# Bridging the Gap: Writing Portable Programs for CPU and GPU using CUDA

cppcon 2024

*Thomas Mejstrik*, Sebastian Woblistin

## Motivation

- Audience
  - Cuda/C
  - Cuda/C++
  - Fortran
  - HPC
  - Liked the title
  - Disliked other titles
  - Nvidia
  - I do not know where I am
- Only Cuda
- Difference CPU/GPU
- Why it makes sense?

## Motivation

- Audience
- Only Cuda
  - What is Cuda
  - Do not ask me about SYCL, Vulkan, ...
  - You can tell me about
- Difference CPU/GPU
- Why it makes sense?

## Motivation

- Audience
- Only Cuda
- Difference CPU/GPU
  - Latency/Throughput
  - Memory bandwidth
  - Number of cores
  - Handling of branches
  - Cache sizes
  - number formats

  Algorithms are designed differently
- Why it makes sense?

## Motivation

- Audience
- Only Cuda
- Difference CPU/GPU
- Why it makes sense?
  - Embarrassingly parallel algorithms
  - User experience
  - Debugging

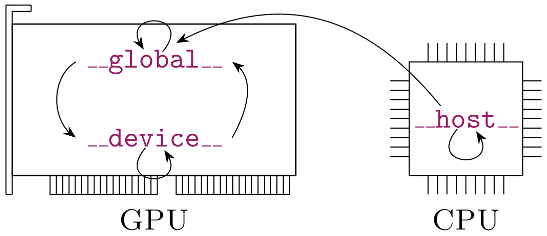
## Hello World without world

```
#include <cstdio>
__device__ int  print()  { return 0; }
__global__ void kernel() { printf( "%i", print() ); }
__host__   void start()  { kernel<<< 2, 3 >>>(); }
int main() {  // implicitly __host__
  start();
  return cudaDeviceSynchronize();
}
```

```
stdout: 000000
return code: 0
```

*nvcc 12.3*

## Allowed function calls in Cuda



If you are still not motivated,
you will not believe what happens next

## Bad cross function calls

```
struct H {
  __host__ int func() { return 42; }
};
struct D {
  __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
    return H{}.func();    //
  //return D{}.func();    //
  //return wrap< H >();   //
  //return wrap< D >();   //
}
```

*nvcc 12.3*

## Bad cross function calls

```
struct H {
  __host__ int func() { return 42; }
};
struct D {
  __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
  //return H{}.func();    // OK
    return D{}.func();    //
  //return wrap< H >();   //
  //return wrap< D >();   //
}
```

*nvcc 12.3*

## Bad cross function calls

```
struct H {
  __host__ int func() { return 42; }
};
struct D {
  __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
  //return H{}.func();    // OK
  //return D{}.func();    // compilation error
    return wrap< H >();   //
  //return wrap< D >();   //
}
```

*nvcc 12.3*

## Bad cross function calls

```
struct H {
  __host__ int func() { return 42; }
};
struct D {
  __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
  //return H{}.func();    // OK
  //return D{}.func();    // compilation error
  //return wrap< H >();   // compilation warning
    return wrap< D >();   //
}
```

*nvcc 12.3*

## Bad cross function calls

```
struct H {
  __host__ int func() { return 42; }
};
struct D {
  __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
  //return H{}.func();    // OK
  //return D{}.func();    // compilation error
  //return wrap< H >();   // compilation warning
  //return wrap< D >();   // no warning, UB at runtime
}
```

*nvcc 12.3*

## Patterns

`__host__ __device__` everything

## __host__ __device__ everything - Solution

```
struct H {
  __host__ __device__ int func() { return 42; }
};
struct D {
  __host__ __device__ int func() { return 666; }
};

template< typename T > __host__ __device__
int wrap() { return T{}.func(); }

int main() {
  // return H{}.func();    // OK
  // return D{}.func();    // OK
  // return wrap< H >();   // OK
  // return wrap< D >();   // OK
}
```

*nvcc 12.3*

## `__host__ __device__` everything - Consequences

- + Easy to use
- ○ May lead to code bloat
- ○ Not always possible

## __host__ __device__ everything - annotations

```
#ifndef CUDATAGS
  #define CUDATAGS
  #ifndef __CUDACC__
    #define __host__
    #define __device__
  #endif
#endif

__host__ __device__
void func() {}
```

## __host__ __device__ everything - annotations

```
#ifndef CUDATAGS
  #define CUDATAGS
  #ifndef __CUDACC__
    #define __host__
    #define __device__
  #endif
#endif

__host__ __device__
void func() {}
```

```
#ifndef CUDATAGS
  #define CUDATAGS
  #ifndef __CUDACC__
    #define HST
    #define DEV
  #else
    #define HST __host__
    #define DEV __device__
  #endif
#endif

HST DEV
void func() {}
```

## Conditional function body

## Conditional function body

- Compilation of Cuda
  - nvcc + host compiler
  - clang
  - HIP / nvc / gpucc / ???
- Language differences
- `__CUDA_ARCH__`

## Conditional function body

- Compilation of Cuda
- Language differences
  - Function signatures
- `__CUDA_ARCH__`

## Conditional function body

- Compilation of Cuda
- Language differences
- `__CUDA_ARCH__`
  - Defined when device code is compiled
  - Restrictions (later)

## Conditional function body

*clang*

```
#include <cstdlib>

__host__ void
r_assert( bool x ) {
  if( !x ) {
    std::abort();
  }
}

__device__ void
r_assert( bool x ) {
  if( !x ) {
    __trap();
  }
}
```

## Conditional function body

*clang*

```
#include <cstdlib>

__host__ void
r_assert( bool x ) {
  if( !x ) {
    std::abort();
  }
}

__device__ void
r_assert( bool x ) {
  if( !x ) {
    __trap();
  }
}
```

*nvcc* + host compiler / clang

```
#include <cstdlib>

__host__ __device__ void
r_assert(  bool x ) {
  if( !x ) {
    #ifndef __CUDA_ARCH__
      std::abort();
    #else
      __trap();
    #endif
  }
}
```

## `__CUDA_ARCH__`

- *The signature of functions, function templates and instantiated function templates, as well as the arguments used to instantiate function templates must not depend on whether `__CUDA_ARCH__` is defined or not*
- `if constexpr` ??

## __CUDA_ARCH__- SKIPPED

```
struct H {
  __host__ void value() {}
};

template< typename T >
__host__ __device__ void func( T t ) { t.value(); }

int main() {
  #ifndef __CUDA_ARCH__ //
    func( H{} );        // UB
  #endif                //
}
```

## `constexpr` everything

## `constexpr` everything

- Context: Function · · · Cuda and non-Cuda compilers · · · host and device side · · · implementation ok · · · `constexpr` · · ·

## `constexpr` everything

- Context: Function ··· Cuda and non-Cuda compilers ··· host and device side ··· implementation ok ··· `constexpr` ···
- Problem: Cannot make changes to code

## `constexpr` everything

- Context: Function ··· Cuda and non-Cuda compilers ··· host and device side ··· implementation ok ··· `constexpr` ···
- Problem: Cannot make changes to code
- Solution: Compile with *nvcc* and `--expt-relaxed-constexpr`

## constexpr everything - 3 known uses

- LBANN uses a defensive strategy: If the source is compiled with `--expt-relaxed-constexpr`, then functions are annotated with `constexpr`, otherwise with `__host__ __device__`

## `constexpr` everything - 3 known uses

- LBANN uses a defensive strategy: If the source is compiled with `--expt-relaxed-constexpr`, then functions are annotated with `constexpr`, otherwise with `__host__ __device__`
- RAPIDS (developed by Nvidia) discussed whether to use `--expt-relaxed-constexpr`, but eventually decided against it

## `constexpr` everything - 3 known uses

- LBANN uses a defensive strategy: If the source is compiled with `--expt-relaxed-constexpr`, then functions are annotated with `constexpr`, otherwise with `__host__ __device__`
- RAPIDS (developed by Nvidia) discussed whether to use `--expt-relaxed-constexpr`, but eventually decided against it
- MatX (developed by Nvidia) uses it

## constexpr everything - Consequences

++ Is also applicable to third party constexpr functions

[1] *github.com/rapidsai/cudf/issues/7795*

## `constexpr` everything - Consequences

++ Is also applicable to third party `constexpr` functions

- + Easy to use
- + Needs minimal changes to the source code

[1] *github.com/rapidsai/cudf/issues/7795*

## `constexpr` everything - Consequences

++ Is also applicable to third party `constexpr` functions
- + Easy to use
- + Needs minimal changes to the source code
- − Only applicable to `constexpr` functions
- − Is an experimental feature ($\leq$ 2016)
- − Future C++ versions?
- − Bad if used in a library

[1] *github.com/rapidsai/cudf/issues/7795*

## `constexpr` everything - Consequences

- ++ Is also applicable to third party `constexpr` functions
- + Easy to use
- + Needs minimal changes to the source code
- – Only applicable to `constexpr` functions
- – Is an experimental feature ($\leq$ 2016)
- – Future C++ versions?
- – Bad if used in a library
- –– May lead to subtle bugs[1]

[1] *github.com/rapidsai/cudf/issues/7795*

## constexpr everything - Failing examples

```
constexpr int foo( int j ) {
  if( j < 0 )  throw;
  return 42;
}
```

becomes on nvcc 12.2 (without compiler warnings)

```
__device__ constexpr int foo( int j ) {
    return 42;
}
```

Jake Hemstad

## constexpr everything - Assessment

```
int bar( int i ) {
  return i * 2;
}

constexpr int foo( int j ) {
  if( j < 0 )  return bar( j );
  return 42;
}
```

becomes on nvcc 12.2 (without compiler warnings)

```
__device__ constexpr int foo( int j ) {
    return 42;
}
```

Jake Hemstad

## constexpr everything - Assessment

```
constexpr int set() {
  auto i = (int*) malloc( sizeof(int) );
  *i = 42;
  int y = *i;
  free( i );
  return y;
}
```

may work, or not, depending on the Cuda version and the system.

## `constexpr` everything - Assessment

```
constexpr int set() {
  auto i = (int*) malloc( sizeof(int) );
  *i =
  int y
  free
  retu
}
```

Takeaway:

*Consider* using `--expt-relaxed-constexpr`

may work, or not, depending on the Cuda version and the system.

## Disable Cuda warnings

## Disable Cuda warnings

- Context: You know everything is ok

## Disable Cuda warnings

- Context: You know everything is ok
- Problem: The compiler does not know everything is ok, and spits out warnings.

## Disable Cuda warnings

- Context: You know everything is ok
- Problem: The compiler does not know everything is ok, and spits out warnings.
- Solution: Disable the warnings

## Disable Cuda warnings: How?

- push - pop pragmas:
  - `nv_diagnostic push`,
  - `nv_diag_suppress`, and
  - `nv_diagnostic pop`
- function scope pragmas
- compiler flags
- `constexpr` everything
- `__host__ __device__` everything

## Disable Cuda warnings: How?

- push - pop pragmas:
- function scope pragmas
  - `#hd_warning_disable` and
  - `#nv_exec_check_disable`
- compiler flags
- `constexpr` everything
- `__host__ __device__` everything

## Disable Cuda warnings: How?

- push - pop pragmas:
- function scope pragmas
- compiler flags
  - `--diag-suppress 20011,20014`
- `constexpr` everything
- `__host__ __device__` everything

## Disable Cuda warnings: How?

- push - pop pragmas:
- function scope pragmas
- compiler flags
- `constexpr` everything
- `__host__ __device__` everything

## Disable Cuda warnings: Consequences

+ Easy to use
◦ Each function has to be annotated manually.
– `#hd_warning_disable` and `#nv_exec_check_disable` pragmas are undocumented, wrong usage may lead to wrongly compiled code[2]
– May hide programming errors. Offensive programming.
–– Future?

[2] *#pragma hd_warning_disable causes nvcc to generate incorrect code (cuda 9.1).*, *forums.developer.nvidia.com/t/57755*.

## Disable Cuda warnings - 3 known uses

- Thrust (Nvidia): `#nv_exec_check_disable`

```
#pragma nv_exec_check_disable
template< typename Policy, typename Iter,
          typename Comp >
__host__ __device__ Iter lower_bound( /* ...*/ );
```

## Disable Cuda warnings - 3 known uses

- Thrust (Nvidia): `#nv_exec_check_disable`
- Eigen: `#nv_exec_check_disable` and `--expt-relaxed-constexpr`

## Disable Cuda warnings - 3 known uses

- Thrust (Nvidia): `#nv_exec_check_disable`
- Eigen: `#nv_exec_check_disable` and `--expt-relaxed-constexpr`
- Dimetor

```
#pragma nv_diag_suppress 20011,20014
#include <Eigen/Core>
#pragma nv_diag_warning 20011,20014
```

## Conditional `__host__ __device__` template

## __CUDA_ARCH__

```
struct H {
  __host__ void value() {}
};

template< typename T >
__host__ __device__ void func( T t ) { t.value(); }

int main() {
  #ifndef __CUDA_ARCH__ //
    func( H{} );        // UB
  #endif                //
}
```

## Cuda proposal

Thank you for listening
Questions welcome