GeantV Geometry:
SIMD abstraction and interfacing with CUDA

Johannes de Fine Licht (johannes.definelicht@cern.ch)
Sandro Wenzel (sandro.wenzel@cern.ch)

Thanks to Michal Husejko and Romain Wartel at TechLab for providing hardware.
Abstracted algorithms

- Write *common code* for multiple types of SIMD “backends” (Vc/Cilk/CUDA).

- Eases code maintenance; requires only one kernel to be written for a given functionality

- Must retain performance of a low level implementation
Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

C-like abstract kernels
Illustrating scalar/SIMD abstraction and kernels

- Single particle interface
- Vector interface
- External CUDA kernels

- Scalar Looper
- Cilk Plus Looper
- Vc Looper
- Thread access

C-like abstract kernels
Illustrating scalar/SIMD abstraction and kernels

- Single particle interface
- Vector interface
- External CUDA kernels

- Scalar Looper
- Cilk Plus Looper
- Vc Looper

- Scalar Backend
- Vc Backend
- Cilk Plus Backend
- CUDA Backend

C-like abstract kernels
Illustrating scalar/SIMD abstraction and kernels

- Single particle interface
  - Scalar Looper
  - scalar Backend

- Vector interface
  - Cilk Plus Looper
  - Cilk Plus Backend

- External CUDA kernels
  - Vc Looper
  - Vc Backend
  - CUDA Backend

- C-like abstract kernels
- C-like specialized kernels

Thread access
- Backend instantiation
- Kernel instantiation
Generic kernels on abstract types

- **Generic programming;** write algorithms in a generic way on abstract types.
- **Implementation depends on the backend.**
- **Backend** determines the types on which the high level operations are performed.
Generic kernels on abstract types

- Generic programming; write algorithms in a **generic** way on **abstract** types
- Implementation depends on the backend
- **Backend** determines the types on which the high level operations are performed
Implementations wrapped in types

- Operations are abstracted
- Backends implement all operations
- Vc already supports arithmetics
- CUDA uses primitive types
- Wrapper structs are implemented if needed
Interfacing with CUDA

- Involving the GPU should be easy

- Principle of common code

- Issue: GPU code needs nvcc, but we want a different compiler for CPU

```cpp
31 CreateRootGeometry();
32 RootManager::Instance().LoadRootGeometry();
33 CudaManager::Instance().LoadGeometry();
34 CudaManager::Instance().Synchronize();
```

**GNU/Clang:**
- C++11
- Vc
- Cilk

**nvcc:**
- CUDA

vs.
Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one or both** environments
- Provide abstraction to interface between environments

namespace vecgeom

Host memory

vecgeom::CudaManager

vecgeom::UnplacedBox
vecgeom::LogicalVolume

... 

namespace vecgeom_cuda

GPU memory

vecgeom_cuda::UnplacedBox
vecgeom_cuda::LogicalVolume

...
Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one** or **both** environments
- Provide abstraction to interface between environments
Dual namespaces

- Utilize distinct namespaces but **same code** for CPU and GPU
- Classes live in **one or both** environments
- Provide abstraction to interface between environments

```
namespace vecgeom
namespace vecgeom_cuda

Host memory
- vecgeom::CudaManager
- vecgeom::UnplacedBox
- vecgeom::LogicalVolume
...

GPU memory
- vecgeom_cuda::UnplacedBox
- vecgeom_cuda::LogicalVolume
...
```

```
Launch kernel
Allocation

CUDA kernel
Instantiate
```
Compilation scheme

Source files .cpp

Optional modules (ROOT, benchmarking...)

Compile for C++11 with vector backend

namespace vecgeom .o

Interface header

Linking

Executable

Source files .cu

CUDA interface .cu

Compile with NVCC with CUDA backend

namespace vecgeom_cuda .o
Status

- Common code for box methods runs for scalar, Vc, Cilk and CUDA backends
- Implemented GPU memory synchronization
- Dual namespaces allows dispatching work to CPU or GPU
- Location can run in either environment
- Results are in agreement

Preliminary benchmarks run in hybrid CPU/GPU environment for four-level box geometry. Location of particles is a tree algorithm, so speedups are only seen at very high particle multiplicity.
**Appendix:** CPU/GPU same interface

Main executable compiled with C++ compiler

CUDA kernel compiled with NVCC
Appendix: Common code

```cpp
template<typename Backend>
VECGeom_INLINE
VECGeom_CUDA_Header_Both

void BoxUnplacedInside( Vector3D<Precision> const &dimensions,
    Vector3D<typename Backend::precision_v> const &localpoint,
    typename Backend::bool_v *const inside )
{
    Vector3D<typename Backend::bool_v> inside_dim(Backend::kFalse);
    for (int i = 0; i < 3; ++i) {
        inside_dim[i] = Abs(localpoint[i]) < dimensions[i];
        if (Backend::early_returns) {
            if (!inside_dim[i]) {
                *inside = Backend::kFalse;
                return;
            }
        }
    }
    if (Backend::early_returns) {
        *inside = Backend::kTrue;
    } else {
        *inside = inside_dim[0] && inside_dim[1] && inside_dim[2];
    }
}
```
Appendix: Template hierarchy
External CUDA kernels

C-like abstract kernels

Vector interface

```cpp
template <TranslationCode trans_code, RotationCode rot_code, 
          typename VolumeType, typename ContainerType>
void Vector3DVectorsPrecisions<VolumeType const &volume,
                          ContainerType const &positions,
                          ContainerType const &directions,
                          Precision const &step_max,
                          Precision const &output> const {
    for (int i = 0; i < positions.size(); ++i) {
        output[i] = 
            volume.template DistanceToZ dispatched <trans_code, rot_code, KScalar[
                                positions[i], directions[i], step_max[i]]
               PRECISION[&step_max[i]],
        result.store(output[i]);
    }
}
```

```cpp
for (int i = 0; i < positions.size(); ++i) {
    const void DistanceToZ dispatched <trans_code, rot_code, KVector[
                                positions[i], directions[i], step_max[i]]
               PRECISION[&step_max[i]],
    result.store(output[i]);
}
```
Vector interface

Scalar Backend

Vc Backend

Cilk Plus Backend

CUDA Backend

External CUDA kernels

C-like abstract kernels