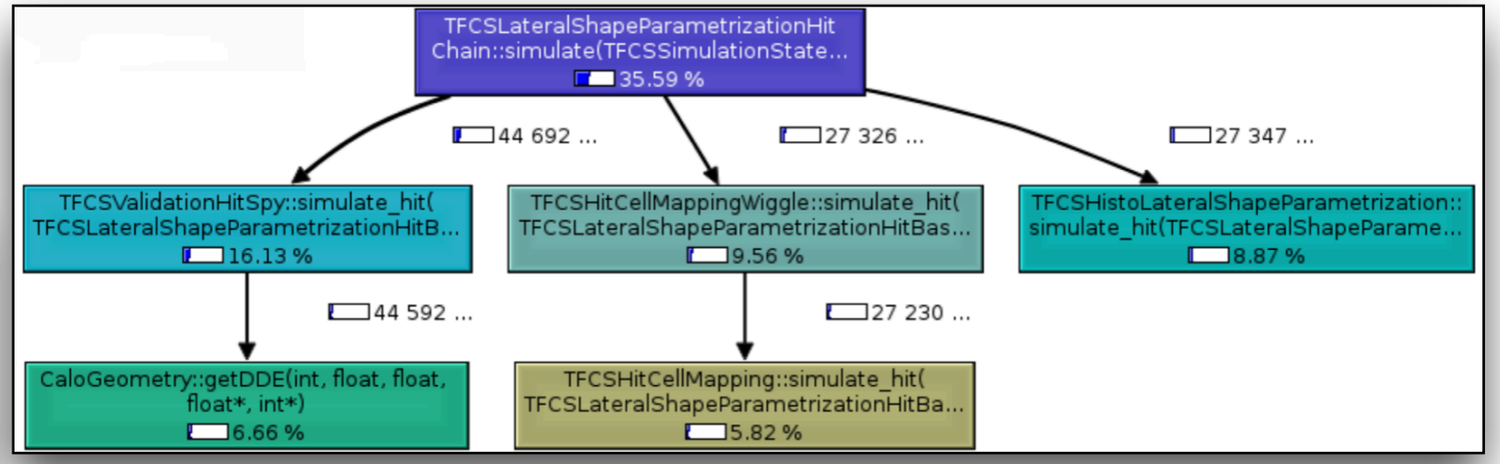
spent here



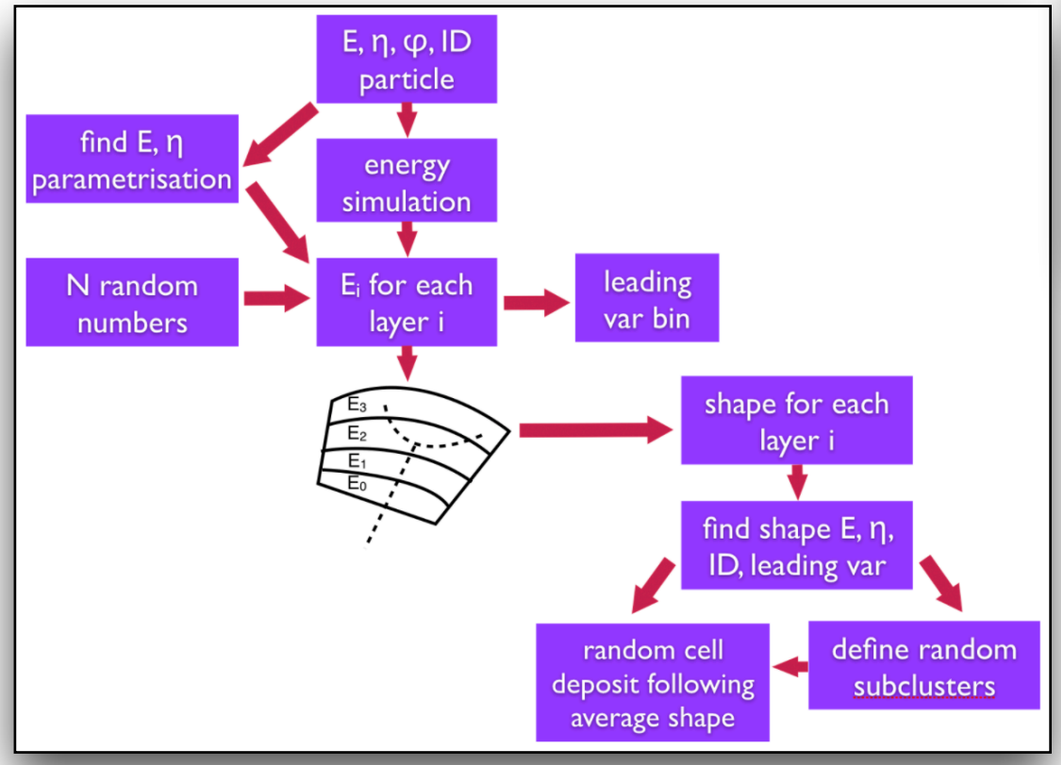
FastCaloSim(V2)

GPU simulation port strategy

No data dependence; Can parallelize hit-processing



Simulation schematic



AtlasComputing/FastCaloSimNewParametrization

- ⊕ GPU-based geometry
  - ⊖ ~200k cells in 24 layers (about 20MB)
  - ⊖ Flatten and simplify for GPU-friendly data structures, e.g. non-standard layouts not supported
- ⊕ Multi-stage kernels
  - ⊖ Block-/work-group-wise atomic update for shared memory
  - ⊖ Reduction of results from all blocks/work-groups
- ⊕ Offload
  - ⊖ [Device] Geometry-based calculations, e.g. extrapolation, hit cells
  - ⊖ [Device] RNG, e.g. energy and hit distributions
  - ⊖ [Shared] Histogram data, e.g. cell counts and energies

Portability

Target multiple backends

- Investigating a number of different parallel computing platforms
  - CUDA (NVIDIA) → industry "standard" and initial target from BNL CSI; only NVIDIA devices
  - Kokkos (Sandia, LANL, ORNL)→ CUDA, HPX, OpenMP and pthreads; CUDA wrapper
  - SYCL (Intel, Codeplay)→ SPIR-V-compatible devices, OpenMP/CL, CUDA*; code rewrite, utilizing DPC++ extensions, e.g. USM, sub-groups

Geometry transfer

Device	malloc_shared [ms]	malloc_device [ms]
Host	0.01114258	0.00351570
CPU	0.00347361	0.00337206
Intel Gen9	0.00421972	0.00366639
CUDA	0.00775582	0.00449499

nvprof					
CUDA	time / ms	calls	avg / us	min / us	max / us
clean	38.6	19975	1.93	1.85	2.3
sim_a	391.7	19975	19.6	12.7	50
sim_ct	49.7	19975	2.49	2.43	2.84
mem d->h	37.9	39950	0.95	0.895	1.76
nvprof (NVIDIA RTX 2080 Super)					
Kokkos/CUDA	time / ms	calls	avg / us	min / us	max / us
clean	61.874	19975	3.097	2.944	3.296
sim_a	366.03	19975	18.324	12.672	47.968
sim_ct	70.973	19975	3.553	3.328	3.776
mem d->h	38.669	40193	0.962	0.895	2.4
rocprofiler (AMD Vega 56)					
Kokkos/HIP	time / ms	calls	avg / us		
clean	177.565	19975	8.889		
sim_a	26042.777	19975	1303.768		
sim_ct	220.934	19975	11.06		
mem d->h	55.817		2.794		

Benchmarks from C.Leggett:
- 10k single-electron events within $0.20 < \eta < 0.25$; single layer (EMB).

N.B. Geo-load not included in "mem d->h".

Benchmarks from vrp (SYCL port):
- ~200k cells, ~20MB

"Node 1" (white cells) and "Node 2" (grey cells).
N.B. Gen9 uses Unified Memory Architecture (UMA).

Reproducibility

e.g. `SyclRng/SimHitRng_test`

```
$ cmake <…> -DSYCL_RNG_CPU=on <…> && cmake --build .
$ ./generic/test/FastCaloSycl/SimHitRng_test
SimHitRng BEGIN
test1()
SimHitRng device:
"Intel(R) Xeon(R) CPU E3-1585 v5 @ 3.50GHz”

Device random number memory allocated...
   allocd_num_hits: 1000
   size: 4 kb

Generated random numbers…
<…>
  test_random_num[384] = [0.679990]
  test_random_num[474] = [0.388999]
  test_random_num[19] = [0.249990]
  test_random_num[999] = [0.578832]
  test_random_num[215] = [0.999082]
  test_random_num[227] = [0.199966]
<…>
```

```
$ cmake <…> -DSYCL_RNG_GPU=on <…> && cmake --build .
$ ./generic/test/FastCaloSycl/SimHitRng_test
SimHitRng BEGIN
test1()
SimHitRng device:
"Intel(R) Gen9 HD Graphics NEO”

Device random number memory allocated...
   allocd_num_hits: 1000
   size: 4 kb

Generated random numbers...
<…>
  test_random_num[999] = [0.578832]
  test_random_num[474] = [0.388999]
  test_random_num[215] = [0.999082]
  test_random_num[227] = [0.199966]
  test_random_num[19] = [0.249990]
  test_random_num[384] = [0.679990]
<…>
```

All supported devices on given machine generate same random numbers (to precision shown), e.g. Intel CPU and Iris GPU. No MKL support for CUDA; workaround by using CPU device.

`FastCaloSycl` github project

https://github.com/vpascuzz/FastCaloSim-GPU/tree/dpcpp

Forked from a private repo;
send me your github account ID if you want access.

Backup

FastCaloSycl

What is it?

- ⊕ Port CPU-intensive code from `Simulation/ISF/ISF_FastCaloSim`
- ⊕ Using Intel/llvm (DPC++), oneMKL
- ⊕ Investigation of SYCL as an accelerator "portability solution" using FastCaloSim(V2) as a testbed

Package (re-)structure

`FastCaloSycl` overview

- ⊕ Refactored code, added new data structures
  - ⊖ Comprises several libraries: `FastCaloSycl`, `SyclGeo`, `SyclCommon` and `SyclTest`
  - ⊖ Serialization for GPU-friendly structures
  - ⊖ Simulation kernel function object
  - ⊖ MKL-based random number generator
- ⊕ Updated `cmake` configuration
  1. `SyclGeo`: `SYCL_TARGET_{CUDA,DEFAULT,CPU,GPU}`
  2. `SyclRng`*: `SYCL_RNG_{~~CUDA~~,DEFAULT,CPU,GPU}`

} Possible to use distinct devices for geometry and RNG

* Not all DPC++ features are compatible with CUDA devices, e.g. experimental::printf, MKL.

Package (re-)structure (II)

FastCaloSimAnalyzer > FastCaloSycl > CMakeLists.txt

```
# Copyright (C) 2002-2020 CERN for the benefit of the ATLAS collaboration

cmake_minimum_required(VERSION 3.10)

add_library(${FastCaloSycl_LIB} SHARED)

set(CMAKE_C_COMPILER "clang")
set(CMAKE_CXX_COMPILER "clang++")

set(SYCL_TARGET_DEFAULT OFF CACHE BOOL "Use the default device for simulation")
set(SYCL_TARGET_CPU OFF CACHE BOOL "Use the CPU device for simulation")
set(SYCL_TARGET_GPU OFF CACHE BOOL "Use the GPU device for simulation")
set(SYCL_TARGET_CUDA OFF CACHE BOOL "Enable CUDA backend for simulation")

add_definitions(-DUSE_SYCL)
```

} Select device to hold geometry, run non-RNG calculations

FastCaloSimAnalyzer > FastCaloSycl > SyclRng > CMakeLists.txt

```
# Copyright (C) 2002-2020 CERN for the benefit of the ATLAS collaboration

cmake_minimum_required(VERSION 3.10)

add_library(SyclRng SHARED)

set(CMAKE_C_COMPILER "clang")
set(CMAKE_CXX_COMPILER "clang++")

set(CMAKE_CXX_FLAGS "-g -O2 -fsycl -std=c++17 -Wno-unknown-cuda-version")

set(SYCL_RNG_DEFAULT OFF CACHE BOOL "Use default device for RNG")
set(SYCL_RNG_CPU OFF CACHE BOOL "Use CPU device for RNG")
set(SYCL_RNG_GPU OFF CACHE BOOL "Use GPU device for RNG")

if (SYCL_RNG_DEFAULT)
  message(STATUS " Targetting default SYCL device for RNG")
  add_definitions(-DSYCL_RNG_DEFAULT)
elseif (SYCL_RNG_CPU)
  message(STATUS " Targetting CPU SYCL device for RNG")
  add_definitions(-DSYCL_RNG_CPU)
elseif (SYCL_RNG_GPU)
  message(STATUS " Targetting GPU SYCL device for RNG")
  add_definitions(-DSYCL_RNG_GPU)
endif()
```

} Select device to perform RNGation

Class overview

- `SyclGeo::GeoRegion`
  - Data class to represent a region of the calorimeter
  - Comprises cells (i.e. `CaloDetDescrElement`), extents
  - Converts `TObject`-derived data structure(s) to standard-layout* class for device
- `SyclGeo::Geo`
  - Holds full calorimeter geometry
  - Allocates device-side memory, and performs loading of host-side to device-side memory using USM
  - Device-side geometry stored in struct, `Geo::DeviceGeo`
- `SyclRng::SimHitRng`
  - Used for sampling number of hits, particle energies
  - RNG done on device (host, SPIRV) with MKL
- `SyclCommon::DeviceCommon`
  - Access `cl::sycl::queue`s and associated `cl::sycl::device`s, and `CUDASelector` class
  - Various `cl::sycl::experimental::printf()` definitions
- `SyclCommon::Props`
  - Contains constants and simulation-related structures
- `FastCaloSycl::SimEvent`
  - Performs simulation using above classes
  - Kernel function object; class implementing `void operator()(…)`; target callable object is called within command group scope

* While DPC++ claims to support non-standard layouts, virtual functions are not [yet] callable in SYCL kernels.

Device-exclusive memory

SyclRng::SimHitRng

```cpp
bool SimHitRng::Alloc(unsigned long ncells, unsigned short max_unique_hits) {
  // Don't reallocated if memory is already allocated.
  if (is_initialized_) {
    return true;
  }

  // Allocate device-side memory for random numbers array.
  random_nums_ =
      (float*)malloc_device(allocd_num_hits_ * sizeof(float), device_, ctx_);
  if (!random_nums_) {
    std::cout << "Cannot allocate device-side memory for random numbers!"
              << std::endl;
    return false;
  }
  // Allocate memory for energy in cells
  cells_energy_ = (float*)malloc_device(ncells * sizeof(float), device_, ctx_);
  if (!cells_energy_) {
    std::cout << "Cannot allocate device-side memory for cell energy!"
              << std::endl;
    return false;
  }
  // Allocate memory for cell properties
  cell_props_ = (fastcalosycl::CellProps*)malloc_device(
      max_unique_hits * sizeof(fastcalosycl::CellProps), device_, ctx_);
  if (!cell_props_) {
    std::cout << "Cannot allocate device-side memory for cell properties!"
              << std::endl;
    return false;
  }
  // Allocate memory for unique hits counter
  num_unique_hits_ = (int*)malloc_device(sizeof(int), device_, ctx_);
  if (!num_unique_hits_) {
    std::cout << "Cannot allocate device-side memory for unique hits counter!"
              << std::endl;
    return false;
  }
  is_initialized_ = true;
  return true;
}
```

RNGation performed exclusively on device;

Copy to host if/when needed:
`cl::sycl::queue::memcopy()`

Unit tests

Test library functionality

- `SimHitRng_test`
  - Test pseudo-random number generation on device
- `GeoRegion_test`
  - Read in geometry via `CaloGeometryFromFile` object (virtual interface)
  - Test pointer arithmetic; ensure cells are correct
- `Geo_test`
  - Extension of `GeoRegion_test`
  - Copies geometry (`CaloDetDescrElements`, `GeoRegions`) to device
  - Perform cell and region cross-checks

Status and `TODO`

- ⊕ All necessary pieces tested and in place
- ⊕ Start simulating
  - ⊖ Finish implementing `FastCaloSycl::SimEvent // TODO`
  - ⊖ Perform histogramming on GPU `// TODO`
- ⊕ Benchmarking
  - ⊖ Against Intel CPU and iGPU, and native CUDA versions `// TODO`
    - ⊖ Expect results by end of the month
  - ⊖ Should have access to Intel DG (dGPU) cards soon (more of apples-to-apples comparison, e.g. no shared silicon between host-device memory)