Building VecGeom: Status and Evolution

VecGeom Developers

WARWICK
THE UNIVERSITY OF WARWICK

VecGeom C++/CUDA Interface and Libraries

- VecGeom build is relatively complex
  - *Shared C++/CUDA code: .cpp → copied → .cu, Host/CUDA compiled*
  - *C++/CUDA code under separate namespaces via `#defines`*
  - *C++ code→ `libvecgeom`, CUDA code → `libvecgeomcuda`*
- Primary consumer side complexity is library structure
  - *`libvecgeom` C++ static library*
  - *`libvecgeomcuda` CUDA shared library (inc. device link)*
  - *`libvecgeomcuda_static` CUDA static library (no device link)*
- Heavy use in AdePT and Celeritas have exposed traps and pitfalls here
  - *Not unique - have seen similar discussions in CAF and HSF meetings*
- ***Both a build system and code design/use problem, so evolution needs thought, especially on integration with simulation/framework components***

Build Evolution: Native CMake Support

- Migrated to native CMake CUDA support in release 1.1.19
  - *Output library structure identical to that generated by old custom FindCUDA*
  - *Clearer documentation on how to consume CUDA libraries*

```
# Use static library
find_package(VecGeom REQUIRED CUDA)

add_executable(MyCUDA MyCUDA.cu)

set_target_properties(MyCUDA PROPERTIES CUDA_SEPARABLE_COMPILATION ON)

target_link_libraries(MyCUDA PRIVATE VecGeom::vecgeomcuda_static)
```

Build Evolution: Native CMake Support

- Migrated to native CMake CUDA support in release 1.1.19
  - *Output library structure identical to that generated by old custom FindCUDA*
  - *Clearer documentation on how to consume static/shared CUDA libraries*

```
# Use shared library
find_package(VecGeom REQUIRED CUDA)

add_library(MyCUDA MyCUDA.cu)

set_target_properties(MyCUDA PROPERTIES CUDA_SEPARABLE_COMPILATION ON)

target_link_libraries(MyCUDA PRIVATE VecGeom::vecgeomcuda)

target_compile_options(MyCUDA
  PRIVATE $<DEVICE_LINK:$<TARGET_FILE:VecGeom::vecgeomcuda_static>>)
```

Traps and Pitfalls in Current Interface/Build

- C++/CUDA libraries have a circular dependency, only identified due to build update and “in anger” use.
  - *Fixable, but shows need for care in design in evolution/portability work*
- Static CUDA library provides easiest, safest use in downstream consumers
  - *Device linking “just works”*
  - *More reliable use, or less worries about, Static/Shared CUDA Runtime use*
- Shared CUDA library is a hack for device linking as CUDA does not support this across shared library boundaries
  - *Link your target to shared library, but use static library in device link step*
  - *Does work, but can get runtime errors if Shared/Static CUDA runtime not consistent between all targets*
- **Not clear this will scale to more complex projects, whether using CUDA or portability schemes**

Future Build/Use 1: Interfaces vs Device Linking

- *Caveat: CUDA/NVIDIA focussed, and based on VecGeom experience!*
- Most discussions on build/interface seem to assume “hotspot”-like interfaces, where all device-side code is hidden from consumers of the library, e.g.:

- *Trivial to build/use as a shared or static library.*
- ***But what happens when we need to use other device code from other libraries or provide device interfaces to consumers, as in the geometry use case?***

```
// Answer.hh

int getTheAnswer(const std::vector<int>& input);
```

```
// Answer.cu
#include "Answer.hh"

__global__ getTheAnswer_kernel(const int* input, const size_t size) {
  // do something, might call other __device__ functions/code
}

int getTheAnswer(const std::vector<int>& input) {
  // ... offload
  getTheAnswer_kernel<<<...>>>(devData, sizeData);
  // ... onload
  return 42;
}
```

Future Build/Use 2: DLL(Device Link) Hell Mk 2?

- Question is ***"do we require shared libraries with public __device__ interfaces"***
- If "*yes*", then the challenge is ***"how to device link across shared library boundaries"***
  - *Solutions appear to have been adopted in HEP experiments/code, e.g. https://github.com/krasznaa/CUDALinkTest*
  - *Approach also under investigation in VecGeom: https://gitlab.cern.ch/VecGeom/VecGeom/-/merge_requests/822 , https://github.com/drbenmorgan/CudaRDCExercise*
- A second challenge would also be doing this correctly/consistently across a large dependency graph of mixed CUDA/C++ libraries.
- ***Maybe an area to engage with vendors/industry more actively if there really is no solution/good practice available?***

Future Build/Use 3: Options for Interface/Build

- If our geometry library provides device code for use by other libraries, appear to have several options:
  - ***Implement device code as header-only, kernels behind C/C++ API?***
  - ***Use static libraries everywhere?***
  - ***Come up with "standard" way to use device linking with shared libraries either ourselves or better in collaboration with vendors?***
- I don't think there's an obvious answer here whether for CUDA or portability schemes like SYCL et al, but very happy to be corrected!
  - *Clear that HEP applications will continue pattern of using building blocks of code that interdepend, so guidance/solutions will be needed, especially as geometry will be a core "lego brick".*
- **Much experience across community and industry on this, so any and all feedback and discussion is very welcome!**

In Summary

- Vecgeom now using native CMake CUDA support
- Some issues to resolve in device linking and use of shared/static CUDA runtime by library and consumers
- Evolution of the build and R&D efforts need thought and wider discussion on
  - *Use of shared libraries with device code*
  - *Interface/Organisation design to assist build and use by consumers*
  - *Support for portability solutions such as SYCL*