MLIR open meeting

[RFC] Adding support for OpenMP GPU target offload

Fabian Mora-Cordero
fmorac@udel.edu
U. of Delaware

Thursday 15th February, 2024

Background

MLIR's GPU compilation infrastructure: serialization

- Target attributes determine how to compile GPU modules
- There is compilation support for NVIDIA `#nvvm.target`, AMD `#rocdl.target` and Intel `#spirv.target`
- With some caveats, the same GPU module can be compiled for different vendors. GPU binaries can hold objects from any target.

```
1  gpu.module @moduleName  [
2    #nvvm.target<chip = "sm_90", libs=[
3      "libomptarget-nvptx-sm_90.bc"
4    ]>,
5    #rocdl.target<chip = "gfx90a", libs=[
6      "libomptarget-amdgpu-gfx90a.bc"
7    ]>,
8  ] {
9   ...
10 }
11 // mlir-opt --gpu-module-to-binary
12 gpu.binary @moduleName [
13   #gpu.object<
14     #nvvm.target<chip = "sm_90">, "Binary blob"
15   >,
16   #gpu.object<
17     #rocdl.target<chip = "gfx90a">, "Binary blob"
18   >
19 ]
```

Listing: GPU compilation operations and attributes

MLIR's GPU compilation infrastructure: serialization

- Target attributes determine how to compile GPU modules
- There is compilation support for NVIDIA `#nvvm.target`, AMD `#rocdl.target` and Intel `#spirv.target`
- With some caveats, the same GPU module can be compiled for different vendors. GPU binaries can hold objects from any target.

```
1  gpu.module @moduleName  [
2    #nvvm.target<chip = "sm_90", libs=[
3      "libomptarget-nvptx-sm_90.bc"
4    ]>,
5    #rocdl.target<chip = "gfx90a", libs=[
6      "libomptarget-amdgpu-gfx90a.bc"
7    ]>,
8  ] {
9   ...
10 }
11 // mlir-opt --gpu-module-to-binary
12 gpu.binary @moduleName [
13   #gpu.object<
14     #nvvm.target<chip = "sm_90">, "Binary blob"
15   >,
16   #gpu.object<
17     #rocdl.target<chip = "gfx90a">, "Binary blob"
18   >
19 ]
```

Listing: GPU compilation operations and attributes

MLIR's GPU compilation infrastructure: serialization

- Target attributes determine how to compile GPU modules
- There is compilation support for NVIDIA `#nvvm.target`, AMD `#rocdl.target` and Intel `#spirv.target`
- With some caveats, the same GPU module can be compiled for different vendors. GPU binaries can hold objects from any target.

```
1  gpu.module @moduleName  [
2    #nvvm.target<chip = "sm_90", libs=[
3      "libomptarget-nvptx-sm_90.bc"
4    ]>,
5    #rocdl.target<chip = "gfx90a", libs=[
6      "libomptarget-amdgpu-gfx90a.bc"
7    ]>,
8  ] {
9   ...
10 }
11 // mlir-opt --gpu-module-to-binary
12 gpu.binary @moduleName [
13   #gpu.object<
14     #nvvm.target<chip = "sm_90">, "Binary blob"
15   >,
16   #gpu.object<
17     #rocdl.target<chip = "gfx90a">, "Binary blob"
18   >
19 ]
```

Listing: GPU compilation operations and attributes

MLIR's GPU compilation infrastructure: embedding

- Offloading attributes determine how to translate binaries and kernel launches
- `#gpu.select_object` is the only offload attribute upstream
  - It supports embedding only one binary in the host module

```
1  gpu.binary @kernels  [#gpu.object<#nvvm.target,
       offload = "BIN">]
2  llvm.func @main() {
3    %0 = llvm.mlir.constant(1 : index) : i64
4    gpu.launch_func  @kernels::@hello blocks in
         (%0, %0, %0) threads in (%0, %0, %0) : i64
5    llvm.return
6  }
7  // mlir-translate --mlir-to-llvmir
8  @kernels_bin_cst = internal constant [3 x i8] c"
       BIN", align 8
9  @kernels_hello_kernel_name = private unnamed_addr
       constant [6 x i8] c"hello\00", align 1
10 define void @main() {
11   %3 = call ptr @mgpuModuleLoad(ptr
         @kernels_bin_cst, i64 3)
12   %4 = call ptr @mgpuModuleGetFunction(ptr %3,
         ptr @kernels_hello_kernel_name)
13   call void @mgpuLaunchKernel(%4, ...)
14   call void @mgpuModuleUnload(ptr %3)
15   ret void
16 }
```

Listing: Translation of GPU operations

MLIR's GPU compilation infrastructure: embedding

- Offloading attributes determine how to translate binaries and kernel launches
- `#gpu.select_object` is the only offload attribute upstream
  - It supports embedding only one binary in the host module

```
1  gpu.binary @kernels  [#gpu.object<#nvvm.target,
       offload = "BIN">]
2  llvm.func @main() {
3    %0 = llvm.mlir.constant(1 : index) : i64
4    gpu.launch_func  @kernels::@hello blocks in
         (%0, %0, %0) threads in (%0, %0, %0) : i64
5    llvm.return
6  }
7  // mlir-translate --mlir-to-llvmir
8  @kernels_bin_cst = internal constant [3 x i8] c"
       BIN", align 8
9  @kernels_hello_kernel_name = private unnamed_addr
       constant [6 x i8] c"hello\00", align 1
10 define void @main() {
11   %3 = call ptr @mgpuModuleLoad(ptr
         @kernels_bin_cst, i64 3)
12   %4 = call ptr @mgpuModuleGetFunction(ptr %3,
         ptr @kernels_hello_kernel_name)
13   call void @mgpuLaunchKernel(%4, ...)
14   call void @mgpuModuleUnload(ptr %3)
15   ret void
16 }
```

Listing: Translation of GPU operations

MLIR's GPU compilation infrastructure: embedding

- Offloading attributes determine how to translate binaries and kernel launches
- `#gpu.select_object` is the only offload attribute upstream
  - It supports embedding only one binary in the host module

```
1  gpu.binary @kernels  [#gpu.object<#nvvm.target,
       offload = "BIN">]
2  llvm.func @main() {
3    %0 = llvm.mlir.constant(1 : index) : i64
4    gpu.launch_func  @kernels::@hello blocks in
         (%0, %0, %0) threads in (%0, %0, %0) : i64
5    llvm.return
6  }
7  // mlir-translate --mlir-to-llvmir
8  @kernels_bin_cst = internal constant [3 x i8] c"
       BIN", align 8
9  @kernels_hello_kernel_name = private unnamed_addr
       constant [6 x i8] c"hello\00", align 1
10 define void @main() {
11   %3 = call ptr @mgpuModuleLoad(ptr
         @kernels_bin_cst, i64 3)
12   %4 = call ptr @mgpuModuleGetFunction(ptr %3,
         ptr @kernels_hello_kernel_name)
13   call void @mgpuLaunchKernel(%4, ...)
14   call void @mgpuModuleUnload(ptr %3)
15   ret void
16 }
```

Listing: Translation of GPU operations

Proposal

Offload embedding attribute

- `#gpu.offload_embedding` a new offload attribute, PR: #78117
- Instead of loading the binaries and kernels every time, everything gets registered into a runtime at startup
- The CUDA, HIP, and LibOMPTarget runtimes become usable, PR: #78116
- The CUDA runtime provides automatic context management, and it's interoperable with the driver

```
1  gpu.binary @kernels
2    <#gpu.offload_embedding<CUDA>>
3    [#gpu.object<#nvvm.target, offload = "BIN">]
4  llvm.func @main() { ... }
5  // mlir-translate --mlir-to-llvmir
6  @__dev_image = ... [3 x i8] c"BIN" ...
7  @__kernel_id = weak constant i8 0
8  @__kernel_name = ... [6 x i8] c"hello\00"
9  @__bin_descriptor = internal constant ...
10 @llvm.global_ctors = ... [@__register_fn]
11 define void @__register_fn() {
12  call void @__register_lib(
13             ptr @__bin_descriptor)
14  ret void
15 }
16 define void @main() {
17  call void @mgpuLaunchKernel(@__kernel_id, ...)
18  ret void
19 }
```

Listing: Translation using the offload embedding attribute

Offload embedding attribute

- `#gpu.offload_embedding` a new offload attribute, PR: #78117
- Instead of loading the binaries and kernels every time, everything gets registered into a runtime at startup
- The CUDA, HIP, and LibOMPTarget runtimes become usable, PR: #78116
- The CUDA runtime provides automatic context management, and it's interoperable with the driver

```
1  gpu.binary @kernels
2    <#gpu.offload_embedding<CUDA>>
3    [#gpu.object<#nvvm.target, offload = "BIN">]
4  llvm.func @main() { ... }
5  // mlir-translate --mlir-to-llvmir
6  @__dev_image = ... [3 x i8] c"BIN" ...
7  @__kernel_id = weak constant i8 0
8  @__kernel_name = ... [6 x i8] c"hello\00"
9  @__bin_descriptor = internal constant ...
10 @llvm.global_ctors = ... [@__register_fn]
11 define void @__register_fn() {
12  call void @__register_lib(
13             ptr @__bin_descriptor)
14  ret void
15 }
16 define void @main() {
17  call void @mgpuLaunchKernel(@__kernel_id, ...)
18  ret void
19 }
```

Listing: Translation using the offload embedding attribute

Offload embedding attribute

- `#gpu.offload_embedding` a new offload attribute, PR: #78117
- Instead of loading the binaries and kernels every time, everything gets registered into a runtime at startup
- The CUDA, HIP, and LibOMPTarget runtimes become usable, PR: #78116
- The CUDA runtime provides automatic context management, and it's interoperable with the driver

```
1  gpu.binary @kernels
2    <#gpu.offload_embedding<CUDA>>
3    [#gpu.object<#nvvm.target, offload = "BIN">]
4  llvm.func @main() { ... }
5  // mlir-translate --mlir-to-llvmir
6  @__dev_image = ... [3 x i8] c"BIN" ...
7  @__kernel_id = weak constant i8 0
8  @__kernel_name = ... [6 x i8] c"hello\00"
9  @__bin_descriptor = internal constant ...
10 @llvm.global_ctors = ... [@__register_fn]
11 define void @__register_fn() {
12  call void @__register_lib(
13             ptr @__bin_descriptor)
14  ret void
15 }
16 define void @main() {
17  call void @mgpuLaunchKernel(@__kernel_id, ...)
18  ret void
19 }
```

Listing: Translation using the offload embedding attribute

Offload embedding attribute

- `#gpu.offload_embedding` a new offload attribute, PR: #78117
- Instead of loading the binaries and kernels every time, everything gets registered into a runtime at startup
- The CUDA, HIP, and LibOMPTarget runtimes become usable, PR: #78116
- The CUDA runtime provides automatic context management, and it's interoperable with the driver

```
1  gpu.binary @kernels
2    <#gpu.offload_embedding<CUDA>>
3    [#gpu.object<#nvvm.target, offload = "BIN">]
4  llvm.func @main() { ... }
5  // mlir-translate --mlir-to-llvmir
6  @__dev_image = ... [3 x i8] c"BIN" ...
7  @__kernel_id = weak constant i8 0
8  @__kernel_name = ... [6 x i8] c"hello\00"
9  @__bin_descriptor = internal constant ...
10 @llvm.global_ctors = ... [@__register_fn]
11 define void @__register_fn() {
12  call void @__register_lib(
13             ptr @__bin_descriptor)
14  ret void
15 }
16 define void @main() {
17  call void @mgpuLaunchKernel(@__kernel_id, ...)
18  ret void
19 }
```

Listing: Translation using the offload embedding attribute

[RFC] #76312 Adding support for OpenMP GPU target offload

- Main point: Enable GPU compilation for OMP target constructs
- The OpenMPIRBuilder is still used; the proposal is only about adding a compilation driver
- Addition of an outlining pass for `omp.target` ops similar to "gpu-kernel-outlining", PR: #78328
- This would allow testing the OMP dialect within MLIR, JIT-ting OMP offload code, mixing GPU and OMP, and developing the OMP dialect independently from flang and clang
- Link: GH gist with a real-world example

```
1  gpu.module @ompModule ... {
2   func.func @main_outlined(...) {
3    omp.target ... {
4     // Target region
5     omp.terminator
6    }
7   }
8  }
9  func.func @main(...) {
10  omp.target ... {
11   // Target region
12   omp.terminator
13  }
14 }
```

Listing: MLIR with host and offload modules. Instead of having 2 MLIR files (host & dev), everything is embedded in a single file.

[RFC] #76312 Adding support for OpenMP GPU target offload

- Main point: Enable GPU compilation for OMP target constructs
- The OpenMPIRBuilder is still used; the proposal is only about adding a compilation driver
- Addition of an outlining pass for `omp.target` ops similar to "gpu-kernel-outlining", PR: #78328
- This would allow testing the OMP dialect within MLIR, JIT-ting OMP offload code, mixing GPU and OMP, and developing the OMP dialect independently from flang and clang
- Link: GH gist with a real-world example

```
1  gpu.module @ompModule ... {
2   func.func @main_outlined(...) {
3    omp.target ... {
4     // Target region
5     omp.terminator
6    }
7   }
8  }
9  func.func @main(...) {
10  omp.target ... {
11   // Target region
12   omp.terminator
13  }
14 }
```

Listing: MLIR with host and offload modules. Instead of having 2 MLIR files (host & dev), everything is embedded in a single file.

[RFC] #76312 Adding support for OpenMP GPU target offload

- Main point: Enable GPU compilation for OMP target constructs
- The OpenMPIRBuilder is still used; the proposal is only about adding a compilation driver
- Addition of an outlining pass for `omp.target` ops similar to "gpu-kernel-outlining", PR: #78328
- This would allow testing the OMP dialect within MLIR, JIT-ting OMP offload code, mixing GPU and OMP, and developing the OMP dialect independently from flang and clang
- Link: GH gist with a real-world example

```
1  gpu.module @ompModule ... {
2   func.func @main_outlined(...) {
3    omp.target ... {
4     // Target region
5     omp.terminator
6    }
7   }
8  }
9  func.func @main(...) {
10  omp.target ... {
11   // Target region
12   omp.terminator
13  }
14 }
```

Listing: MLIR with host and offload modules. Instead of having 2 MLIR files (host & dev), everything is embedded in a single file.

[RFC] #76312 Adding support for OpenMP GPU target offload

- Main point: Enable GPU compilation for OMP target constructs
- The OpenMPIRBuilder is still used; the proposal is only about adding a compilation driver
- Addition of an outlining pass for `omp.target` ops similar to "gpu-kernel-outlining", PR: #78328
- This would allow testing the OMP dialect within MLIR, JIT-ting OMP offload code, mixing GPU and OMP, and developing the OMP dialect independently from flang and clang
- Link: GH gist with a real-world example

```
1  gpu.module @ompModule ... {
2   func.func @main_outlined(...) {
3    omp.target ... {
4     // Target region
5     omp.terminator
6    }
7   }
8  }
9  func.func @main(...) {
10  omp.target ... {
11   // Target region
12   omp.terminator
13  }
14 }
```

Listing: MLIR with host and offload modules. Instead of having 2 MLIR files (host & dev), everything is embedded in a single file.

[RFC] #76312 Adding support for OpenMP GPU target offload

- Main point: Enable GPU compilation for OMP target constructs
- The OpenMPIRBuilder is still used; the proposal is only about adding a compilation driver
- Addition of an outlining pass for `omp.target` ops similar to "gpu-kernel-outlining", PR: #78328
- This would allow testing the OMP dialect within MLIR, JIT-ting OMP offload code, mixing GPU and OMP, and developing the OMP dialect independently from flang and clang
- Link: GH gist with a real-world example

```
1  gpu.module @ompModule ... {
2   func.func @main_outlined(...) {
3    omp.target ... {
4     // Target region
5     omp.terminator
6    }
7   }
8  }
9  func.func @main(...) {
10  omp.target ... {
11   // Target region
12   omp.terminator
13  }
14 }
```

Listing: MLIR with host and offload modules. Instead of having 2 MLIR files (host & dev), everything is embedded in a single file.

Future work

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

LLVM project offload

- Is an initiative proposed by Johannes Doerfert to make an official LLVM offload runtime, RFC: #74302
- The starting point is LibOMPTarget, and it will be transformed into a vendor-agnostic runtime API for GPU constructs
- The plan is to support NVIDIA, AMD, and Intel
- It could allow multi-vendor fat binaries
- It would allow JIT-compiling for AMD targets
- `#gpu.offload_embedding` is the starting point for supporting it in MLIR
- Eventually, we should consider dropping our GPU vendor wrappers in favor of LLVM offload

Questions?

Acknowledgments

- This material is based upon work supported by the National Science Foundation (NSF) under grant no. 1814609.
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.
- This research was also supported by BNL via ECP SOLLVE internship.