# Automatic and Customizable Code Rewriting and Refactoring with Clang

**Alister Johnson** University of Oregon

Johannes Doerfert Argonne National Lab

Disclaimer: This video reflects mid-October project status, changes may have since been made.

(Also presented at LLVM-HPC at SC21.)

## Motivating Example: CUDA to HIP

```
  cudaMalloc(&dev_array, num_bytes);
→ hipMalloc(&dev_array, num_bytes);
```

- Simple: search-and-replace `cudaMalloc` with `hipMalloc`

# Motivating Example: CUDA to HIP

```
   cudaMalloc(&dev_array, num_bytes);
→  hipMalloc(&dev_array, num_bytes);
```

- Simple: search-and-replace `cudaMalloc` with `hipMalloc`

```
   custom_kernel<<<(N+255)/256, 256>>>(N, dev_array);
→  hipLaunchKernelGGL(custom_kernel, (N+255)/256, 256, 0, 0, N, dev_array);
```

- Complex: could possibly use sed or awk, but those are not optimal

## Motivating Example: CUDA to HIP

```
[[clang::matcher("kernel_call")]]
auto cuda_kernel(int numBlocks, int numThreads) {
     auto arg1, arg2;
     {
          kernel<<<numBlocks, numThreads>>>(arg1, arg2);
     }
}
```

# Motivating Example: CUDA to HIP

```
[[clang::matcher("kernel_call")]]
auto cuda_kernel(int numBlocks, int numThreads) {
     auto arg1, arg2;
     {
          kernel<<<numBlocks, numThreads>>>(arg1, arg2);
     }
}

[[clang::replace("kernel_call")]]
auto hip_kernel(int numBlocks, int numThreads) {
     auto arg1, arg2;
     {
          hipLaunchKernelGGL(kernel, numBlocks, numThreads, 0, 0, arg1, arg2);
     }
}
```

## This Work

- A prototype (source-to-source) code rewriting tool built on Clang
- A user interface for this tool that is user-friendly and customizable
- An example implementation of `hipify` using this tool (in progress)

## User Interface

- Pure C++ for simple learning curve
- New C++ attributes to define two sets of functions:
  - Matchers - describe which code to rewrite
    - `[[clang::matcher("<matcher_name>")]]`
  - Modifiers - describe how to rewrite matched code
    - `[[clang::replace("<list of matchers>")]]`
    - `[[clang::insert_before("<list of matchers>")]]`
    - `[[clang::insert_after("<list of matchers>")]]`
- Implemented with Clang's AST matcher and Rewriter interfaces

## Use Cases

- API updates
- Switching APIs or programming models (e.g., CUDA to HIP)
- Adding (selective) instrumentation
- Device-specific code (e.g., OpenMP pragmas or kernels)
- Error-checking asserts
- Checkpointing
- Performance portability layers[1]
- And many more!

[1]J. K. Holmen, B. Peterson and M. Berzins, "An Approach for Indirectly Adopting a Performance Portability Layer in Large Legacy Codes," P3HPC 2019.