SPIR-V*: Default Interface to Intel® Graphics Compiler for OpenCL™ Workloads

Intel® Graphics Compiler has recently switched from SPIR* to SPIR-V* as an intermediate representation for OpenCL™ workloads. This may seem like an internal change in the compiler, not visible to the user, but it shows our commitment to support Khronos* open standards. Now most of the compute workloads downloaded to the compiler will use SPIR-V.

The benefits of using SPIR-V in OpenCL applications and how to start working with it today are outlined below.

SPIR-V* – A Khronos* Intermediate Representation for Graphic and Compute Workloads

Intermediate representations (IR) play a big role in modern compiler architectures. They bring the human-readable source language such as C++, GLSL or OpenCL* C closer to the machine representation, yet they are still abstract. IR's enable compilers to perform common optimizations independently of the target machine specifics, or GPU generation in case of the Intel Graphics Compiler.

Intel Graphics Compiler is part of Intel® Graphics Driver and was designed to consume such intermediate representation produced by various front-ends (e.g., Vulkan*, OpenGL*, DirectX*) as its input and produce machine code for given GPU generation on output.

SPIR-V is a modern binary low-level intermediate language supported by Khronos and used in many APIs, like Vulkan, OpenCL and OpenGL.[1] In the OpenCL world, it is a successor to SPIR[2] language, which is based on LLVM-IR.

To get an overview of SPIR-V design please check SPIR-V whitepaper.

As the SPIR-V is a binary format, it was not intended to be read by humans. Nevertheless, let's look at a dis-assembled simple OpenCL kernel that's using the cl_intel_subgroups extension:

OpenCL C Kernel:

```
#pragma OPENCL EXTENSION cl_intel_subgroups : enable

__kernel void testKernel(__global uint *shuffle_results) {
        size_t tid = get_global_id(0);
        uint value = get_sub_group_local_id();
        uint new_value = intel_sub_group_shuffle(value, 1);
        shuffle_results[tid] = new_value;
}
```

Generated SPIR-V:

[1] https://www.khronos.org/registry/spir-v/
[2] https://www.khronos.org/registry/SPIR/

```
; SPIR-V
; Version: 1.0
; Generator: Khronos LLVM/SPIR-V Translator; 14
; Bound: 36
; Schema: 0
               OpCapability Addresses
               OpCapability Linkage
               OpCapability Kernel
               OpCapability Int64
               OpCapability SubgroupShuffleINTEL
          %1 = OpExtInstImport "OpenCL.std"
               OpMemoryModel Physical64 OpenCL
               OpEntryPoint Kernel %12 "testKernel"
         %32 = OpString "kernel_arg_type.testKernel.uint*,"
               OpSource OpenCL_C 200000
               OpName %__spirv_BuiltInGlobalInvocationId
"__spirv_BuiltInGlobalInvocationId"
               OpName %__spirv_BuiltInSubgroupLocalInvocationId
"__spirv_BuiltInSubgroupLocalInvocationId"
               OpName %shuffle_results "shuffle_results"
               OpName %shuffle_results_addr "shuffle_results.addr"
               OpName %tid "tid"
               OpName %value "value"
               OpName %new_value "new_value"
               OpName %call "call"
               OpName %call1 "call1"
               OpName %call2 "call2"
               OpName %arrayidx "arrayidx"
               OpDecorate %33 Constant
         %33 = OpDecorationGroup
               OpDecorate %34 Alignment 4
         %34 = OpDecorationGroup
               OpDecorate %35 Alignment 8
         %35 = OpDecorationGroup
               OpDecorate %__spirv_BuiltInGlobalInvocationId BuiltIn
GlobalInvocationId
               OpDecorate %__spirv_BuiltInSubgroupLocalInvocationId BuiltIn
SubgroupLocalInvocationId
               OpDecorate %__spirv_BuiltInGlobalInvocationId LinkageAttributes
"__spirv_BuiltInGlobalInvocationId" Import
               OpDecorate %__spirv_BuiltInSubgroupLocalInvocationId LinkageAttributes
"__spirv_BuiltInSubgroupLocalInvocationId" Import
               OpGroupDecorate %33 %__spirv_BuiltInGlobalInvocationId
%__spirv_BuiltInSubgroupLocalInvocationId
               OpGroupDecorate %34 %value %new_value
               OpGroupDecorate %35 %shuffle_results_addr %tid
      %ulong = OpTypeInt 64 0
       %uint = OpTypeInt 32 0
     %uint_1 = OpConstant %uint 1
    %v3ulong = OpTypeVector %ulong 3
%_ptr_UniformConstant_v3ulong = OpTypePointer UniformConstant %v3ulong
%_ptr_UniformConstant_uint = OpTypePointer UniformConstant %uint
       %void = OpTypeVoid
%_ptr_CrossWorkgroup_uint = OpTypePointer CrossWorkgroup %uint
         %11 = OpTypeFunction %void %_ptr_CrossWorkgroup_uint
```

```
%_ptr_Function__ptr_CrossWorkgroup_uint = OpTypePointer Function
%_ptr_CrossWorkgroup_uint
%_ptr_Function_ulong = OpTypePointer Function %ulong
%_ptr_Function_uint = OpTypePointer Function %uint
%__spirv_BuiltInGlobalInvocationId = OpVariable %_ptr_UniformConstant_v3ulong
UniformConstant
%__spirv_BuiltInSubgroupLocalInvocationId = OpVariable %_ptr_UniformConstant_uint
UniformConstant
         %12 = OpFunction %void DontInline %11
%shuffle_results = OpFunctionParameter %_ptr_CrossWorkgroup_uint
         %14 = OpLabel
%shuffle_results_addr = OpVariable %_ptr_Function__ptr_CrossWorkgroup_uint Function
        %tid = OpVariable %_ptr_Function_ulong Function
      %value = OpVariable %_ptr_Function_uint Function
  %new_value = OpVariable %_ptr_Function_uint Function
               OpStore %shuffle_results_addr %shuffle_results Aligned 8
         %22 = OpLoad %v3ulong %__spirv_BuiltInGlobalInvocationId
       %call = OpCompositeExtract %ulong %22 0
               OpStore %tid %call Aligned 8
      %call1 = OpLoad %uint %__spirv_BuiltInSubgroupLocalInvocationId
               OpStore %value %call1 Aligned 4
         %25 = OpLoad %uint %value Aligned 4
      %call2 = OpSubgroupShuffleINTEL %uint %25 %uint_1
               OpStore %new_value %call2 Aligned 4
         %28 = OpLoad %uint %new_value Aligned 4
         %29 = OpLoad %_ptr_CrossWorkgroup_uint %shuffle_results_addr Aligned 8
         %30 = OpLoad %ulong %tid Aligned 8
   %arrayidx = OpInBoundsPtrAccessChain %_ptr_CrossWorkgroup_uint %29 %30
               OpStore %arrayidx %28 Aligned 4
               OpReturn
               OpFunctionEnd
```

As you can see, it’s not very convenient to read or analyze, but as we’ll discuss later, that’s actually a benefit.

Benefits of SPIR-V for OpenCL™ App Developers

Many benefits listed in the whitepaper target driver vendors like Intel, but there are some that we think are worth considering for OpenCL application vendors:

- Once SPIR-V kernel is generated, the compilation time of it is much faster compared to text kernel (OpenCL C). There’s no need for text-based parsing on user machine, so your application will start faster.
- Protected Intellectual Property is not shipped with the application. When shipped with OpenCL C text kernels, it is easy to figure out what algorithms to use. When the kernels are written in SPIR-V, it requires reverse engineering.
- SPIR-V is closer to the hardware. To tune the performance of the app, do it in a way that is not possible on the OpenCL C level.
- It is easily extendable. A researcher or a compiler enthusiast might like to test some new ideas to speed up the application, by taking the open source Intel Graphics Compiler and play with extended SPIR-V code.

- The SPIR-V specification and the ecosystem is being actively developed by Khronos. There are tools for validation, assembly, disassembly, conversion and optimization libraries available on GitHub*, e.g., SPIRV-Tools, SPIRV-Cross, SPIRV-LLVM-Translator.

SPIR-V* in Intel® Graphics Driver

Let's take a high-level look of how OpenCL kernel is compiled in Intel Graphic Driver.
In OpenCL 2.1 there are many ways to provide source code from the kernel to the driver:

ClCreateProgramWithBinary

The binary is fully dependent on the device that it was created on -- the application doesn't create it. That means that binaries are not portable to other platforms. This is probably mostly used to cache kernels by the application.

You can also use cl_khr_spir extension to provide a kernel in SPIR format. SPIR standard supports OpenCL C up to 2.0 and was replaced by SPIR-V in next releases.

ClCreateProgramWithSource

This is the most common path. The kernel source is provided in text form, kernels are written in OpenCL C high-level language. The compilation flow in Intel Graphic Driver looks like this:

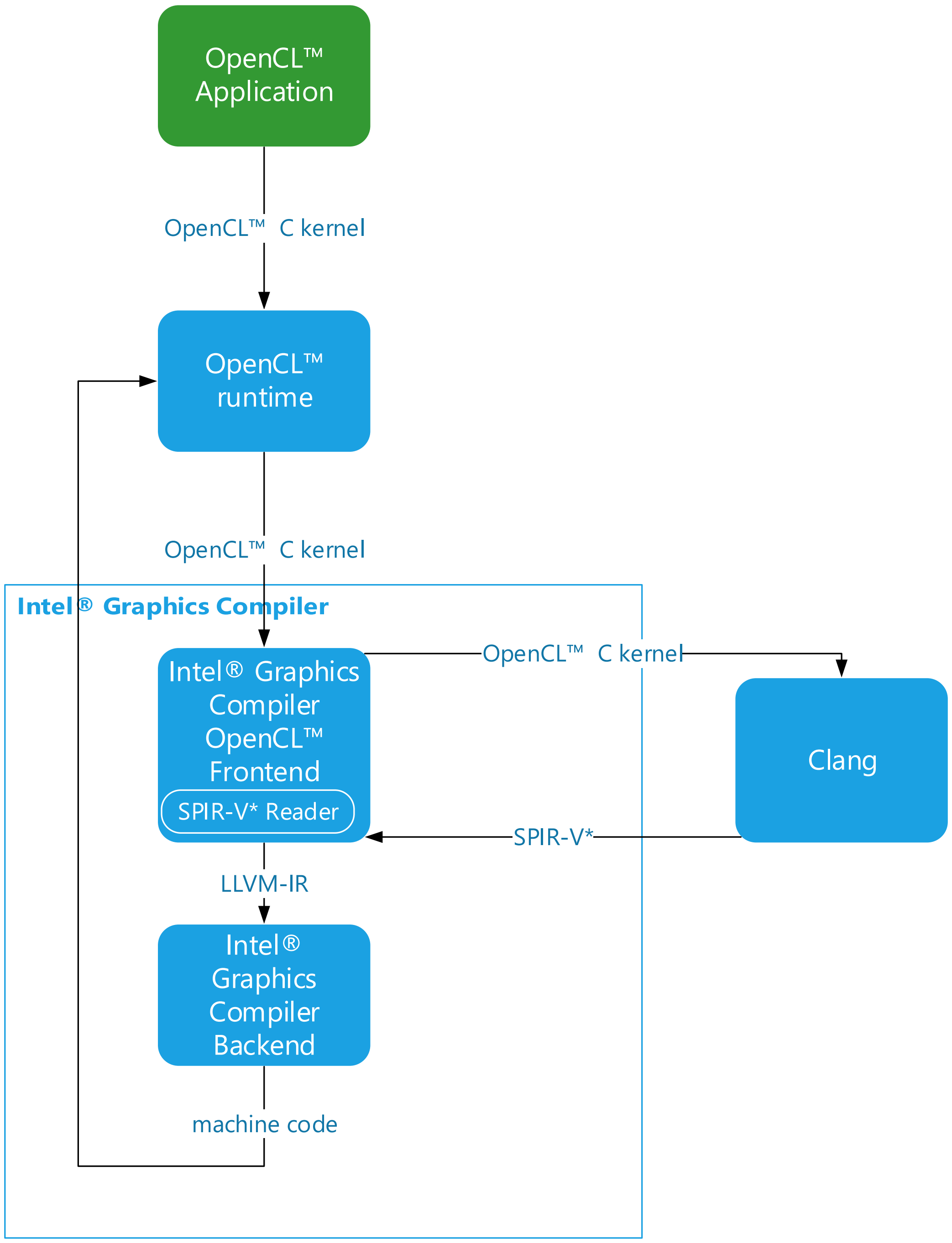

Figure 1: Compilation of a kernel written in OpenCL C

Inside the Intel Graphics Compiler, Clang needs to be called to translate OpenCL C kernel to SPIR-V. Then Intel Graphics Compiler OpenCL Frontend translates it to LLVM-IR that the compiler uses internally to produce hardware instructions (machine code).

ClCreateProgramWithIL.
This call provides implementation defined intermediate language (IL), which would still not be portable as in ClCreateProgramWithBinary, or use SPIR-V – portable IL defined by Khronos.

As you can see, preparing the SPIR-V when shipping your application can save compilation time on client machines – Clang stage is not needed here:

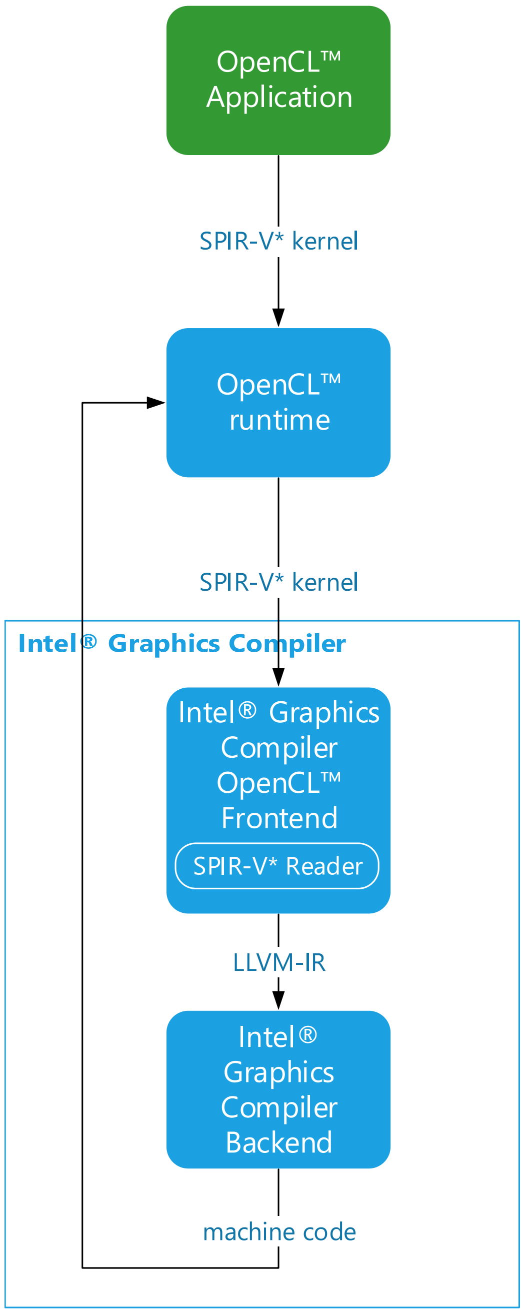

*Figure 2. Compilation of a kernel written in SPIR-V**

How to Generate SPIR-V for Existing OpenCL Kernels

As shown before, in the driver stack the SPIR-V is generated by Clang. A “standalone” version of Clang can be downloaded from Khronos Group’s GitHub*.

After successfully built, generating SPIR-V is easy:

```
clang -cc1 -emit-spirv -triple=spir64-unknown-unknown -cl-std=CL2.0 -include opencl.h
kernel.cl -o kernel.spir
```

Then use this kernel in the app with clCreateProgramWithIL OpenCL API call.

How to Hand-tune the SPIR-V* Kernel

SPIR-V is a binary format, so an assembler and dis-assembler are needed.

You can download them here:

The basic steps are:

1. Disassemble your SPIR-V kernel with `spirv-dis`
2. Edit the disassembly
3. Assemble again with `spirv-as`
4. Validate it with `spirv-val`

Conclusion

SPIR-V is supported in Intel Graphics Compiler since the introduction of OpenCL 2.1 and Vulkan 1.0 API. In Vulkan SPIR-V is the only shader representation we support. In OpenCL 2.1, Use OpenCL C or SPIR-V.

The majority of OpenCL applications use OpenCL C and as shown before, need to be translated to some intermediate representation. We recently moved from SPIR to SPIR-V for this task. This is a way to make SPIR-V tested and validated even more thoroughly and shows our commitment to support Khronos standards.

Resources

Intel Graphics Compiler
SPIRV-Tools
SPIRV-Cross
SPIRV-LLVM-Translator