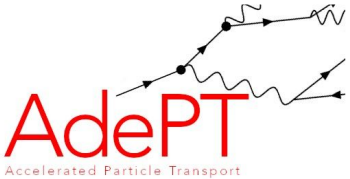


AdePT project

porting particle transport simulation to oneAPI

-user experience-

Daniel-Florin Dosaru

Thesis advisor: Prof. James Larus
CERN mentor: Predrag Buncic

Master project report presentation

August 23rd 2021

EPFL

About me

- Master student in Computer Science at EPFL
  - Software systems specialization
- OpenLab Technical Student at CERN sponsored by Intel
  - EP-SFT department
- Little background in C++
  - no prior SYCL or compiler experience

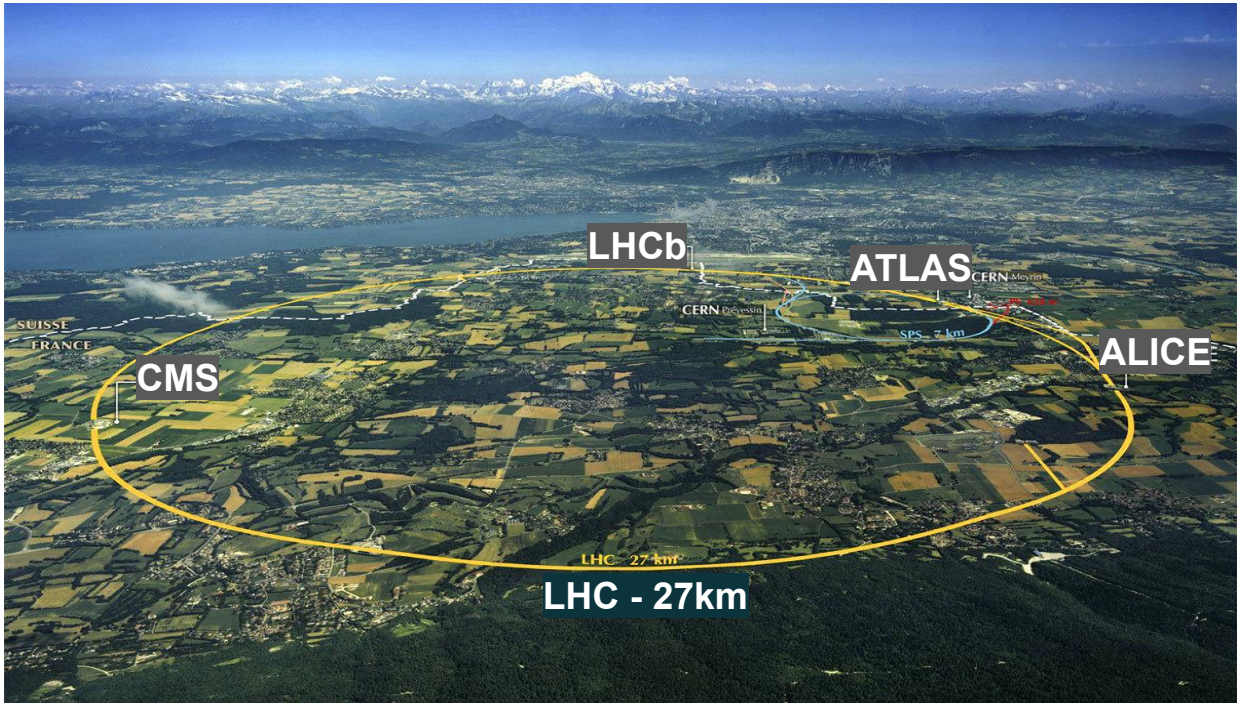
dosarudaniel@gmail.com
Linkedin, Github

Content

- High Energy Physics context
- Detector simulation
- AdePT project
- Related work
- oneAPI and SYCL
- User experience in porting simulation code (AdePT project) to oneAPI
  - SYCL limitations and other issues
  - Workarounds
- Conclusions

About CERN and Large Hadron Collider (LHC)

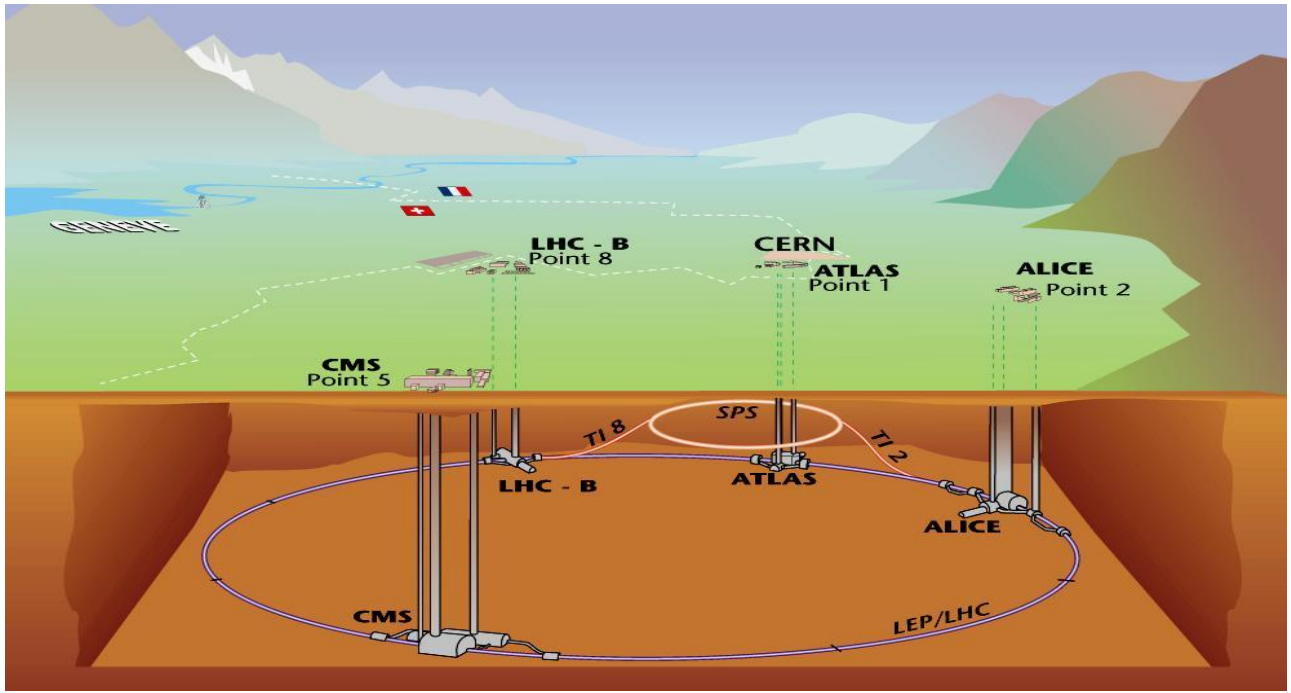
- Between 50 and 175 meters underground
- The highest-energy and largest particle accelerator ever built
- **Rare phenomena are studied,** huge quantities of recorded collisions (events) must be considered



Aerial shot of the region where LHC is placed

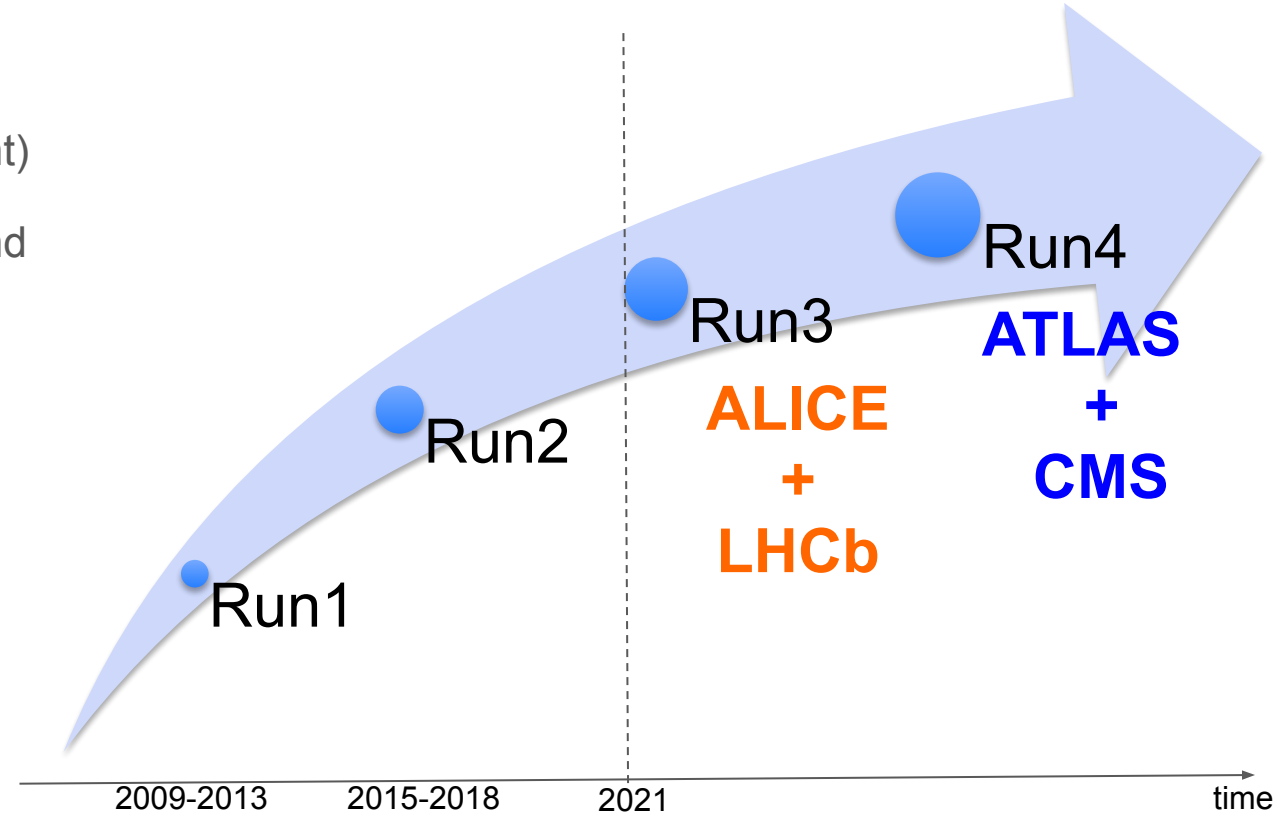
Physics experiments at the LHC

- LHC: one accelerator, four (main) experiments: ALICE, ATLAS, CMS, LHCb
- The four experiments share the same beams and take data at the same time
- Different physics reach: different detector technologies, different readout speeds, different data volumes



Upgrade schedule

- CPU needs (per event) will grow with track multiplicity (pileup) and energy
- Storage needs are proportional to accumulated luminosity



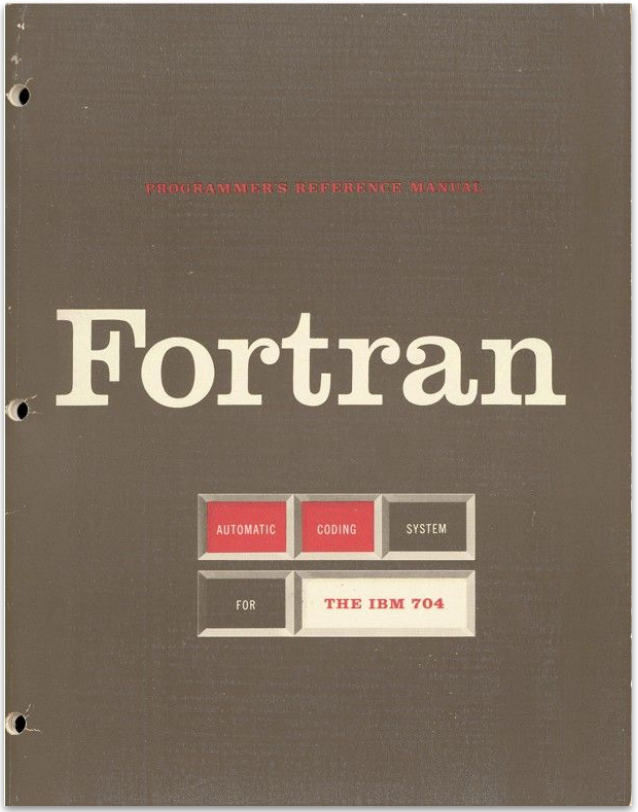
Computing @ LHC

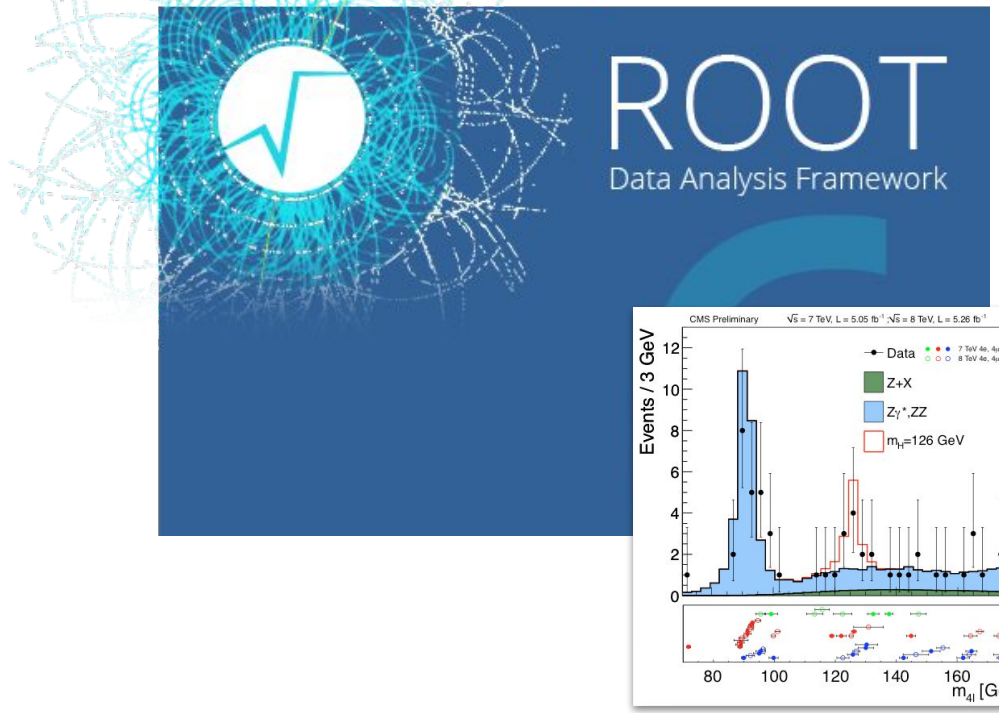
- huge detectors
  - physical size and number of electronic channels
- large collaborations
  - thousands of people
  - hundreds of institutions

- decade of lead time before the construction
- decade of construction and commissioning
- at least two decades of exploitation and continuous upgrades

- huge time span for software
- CERN has to: preserve, proces and re-process its data from different stages.

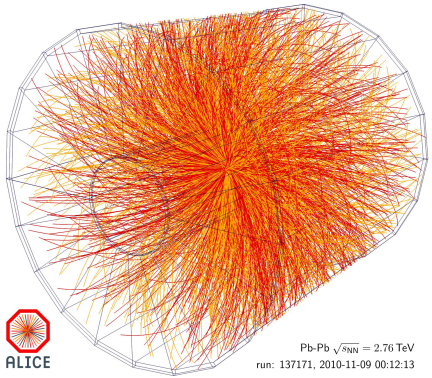
Migration from Fortran to C++...





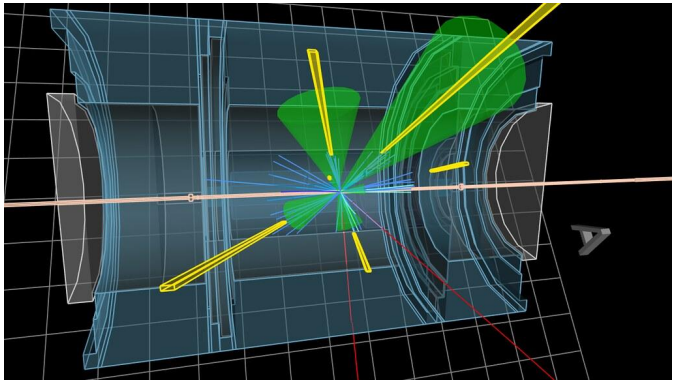
C++ @ LHC

- Millions of lines of code on C++ to support :
  - Raw data reconstruction
  - Simulation
  - Analysis



Recently, there was a significant penetration of python in particular on the end user analysis side.

C++ @ LHC - Detector simulation



Simulation of the interaction of elementary particles with matter

- also written in C++ on top of Geant4 toolkit developed by HEP community in collaboration with other scientific communities

Simulating the physics requires large amount of computing resources

- 45-90% of total computing budget goes to simulation

Worldwide LHC Computing Grid

CERN’s 15 years old distributed computing and data storage infrastructure

- spread over 150 computing centres
- 40 countries

It currently operates about

- 1M of commodity CPU cores
- 1EB of storage.

and it is growing...

The end of an era...

- In the past benefited from a significant underlying simplifications:
  - One platform (linux)
  - One architecture (x86)
  - One compiler (gcc)
- Already the end of the Run 1 it was clear that CERN's life is going to get significantly more complicated:
  - Growing needs for computing resources cannot be matched by the growth of the Grid under the fixed budget
- Distributed funding ⟶ the end of homogenous platform dream

Why HPCs and GPUs

- Forced to look into hardware accelerators such as GPUs
- By now the experiments are making significant use of GPUs for their online and offline reconstruction.

- Since a large fraction of CPU time is actually spent in simulation CERN needs to look into how it could benefit from GPUs in simulation.

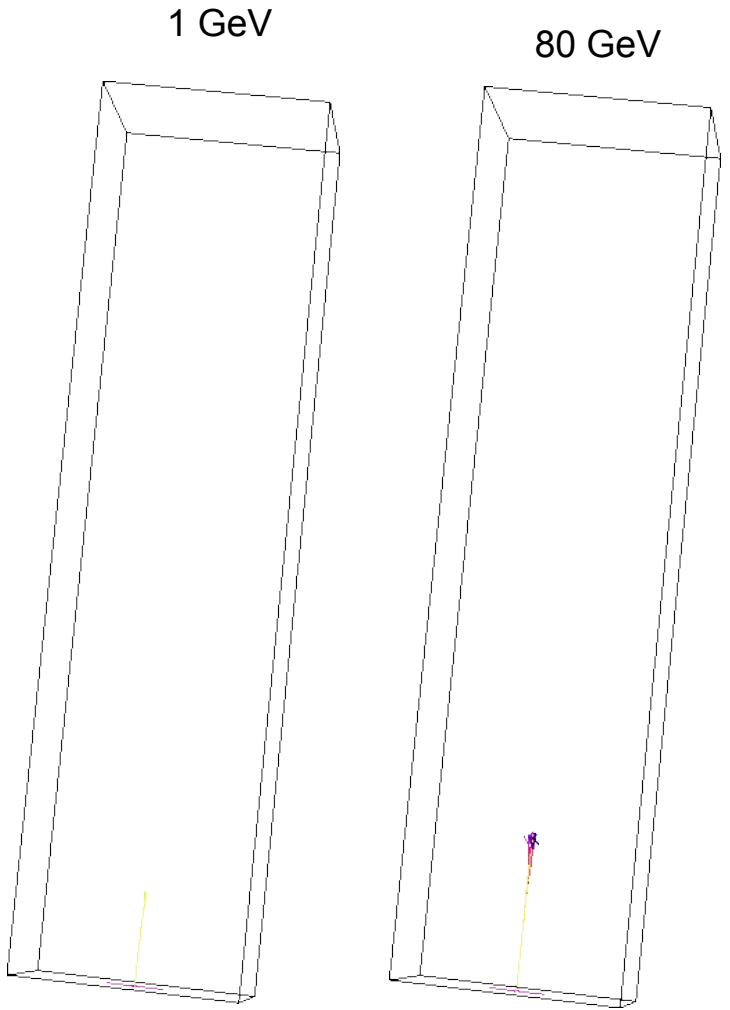
AdePT Project

Particle transport simulation is generally not well suited for running on hardware accelerators such as GPUs

- Some aspects of simulation, such as simulation of developing showers in the calorimeters, could make use of GPUs, but:
  - very time consuming
  - requires only a subset of physics processes to be ported to GPU

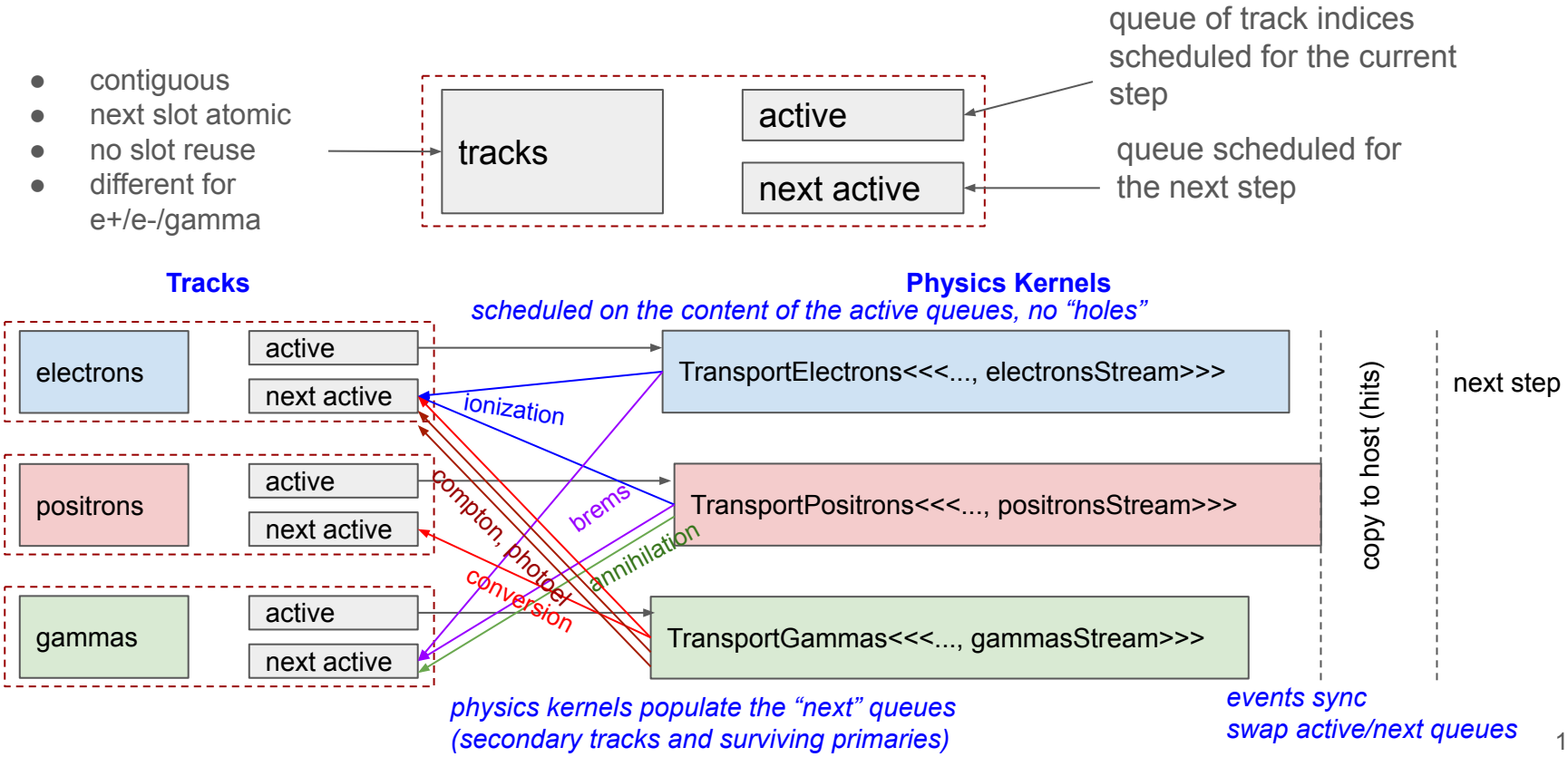
Accelerated **D**emonstrator of **E**lectromagnetic **P**article **T**ransport

- R&D project aiming to port Electromagnetic (EM) physics transport components of Geant4 physics simulation to GPU and provide a plug-in applicable to specific “regions”.
- Specifically targeting calorimeter simulations



Slide adapted from Stephan Hageboeck’s slides

Particle transport workflow



Related work

- CUDA - speed up computing applications by leveraging the power of NVIDIA GPUs
  - targeting only NVIDIA hardware is not sufficient

- Allen [1] - A High Level Trigger on GPUs for LHCb - A farm of 500 GPUs can process the track reconstruction for several sub-detectors at 40Tbit/s
  - implementation entirely on GPUs, not applicable for the simulation

- At the ALICE experiment [2] CPU and GPU algorithms are shared and written using preprocessor macros. Supports GPUs via CUDA, OpenCL and HIP, parallelizes the algorithms on CPU using OpenMP
  - Custom macros must be maintained for each platform

- alpaka [3] Abstraction Library for Parallel Kernel Acceleration developed by Helmhotz/Dresden and the CASUS institute includes a library and tools for programming on hardware accelerators.
  - fisher_price using alpaka

and

oneAPI (launched in June 2019 [4])

- unified programming model that
- delivers a common developer experience across different Intel hardware

Intel oneAPI uses DPC++ as a programming language

- open, cross-architecture language built on the ISO C++
- incorporates SYCL from the Khronos Group

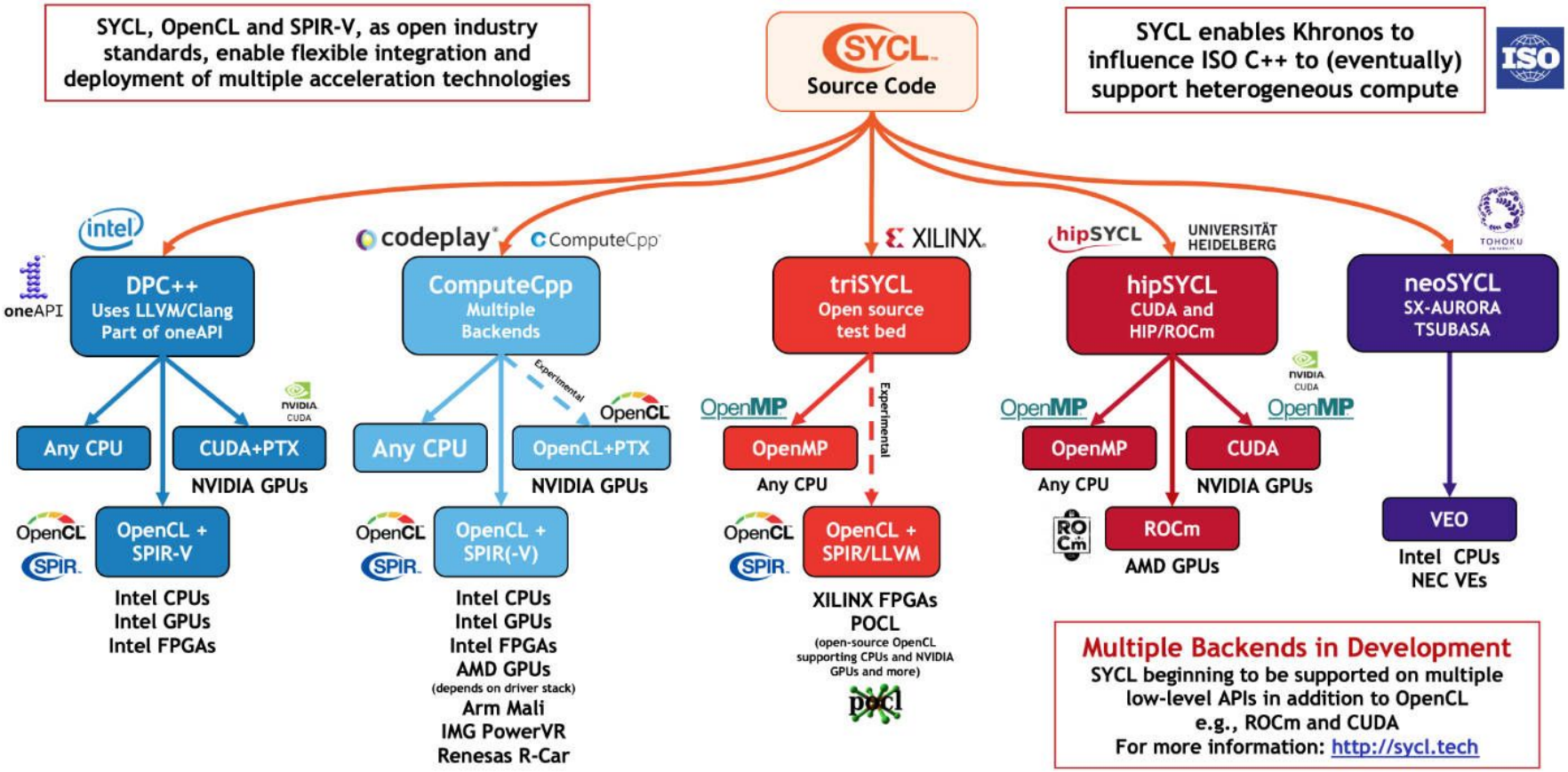
SYCL is a new programming model based on OpenCL

- hides the complexity of OpenCL by reducing the host-side code needed

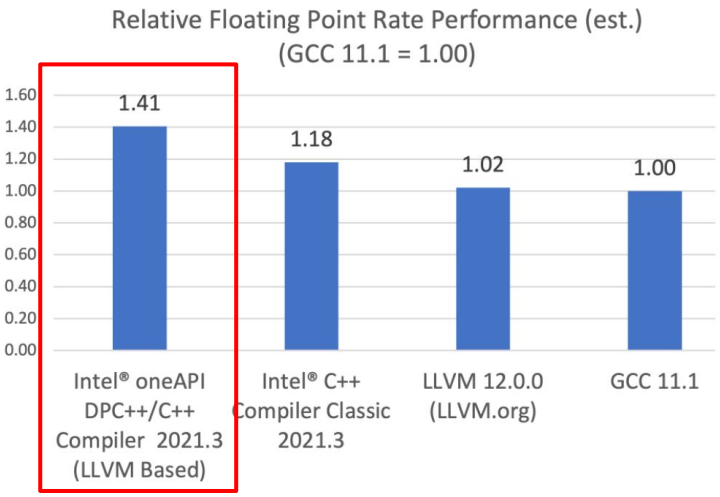
Codeplay contribution to DPC++ brought SYCL support for NVIDIA GPUs (also future developments for AMD GPUs).

With single invocation of the compiler which will produce a fat binary.

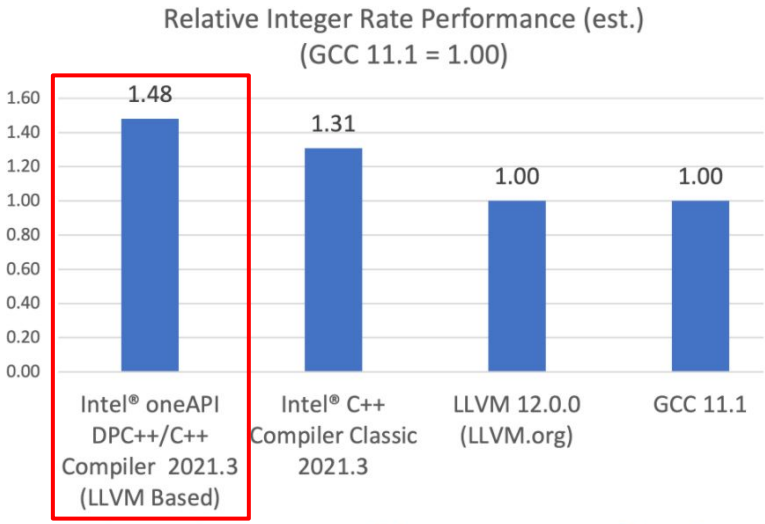
SYCL implementations



Benefits of adopting LLVM



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate 2017 Floating Point suite



Estimated: internal measurement of the geometric mean of the C/C++ workloads from the SPECrate 2017 Integer suite

SPECspeed 2017

Better performance

Source: [Intel C/C++ compilers complete adoption of LLVM](), August 2021

oneAdePT

- The current AdePT code depends on external physics (G4HepEm) and geometry (VecGeom) libraries.
  - Currently targeting the NVIDIA/CUDA platform
  - In future: target various CPU and GPU targets starting from the same source
- **oneAdePT** is my attempt to port AdePT point in time snapshot to oneAPI. It includes:
    - AdePT core utilities and particle transport
    - RUNLUX++ random number generator
    - magnetic field routines
    - G4HepEm library.
- The goal was to get experience in CUDA code migration to DPC++ and evaluation of DPC++ ability to use with legacy CUDA libraries (VecGeom).
- Source code: https://github.com/dosarudaniel/oneAdePT

Work and initial road map

- Daily meetings with hands-on sessions
- Logging daily progress, issues and solutions in a shared document

Initial road-map:

1. Convert CUDA sample codes (unit tests and fisher_price) to oneAPI
2. Convert example9 demonstrator to oneAPI
3. Update cmake files to be able to build example9 project
4. Start fixing compile errors until it compiles
5. Start debugging until it runs successfully
6. Compare performance to native CUDA application
7. Convert cuda G4HepEm library to oneAPI and repeat performance test
8. Convert vecgeom to oneAPI and repeat performance test
9. Compare the performance of the code running on CPU and GPU
10. Run the applications on Intel's devCloud and compare performance on Intel GPU and CPU using dpcpp.
11. Write final documentation

Migration from CUDA to DPC++

I used Intel® DPC++ Compatibility Tool (dpct) to bootstrap the porting process

- There are some unavailable equivalent API's in SYCL which I had to rewrite
- But, the tool has useful diagnostics messages (+ centralized reference)
  - Validating the use of work-group sizes

Intel® DPC++ Compatibility Tool in use

```
// *** ELECTRONS ***
int numElectrons = stats->inFlight[ParticleType::Electron];
if (numElectrons > 0) {
  transportBlocks = (numElectrons + TransportThreads - 1) / TransportThreads;
  transportBlocks = std::min(transportBlocks, MaxBlocks);

  TransportElectrons<true><<<transportBlocks, TransportThreads, 0, electrons.stream>>>(
      electrons.tracks, electrons.queues.currentlyActive, secondaries, electrons.queues.nextActive,
      electrons.queues.relocate, scoring);

  COPCORE_CUDA_CHECK(cudaEventRecord(electrons.event, electrons.stream));
  COPCORE_CUDA_CHECK(cudaStreamWaitEvent(stream, electrons.event, 0));
}
```

.cu file

```
// *** ELECTRONS ***
int numElectrons = stats->inFlight[ParticleType::Electron];
if (numElectrons > 0) {
  transportBlocks = (numElectrons + TransportThreads - 1) / TransportThreads;
  transportBlocks = std::min(transportBlocks, MaxBlocks);

  electrons.stream->submit([&](sycl::handler &cgh) {
    Track *electronsTracks = electrons.tracks;
    adept::MParray *currentlyActive = electrons.queues.currentlyActive;
    adept::MParray *nextActive = electrons.queues.nextActive;
    adept::MParray *relocate = electrons.queues.relocate;
    cgh.parallel_for(
        sycl::nd_range<3>(sycl::range<3>(1, 1, transportBlocks) *
                              sycl::range<3>(1, 1, TransportThreads),
                          sycl::range<3>(1, 1, TransportThreads)),
        [=](sycl::nd_item<3> item_ct1) {
          TransportElectrons<true>(electronsTracks,
                                   currentlyActive,
                                   secondaries,
                                   nextActive,
                                   relocate,
                                   scoring,
                                   item_ct1,
                                   electronManager_p,
                                   g4HepEmPars_p,
                                   g4HepEmData_p);
        });
  });
  /*
  DPCT1012:17: Detected kernel execution time measurement pattern and
  generated an initial code for time measurements in SYCL. You can change
  the way time is measured depending on your goals.
  */
  /*
  DPCT1024:18: The original code returned the error code that was further
  consumed by the program logic. This original code was replaced with 0. You
  may need to rewrite the program logic consuming the error code.
  */
  electrons.event_ct1 = std::chrono::steady_clock::now();
```

.dp.cpp file

Compilation experience

Used Intel's **open source** DPC++ compiler (clang++) since I really needed to use the Codeplay contribution to DPC++ which adds support for NVIDIA GPUs

- Intel's DPC++ does **not** have support for non-Intel target
- For this use case, support for CUDA BE in DPC++ is important

- Helpful and enthusiastic SYCL developer community
- Complexity of the full software stack
- Documentation is hard to find (not centralized)
- User community not yet well established
  - Few examples from other users
- Converted source compiled successfully

SYCL standard limitations

- SYCL kernel cannot call a virtual function
  - Addressed by adding a non-virtual dispatch layer in AdePT ✅
- SYCL kernel cannot use a non-const global variable
  - Added the variables as kernel parameters ✅
- SYCL kernel cannot call through a function pointer
  - Abstract classes were converted to concrete classes ✅
- Math functions in the kernel must be used from sycl:: namespace, not std::
  - Using preprocessor macros ✅

Using math functions from std::

Calling std:: math function in the SYCL kernel:

```
fatal error: error in backend: Undefined external symbol "log"
clang-13: error: clang frontend command failed with exit code 70
```

or

```
fatal error: error in backend: Cannot select: t53: f64 = fcos t52
  t52: f64 = fdiv t51, t37
```

test12.cpp

```
void kernel(double *step)
{
  *step =  log(*step);
}
```

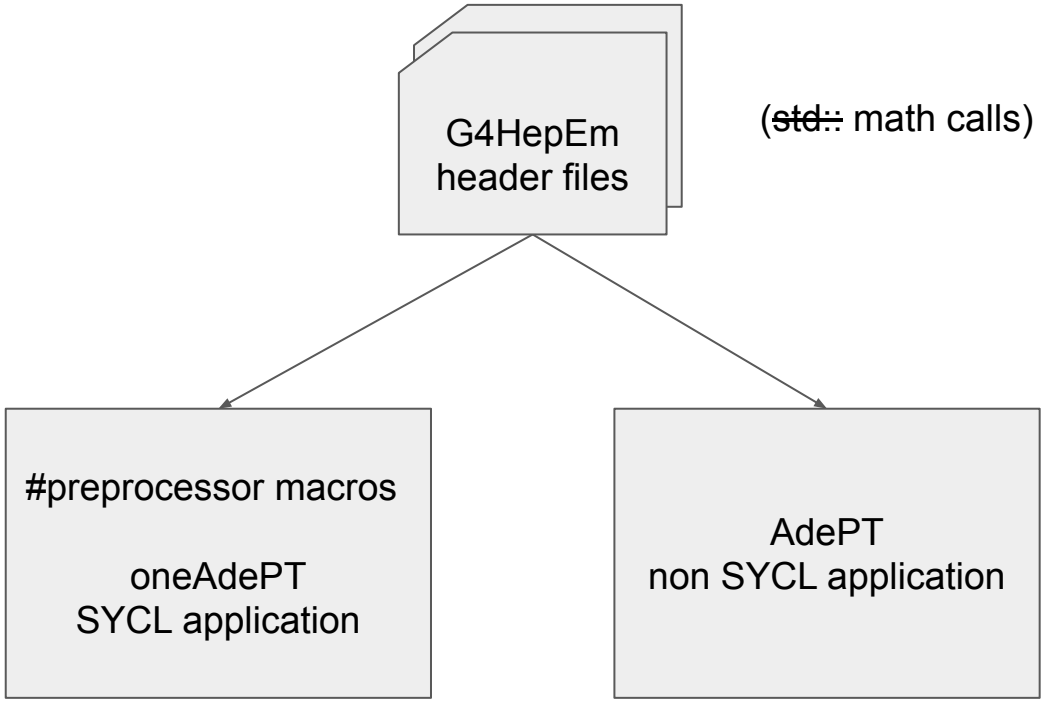
```
#if (defined( __SYCL_DEVICE_ONLY__))
#define log sycl::log
#else
#define log std::log
#endif

void kernel(double *step)
{
  *step =  log(*step);
}
```

Math functions which needed this hack:
log, exp, cos, sin, pow, frexp, ldexp, modf

Other functions which did not need this hack:
abs, fabs, sqrt

Reported on github

Usage of std:: math function (context)



Limitations: supporting a 3rd party library (1)

No out of the box support for 3rd party libraries: cuda compiled libraries

basic_add.cu

```
__host__ __device__ int add_test(int a, int b)
{
    return a + b;
}
```

$ clang++ -fgpu-rdc --cuda-gpu-arch=sm_75 field.cu -c → **basic_add.o**

test.cpp

```
extern SYCL_EXTERNAL int add_test(int a, int b);

void kernel(double *result)
{
  *result = add_test(1, 2);
}
```

*$ clang++ test.cpp **basic_add.o** -fsycl -fsycl-unnamed-lambda -fsycl-targets=nvptx64-nvidia-cuda-sycldevice*

Linking error: "ptxas fatal : Unresolved extern function '_Z11add_testii'"

basic_add.o is not passed to ptxas and thus it can't find required symbol.

Check the github discussion on this topic: https://github.com/intel/llvm/discussions/3627
Other compilation and linking trials described in this document

Limitations: supporting a 3rd party library (2)

Trying to use a cpp wrapper:compiling

basic_add.cu

```
__host__ __device__ int add_test(int a, int b)
{
    return a + b;
}

int my_add_test(int a, int b){ return add_test(a, b);} //wrapper
```

nvcc -dc basic_add.cu → basic_add.o

test.cpp

```
extern SYCL_EXTERNAL int my_add_test(int a, int b);

int main(void)
{
  sycl::default_selector device_selector;

  sycl::queue q_ct1(device_selector);
  std::cout <<  "Running on " << q_ct1.get_device().get_info<cl::sycl::info::device::name>() << "\n";

  q_ct1.submit([&](sycl::handler &cgh) {
    cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, 1),
      sycl::range<3>(1, 1, 1)),
      [=](sycl::nd_item<3> item_ct1) {
        my_add_test(1, 2); // ptxas fatal   : Unresolved extern function '_Z11my_add_testii'"
    });
  }).wait();
}
```

test.o

```
$ clang++ -c test.cpp -o test.o -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice -fsycl-unnamed-lambda
```

Suggested idea https://stackoverflow.com/a/64527623

Limitations: supporting a 3rd party library (3)

Trying to use a cpp wrapper:compiling

basic_add.cu

```
1  __host__ __device__ int add_test(int a, int b)
2  {
3      return a + b;
4  }
5
6  int my_add_test(int a, int b){ return add_test(a, b);} //wrapper
```

nvcc -dc basic_add.cu → basic_add.o

test.cpp

```
extern SYCL_EXTERNAL int my_add_test(int a, int b);

int main(void)
{
  sycl::default_selector device_selector;

  sycl::queue q_ct1(device_selector);
  std::cout <<  "Running on " << q_ct1.get_device().get_info<cl::sycl::info::device::name>() << "\n";




      my_add_test(1, 2);


}
```

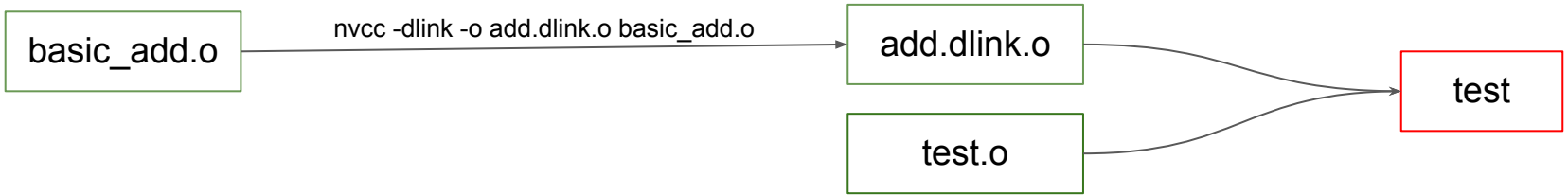
test.o

```
$ clang++ -c test.cpp -o test.o -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice -fsycl-unnamed-lambda
```

Suggested idea https://stackoverflow.com/a/64527623

Limitations: supporting a 3rd party library (4)

Trying to use a cpp wrapper:compiling




```
$ clang++ test.o add.dlink.o basic_add.o -o test -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice -fsycl-unnamed-lambda
-L/usr/local/cuda-11.1/targets/x86_64-linux/lib/ -lcudart
```

```
Linking error: "ptxas fatal   : Unresolved extern function '_Z11my_add_testii'"
```

Attempting to use interop_task

Note: The ***interop_task*** and ***host_task*** commands are only callable from the command group scope.

```
__global__ void stepInField(double *res, double *kinE, double *mass, int *charge, int n)
{
    // Get our global thread ID
    int id = blockIdx.x * blockDim.x + threadIdx.x;

    // Make sure we do not go out of bounds
    if (id < n) {
        res[id] = kinE[id]*mass[id]*charge[id];
    }
}
```

```
q_ct1.submit([&](sycl::handler &cgh) {

    auto accRes = bRes.get_access<sycl::access::mode::write>(cgh);
    auto acckinE = bkinE.get_access<sycl::access::mode::read>(cgh);
    auto accMass = bMass.get_access<sycl::access::mode::read>(cgh);
    auto accCharge = bCharge.get_access<sycl::access::mode::read>(cgh);

    cgh.interop_task([=](sycl::interop_handler ih) {
        auto res = reinterpret_cast<double *>(ih.get_mem<sycl::backend::cuda>(accRes));
        auto kinE = reinterpret_cast<double *>(ih.get_mem<sycl::backend::cuda>(acckinE));
        auto Mass = reinterpret_cast<double *>(ih.get_mem<sycl::backend::cuda>(accMass));
        auto Charge = reinterpret_cast<int *>(ih.get_mem<sycl::backend::cuda>(accCharge));

        stepInField<<<1,1>>>(res, kinE, Mass, Charge, N);
    });

}).wait();
```

Source code

Recents developments

The latest version (**August 2021 release**) of the clang compiler supports calling a CUDA kernel from a SYCL kernel, but the linking steps should be changed manually.

First step is to compile the CUDA code to LLVM bitcode:

```
$ clang++ -fgpu-rdc --cuda-gpu-arch=sm_50 basic_add.cu -emit-llvm -c
```

- `-fgpu-rdc` → Generates relocatable device code, also known as separate compilation mode
- `-emit-llvm` → Use the LLVM representation for assembler and object files
- `-c` → Run preprocess, compile, and assemble steps

This command outputs the **`basic_add-cuda-nvptx64-nvidia-cuda-sm_50.bc`** file (llvm bitcode code)

Calling the CUDA kernel

test.cpp

```
extern SYCL_EXTERNAL int add_test(int a, int b);

void kernel(double *result)
{
  *result = add_test(1, 2);
}
```


test.cpp

```
extern SYCL_EXTERNAL int add_test(int a, int b);

void kernel(double* result)
{
  #if defined(__SYCL_DEVICE_ONLY__) && defined(__NVPTX__)
    *result = add_test(1, 2);
  #endif
}
```

https://github.com/intel/llvm/discussions/3627#discussioncomment-1136268

Manually changing the llvm-link step

1. `clang-13 -cc1 -triple nvptx64-nvidia-cuda-sycldevice [...] test.cpp`
2. `llvm-link [...]` **`basic_add-cuda-nvptx64-nvidia-cuda-sm_50.bc`**
3. `sycl-post-link [...]`
4. `clang-13 [...] -o /tmp/test12-55b00d.s`
5. `ptxas -m64 -O0 -gpu-name sm_50 -output-file /tmp/test-7885b2.o" "/tmp/test-55b00d.s"`
6. `fatbinary [...]`
7. `clang-offload-wrapper [...]`
8. `llc [...]`
9. `clang-13 [...]`
10. `ld [...] basic_add.o -o test`

Compilation steps for test.cpp when adding “-###” flag

Current status of oneAdePT

- oneAdePT finally compiles and links successfully
- Converted sources and detailed instructions are available on github:

https://github.com/dosarudaniel/oneAdePT

Current status:

- Debugging runtime errors
- Reproducing the runtime error in a simpler code snippet to report them to LLVM community if needed

Conclusion

- oneAdePT repository contains the converted AdePT project from CUDA to SYCL and detailed instructions on how to compile and modify the linking steps to obtain the fat binary
- SYCL and oneAPI might have a potential to be useful towards support for heterogeneous platforms due to its simplified syntax (compared to OpenCL)
- SYCL limitations, clang limitations and bugs needed different workarounds that exceeded the initial road map
- Current status of the project: run-time errors to be fixed
- No compiler with full SYCL 2020 standard (rel. 2021) support, yet
- SYCL does not integrate well with CUDA environment, yet (especially with 3rd party libraries)
- Clang compiler is still rapidly evolving

References

[1] Roel Aaij, Johannes Albrecht, M Belous, P Billoir, T Boettcher, A Brea Rodríguez, D Vom Bruch, DH Cámpora Pérez, A Casais Vidal, DC Craik, et al. **Allen**: A high-level trigger on gpus for lhcb. Computing and Software for big Science, 4(1):1–11, 2020.

[2] ALICE GPU algorithms presentation

[3] alpaka: Abstraction Library for PArallel Kernel Acceleration github repository

[4] Intel unveils [...] oneAPI software stack with unified and scalable abstraction for heterogeneous architectures - news article

Acknowledgements

James Larus, EPFL supervisor

Predrag Buncic, CERN supervisor

Jonas Hahnfeld, Andrei Gheaţă, Pere Mato Vila, Witold Pokorski and Stephan Hageböck, CERN

Igor Vorobstov, Klaus-Dieter Oertel, Intel

Alexey Sachkov, Alexander Batashev,Steffen Larsen, llvm github community

Ruyman Reyes, Codeplay

Q&A

Thank you!

daniel-florin.dosaru@cern.ch

Backup slides

Converting the RanluxppDouble random generator

Removed CUDA annotations

```
/// Compute `a + b` and set `overflow` accordingly.
__host__ __device__
static inline uint64_t add_overflow(uint64_t a, uint64_t b,
{
  uint64_t add = a + b;
  overflow     = (add < a);
  return add;
}
```

```
/// Compute `a + b` and set `overflow` accordingly.

static inline uint64_t add_overflow(uint64_t a, uint64_t b,
{
  uint64_t add = a + b;
  overflow     = (add < a);
  return add;
}
```

Atomic operations

```
25 struct AtomicBase_t {
26
27 #ifndef COPCORE_DEVICE_COMPILATION
28   /// Standard library atomic behaviour.
29   using AtomicType_t = std::atomic<Type>;
30 #else
31   /// CUDA atomic operations.
32   using AtomicType_t = Type;
33 #endif
34
35   AtomicType_t fData{0}; ///< Atomic data
36
37   /** @brief Constructor taking an address */
38   __host__ __device__
39   AtomicBase_t() : fData{0} {}
```

```
24 struct AtomicBase_t {
25
26   using AtomicType_t =  sycl::ONEAPI::atomic_ref<Type,
27                         sycl::ONEAPI::memory_order::relaxed,
28                         sycl::ONEAPI::memory_scope::system,
29                         sycl::access::address_space::global_space>;
30
31   Type a = 0; AtomicType_t fData{a};
32
33   AtomicBase_t(Type t = 0) : fData(a) { a = t;}
```

Using sycl::ONEAPI::atomic_ref instead of std::atomic - full diff

Pointer arithmetics

SYCL device code does not support function pointers in general

- Abstract classes were transformed in concrete classes

Virtual functions

SYCL device code does not support virtual function calls

ERROR:

```
LoopNavigator.h:42:17: error: SYCL kernel cannot call a virtual function
      if (!vol->UnplacedContains(point)) return nullptr;
                ^
```

```
class VPlacedVolume:
   /// Direct dispatch to Contains of underlying unplaced volume without coordinate/placement transformation.
   VECCORE_ATT_HOST_DEVICE
   virtual bool UnplacedContains(Vector3D<Precision> const &localPoint) const = 0;
```

Virtual functions

```
#define DISPATCH_TO_PLACED_VOLUME(vol, type, func, ...)                                                \
  switch (vol->GetType()) {                                                                            \
    DISPATCH_TO_PLACED_VOLUME_KIND(vol, kBox, PlacedBox, BoxImplementation, type, func, __VA_ARGS__);  \
    DISPATCH_TO_PLACED_VOLUME_KIND(vol, kTrd, PlacedTrd, TrdImplementation<vecgeom::TrdTypes::UniversalTrd>, type, \
                                   func, __VA_ARGS__);                                                 \
    DISPATCH_TO_PLACED_VOLUME_KIND(vol, kTube, PlacedTube, TubeImplementation<vecgeom::TubeTypes::UniversalTube>,  \
                                   type, func, __VA_ARGS__);                                           \
  default:                                                                                             \
    break;                                                                                             \
  }
```

- Added a non-virtual dispatcher layer in AdepT

jonas.hahnfeld@cern.ch

Virtual functions

- Added support in VecGeom : VolumeTypes enum

jonas.hahnfeld@cern.ch

VolumeTypes.h:

```
#ifndef VECGEOM_VOLUMES_VOLUMETYPES_H_
#define VECGEOM_VOLUMES_VOLUMETYPES_H_

namespace vecgeom {
enum VolumeTypes {
  kUnknown = 0,

  kBoolean         = 1,
  kBox             = 2,
  kCoaxialCones    = 3,
  kCone            = 4,
  kCutTube         = 5,
```

UnplacedTrapezoid.h: constructor

```
VECCORE_ATT_HOST_DEVICE
UnplacedTrapezoid::UnplacedTrapezoid(
{
+   fType = VolumeTypes::kTrapezoid;
```

UnplacedVolume.h:

```
class VUnplacedVolume {

private:
  friend class CudaManager;

protected:
+   VolumeTypes fType = VolumeTypes::kUnknown;
+   /// Returns the type of this volume.
+   VECCORE_ATT_HOST_DEVICE
+   VolumeTypes GetType() const { return fType; }
```

RUNLUX++

Generators with excellent quality and statistical properties:

- MIXMAX (1991 / 2015)
- RANLUX (1993 / 1994)

RANLUX: subtract-with-borrow generator with simple recursion

- Waste intermediate states to decorrelate generated numbers
- Luxury level: higher quality with longer computing time

RANLUX++: use the equivalent Linear Congruential Generator (LCG)

- Avoid computing unneeded intermediate results, much higher quality “for free”
- Advantage: fast skipping of numbers / “jumping” in generated sequence

Paper: *A Portable High-Quality Random Number Generator for Lattice Field Theory Simulations*

Slide from jonas.hahnfeld@cern.ch