UXL Language SIG Meeting

# SYCL upstreaming strategy

Tom Honermann
2025-02-04

## SYCL front end responsibilities

- Recognize calls to SYCL kernel invocation functions
- Generate and emit SYCL kernel entry point functions
- Identify and emit all functions reachable from SYCL kernels
- Coordinate with the SYCL run-time library to marshal SYCL kernels and their arguments from the host to a device
  - With custom marshaling protocols for SYCL special types
- Diagnostics
  - SYCL kernel name requirements
  - SYCL kernel parameter requirements
  - Use of restricted language features in device functions

## DPC++ SYCL implementation strategy

- Multiple pass compilation model
  - One pass for host compilation
  - One pass for each device compilation
- Integration headers and footers
  - Used to coordinate host and device compilation
    - SYCL kernel entry point function name
    - SYCL kernel parameter information
    - SYCL kernel size
    - SYCL kernel source location information
    - SYCL specialization constants
  - Files generated during device compilation
  - Files pre-included during host compilation
  - Enables use of host compilers that are not SYCL aware

## Clang community compromise

- No integration headers or footers
  - Host compilation must be SYCL aware
  - No support for third party SYCL unaware host compilers (for now)
- No new impediments to future one-pass compilation models
  - A multiple-pass compilation model is ok
- No implementation-detail functions in the AST

## DPC++ SYCL kernel interface

```
template<typename KernelNameType, typename KernelType>
[[clang::sycl_kernel]]
void sycl_kernel_entry_point(
    KernelType kernel [[maybe_unused]])
{
#if defined(__SYCL_DEVICE_ONLY__)
  kernel();
#endif
}
```

## Front end changes relative to DPC++

- The **`sycl_kernel`** attribute will be deprecated and replaced by a new **`sycl_kernel_entry_point(kernel-name)`** attribute
- No need for an option to enable or disable reduced feature set modes (e.g., for support of unnamed lambda kernels)
- AST changes
- Name changes for generated SYCL kernel caller functions
- A new suite of builtin functions for use by a SYCL run-time library to query SYCL kernel properties

## Kernel attribute changes

**sycl_kernel**

- Only appertains to function templates
- Infers the kernel name type from a function template signature
- Requires an empty function body for host compilation

**sycl_kernel_entry_point**

- Appertains to functions and function templates
- Requires the kernel name type be passed as an argument
- Defines behavior for host function invocation (a no-op)

# AST changes relative to DPC++

- DPC++ emits a generated function in the AST thus exposing implementation-detail (function name, etc…)
  - See the `_ZTSZ4mainE1K` function in the device AST at [https://godbolt.org/z/j6MYYdddY](https://godbolt.org/z/j6MYYdddY)
- Upstream will instead augment the body of the SYCL kernel entry point function using two new AST nodes:
  - **`OutlinedFunctionDecl`** describes the SYCL kernel caller function to be emitted. The declaration includes the function parameters (potentially decomposed as required for a SYCL kernel) and a function body that reconstitutes the original SYCL entry point function parameters. The declaration omits other details like a function name or calling convention
  - **`SYCLKernelCallStmt`** wraps the original body of the SYCL entry point function and the outlined function definition for the SYCL kernel caller function

# AST example

```
template<typename KernelNameType, typename KernelType>
[[clang::sycl_kernel_entry_point(KernelNameType)]]
void sycl_kernel_entry_point(KernelType kernel) {
  kernel();
}
struct Kernel {
  int dm1, dm2;
  void operator()() const;
};
void f(Kernel k) {
  sycl_kernel_entry_point<class kernel_name>(k);
}
```

# AST example

```
FunctionDecl 'sycl_kernel_entry_point<kernel_name>(Kernel)'
  TemplateArgument type 'kernel_name'
  TemplateArgument type 'Kernel'
  ParmVarDecl kernel 'Kernel'
  SYCLKernelCallStmt
    CompoundStmt
      <original statements>
    OutlinedFunctionDecl
      ImplicitParamDecl 'dm1' 'int'
      ImplicitParamDecl 'dm2' 'int'
      CompoundStmt
        VarDecl 'kernel' 'Kernel'
          <initialization of 'kernel' with 'dm1' and 'dm2'>
        <transformed statements with redirected references of 'kernel'>
```

# Naming changes relative to DPC++

- The SYCL kernel caller function name will change
- DPC++ misappropriates the typeinfo special name from the Itanium ABI:
  - `_ZTS11kernel_name`
  - `typeinfo name for kernel_name`
- Upstream will use a function template specialization with a reserved name:
  - `_Z20__sycl_kernel_callerI11kernel_nameEvv`
  - `void __sycl_kernel_caller<kernel_name>()`
- The `sycl_unique_stable_name` builtin function will be deprecated and removed

## Builtin functions

- Kernel information will not be exposed by integration headers, so…

```
__builtin_sycl_kernel_name(kernel-name) -> const char*
__builtin_sycl_kernel_param_count(kernel-name) -> int
__builtin_sycl_kernel_param_kind(kernel-name, integer) -> int
__builtin_sycl_kernel_param_size(kernel-name, integer) -> int
__builtin_sycl_kernel_param_offset(kernel-name, integer) -> int
__builtin_sycl_kernel_param_access_target(kernel-name, integer) -> int
__builtin_sycl_kernel_file_name(kernel-name) -> const char*
__builtin_sycl_kernel_function_name(kernel-name) -> const char*
__builtin_sycl_kernel_line_number(kernel-name) -> unsigned int
__builtin_sycl_kernel_column_number(kernel-name) -> unsigned int
__builtin_sycl_kernel_size(kernel-name) -> size_t
```

## SYCL specification concerns

- #454: Correspondence of unnamed lambdas as kernels across host and device compilation
- #543: What does "Exception-handling cannot be used inside a device function" actually mean?
- #568: Can the types used for kernel names be cv-qualified?
- #603: The term "kernel" is used for multiple purposes
- #612: Can pointer to data member types be used in device code? Are they device copyable?
- #629: Types in the `std` namespace do not need to be prohibited as kernel names due to forward declaration requirements

## SYCL upstreaming status

- Three PRs have been accepted and merged:
  - PR 111389: [SYCL] The sycl_kernel_entry_point attribute.
  - PR 120327: [SYCL] Basic diagnostics for the sycl_kernel_entry_point attribute.
  - PR 122379: [SYCL] AST support for SYCL kernel entry point functions.
- Changes are being staged for one additional PR:
  - [SYCL] Offload kernel code generation for SYCL kernel entry point functions.
- Documentation: https://clang.llvm.org/docs/AttributeReference.html#sycl-kernel-entry-point

## SYCL upstreaming status

- In progress:
  - Codegen support for SPIR-V, NVPTX, and AMDGCN
  - Support for builtin functions
  - Warnings for kernel name types that are not forward declarable
- Awaiting implementation:
  - Kernel argument decomposition and restitution
  - Support for SYCL special types
  - Address space annotation for kernel parameters
  - Diagnostics for language restrictions in device functions
  - Support for SYCL specialization constants

intel®