SPIR-V and its place in the LLVM ecosystem

ARM

Neil Hickey neil.hickey@arm.com - ARM

Jakub (Kuba) Kuderski kubakuderski@gmail.com- PUT

EuroLLVM'17

March 2017

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

Khronos history

- Founded in 2000
- Manages specifications of GPU languages, e.g.
  - OpenGL
  - OpenCL
  - OpenGL|ES
  - Vulkan

KHRONOS™ GROUP

CONNECTING SOFTWARE TO SILICON

ARM

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

ARM

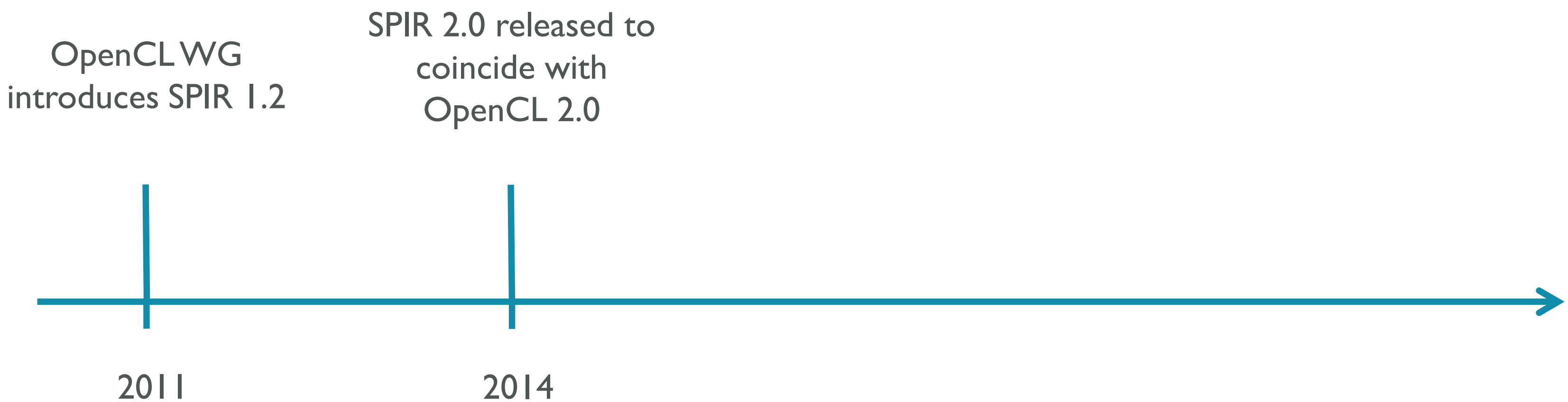
SPIR History

ARM

SPIR History

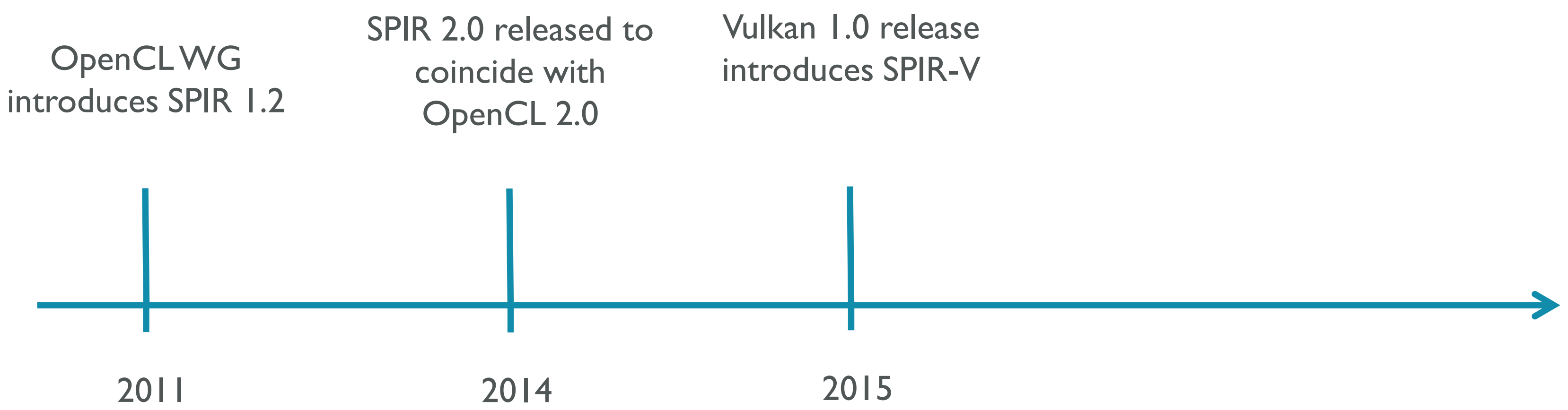
ARM

SPIR History

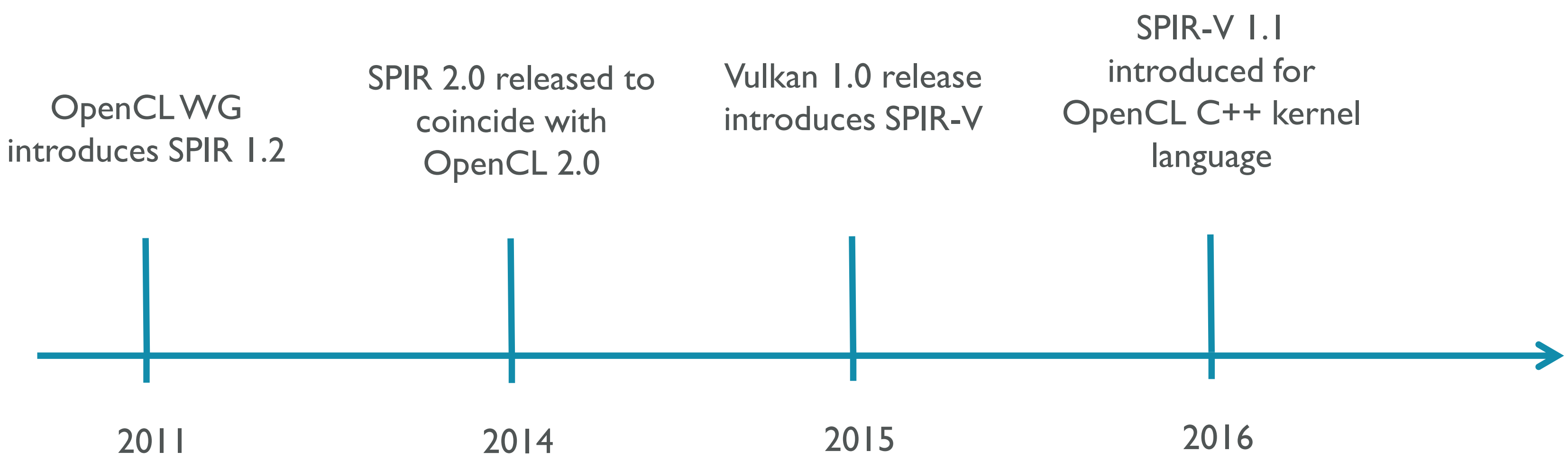


ARM

SPIR History



SPIR History



ARM

SPIR-V purpose

- Introduced by Khronos to be a common intermediate language across Khronos IP domains
  - Intermediate language
  - Feature set closely tied to Vulkan and OpenCL
  - Not designed to be coded by hand
  - Requires decoding step to uncover algorithm
  - Allows for new, novel, shader and compute languages

SPIR-V ecosystem

Input language

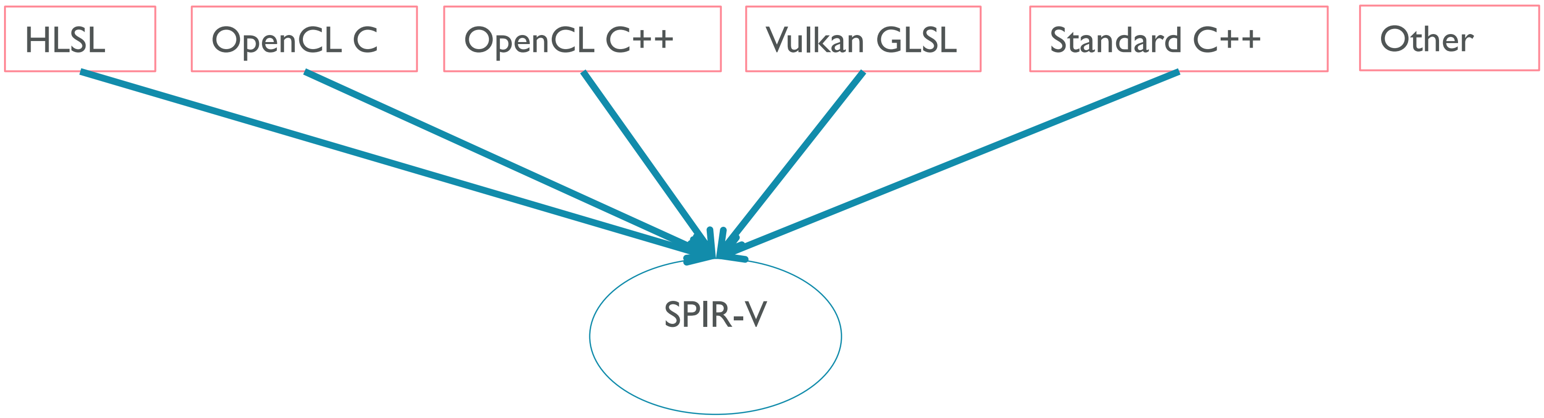
HLSL	OpenCL C	OpenCL C++	Vulkan GLSL	Standard C++	Other

ARM

SPIR-V ecosystem

Input language

Open source or Proprietary toolchains



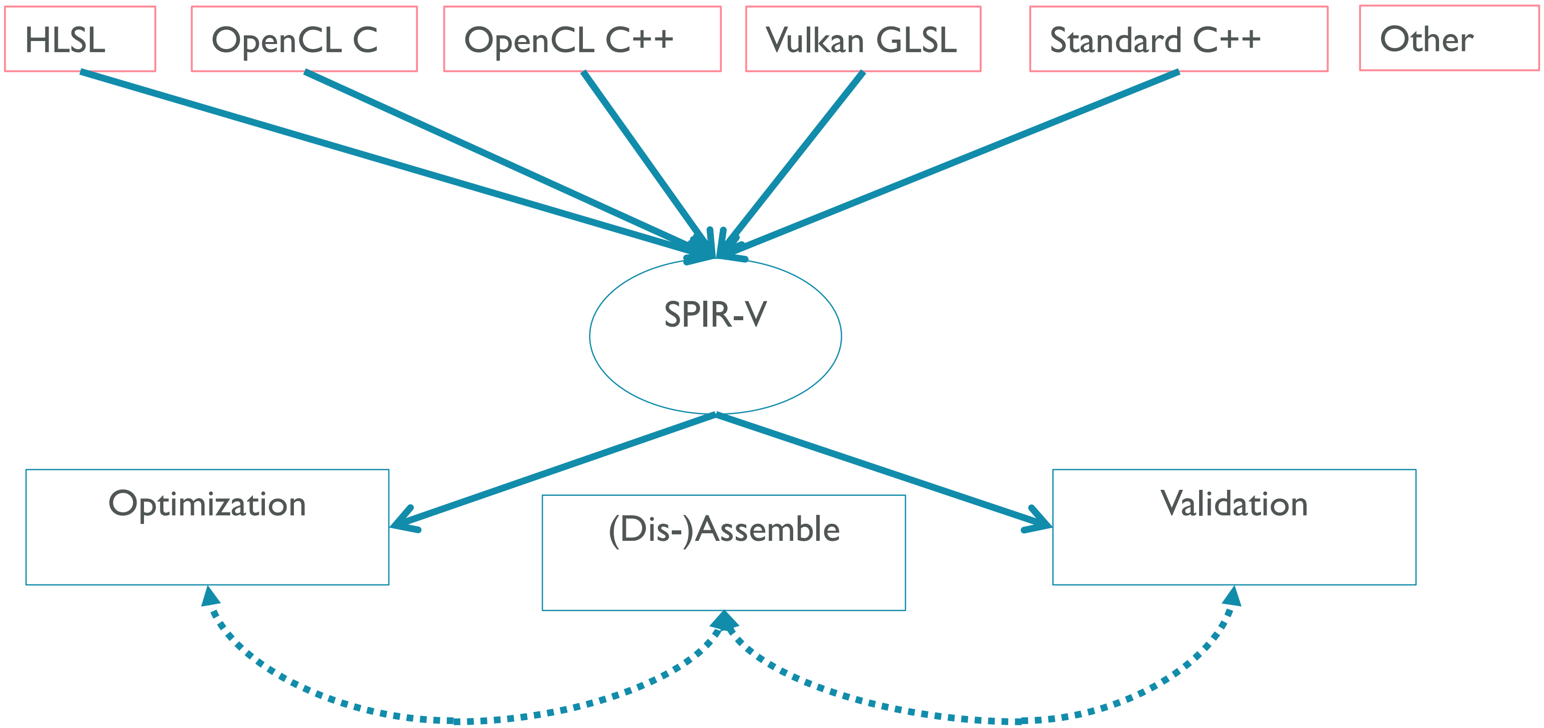
ARM

SPIR-V ecosystem

Input language

Open source or Proprietary toolchains

Target independent optimisation and validation.



ARM

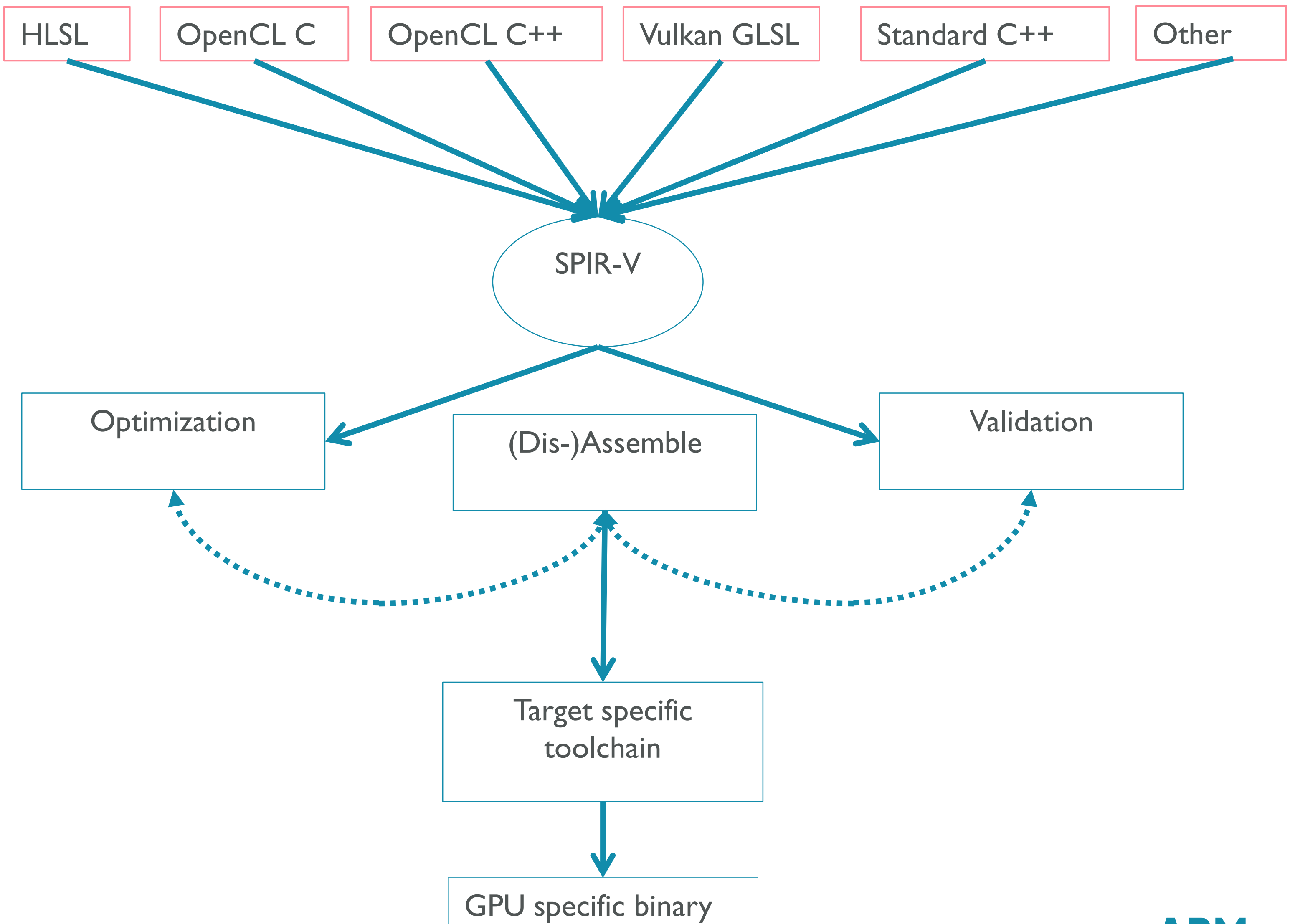
SPIR-V ecosystem

Input language

Open source or Proprietary toolchains

Target independent optimisation and validation.

GPU vendor proprietary



Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

Sample SPIR-V Structure

```
#version 450
in vec4 color1;
in vec4 multiplier;
noperspective in vec4 color2;
out vec4 color;
struct S {
     bool b;
     vec4 v[5];
     int i;
};

uniform blockName {
     S s;
     bool cond;
};
void main()
{
     vec4 scale = vec4(1.0, 1.0, 2.0,
1.0);
     if (cond)
          color = color1 + s.v[2];
     else
          color = sqrt(color2)*scale;
     for (int i = 0; i < 4; ++i)
          color*= multiplier;
}
```

Corresponding SPIR-V – simple ☺

```
; Magic:     0x07230203 (SPIR-V)
; Version:   0x00010000 (Version: 1.0.0)
; Generator: 0x00080001 (Khronos Glslang Reference Front End; 1)
; Bound:     63
; Schema:    0

               OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Fragment %4 "main" %31 %33 %42 %57
               OpExecutionMode %4 OriginLowerLeft

; Debug information
               OpSource GLSL 450
               OpName %4 "main"
               OpName %9 "scale"
               OpName %17 "S"
               OpMemberName %17 0 "b"
               OpMemberName %17 1 "v"
               OpMemberName %17 2 "i"
               OpName %18 "blockName"
               OpMemberName %18 0 "s"
               OpMemberName %18 1 "cond"
               OpName %20 ""
               OpName %31 "color"
               OpName %33 "color1"
               OpName %42 "color2"
               OpName %48 "i"
               OpName %57 "multiplier"

; Annotations (non-debug)
               OpDecorate %15 ArrayStride 16
               OpMemberDecorate %17 0 Offset 0
               OpMemberDecorate %17 1 Offset 16
               OpMemberDecorate %17 2 Offset 96
               OpMemberDecorate %18 0 Offset 0
               OpMemberDecorate %18 1 Offset 112
               OpDecorate %18 Block
               OpDecorate %20 DescriptorSet 0
               OpDecorate %42 NoPerspective

; All types, variables, and constants
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2                      ; void ()
          %6 = OpTypeFloat 32                         ; 32-bit
float
          %7 = OpTypeVector %6 4                      ; vec4
          %8 = OpTypePointer Function %7              ; function-
local vec4*
         %10 = OpConstant %6 1
         %11 = OpConstant %6 2
         %12 = OpConstantComposite %7 %10 %10 %11 %10 ; vec4(1.0,
1.0, 2.0, 1.0)
         %13 = OpTypeInt 32 0                         ; 32-bit
int, sign-less
         %14 = OpConstant %13 5
         %15 = OpTypeArray %7 %14
         %16 = OpTypeInt 32 1
         %17 = OpTypeStruct %13 %15 %16
         %18 = OpTypeStruct %17 %13
         %19 = OpTypePointer Uniform %18
         %20 = OpVariable %19 Uniform
         %21 = OpConstant %16 1
         %22 = OpTypePointer Uniform %13
         %25 = OpTypeBool
         %26 = OpConstant %13 0
         %30 = OpTypePointer Output %7
         %31 = OpVariable %30 Output
         %32 = OpTypePointer Input %7
         %33 = OpVariable %32 Input
         %35 = OpConstant %16 0
         %36 = OpConstant %16 2
         %37 = OpTypePointer Uniform %7
         %42 = OpVariable %32 Input
         %47 = OpTypePointer Function %16
         %55 = OpConstant %16 4
         %57 = OpVariable %32 Input

; All functions
          %4 = OpFunction %2 None %3                  ; main()
          %5 = OpLabel
          %9 = OpVariable %8 Function
         %48 = OpVariable %47 Function
               OpStore %9 %12
         %23 = OpAccessChain %22 %20 %21              ; location
of cond
         %24 = OpLoad %13 %23                         ; load 32-
bit int from cond
         %27 = OpINotEqual %25 %24 %26                ; convert to
bool
               OpSelectionMerge %29 None              ; structured
if
               OpBranchConditional %27 %28 %41        ; if cond
         %28 = OpLabel                                ; then
         %34 = OpLoad %7 %33
         %38 = OpAccessChain %37 %20 %35 %21 %36      ; s.v[2]
         %39 = OpLoad %7 %38
         %40 = OpFAdd %7 %34 %39
               OpStore %31 %40
               OpBranch %29
         %41 = OpLabel                                ; else
         %43 = OpLoad %7 %42
         %44 = OpExtInst %7 %1 Sqrt %43               ; extended
instruction sqrt
         %45 = OpLoad %7 %9
         %46 = OpFMul %7 %44 %45
               OpStore %31 %46
               OpBranch %29
         %29 = OpLabel                                ; endif
               OpStore %48 %35
               OpBranch %49
         %49 = OpLabel
               OpLoopMerge %51 %52 None               ; structured
loop
               OpBranch %53
         %53 = OpLabel
         %54 = OpLoad %16 %48
         %56 = OpSLessThan %25 %54 %55                ; i < 4 ?
               OpBranchConditional %56 %50 %51        ; body or
break
         %50 = OpLabel                                ; body
         %58 = OpLoad %7 %57
         %59 = OpLoad %7 %31
         %60 = OpFMul %7 %59 %58
               OpStore %31 %60
               OpBranch %52
         %52 = OpLabel                                ; continue
target
         %61 = OpLoad %16 %48
         %62 = OpIAdd %16 %61 %21                     ; ++i
               OpStore %48 %62
               OpBranch %49                           ; loop back
         %51 = OpLabel                                ; loop merge
point
               OpReturn
               OpFunctionEnd
```


SPIR-V Capabilities

- Specifies what parts of the full SPIR-V spec this particular binary will use.
- Used by validation tools to make sure of correctness.

```
; Magic:     0x07230203 (SPIR-V)
; Version:   0x00010000 (Version: 1.0.0)
; Generator: 0x00080001 (Khronos Glslang Reference Front End; 1)
; Bound:     63
; Schema:    0

                OpCapability Shader
                OpCapability Int64
                OpCapability Float16
```

Memory model, Addressing model

- Memory and addressing model specified directly

```
%1 = OpExtInstImport "GLSL.std.450"
     OpMemoryModel Logical GLSL450
     OpEntryPoint Fragment %4 "main" %31 %33 %42 %57
     OpExecutionMode %4 OriginLowerLeft
```

- Logical addressing model. Pointers are abstract, pointers to pointers are not allowed
- Non-logical address model allow physical pointers to be created.
  - Physical32 or Physical64
- GLSL450 Memory model needed by later versions of GLSL ad ESSL, other options currently: Simple – no consistency semantics, OpenCL – OpenCL memory model

Entry point, Execution mode and model

- Shader and kernel entry points specified directly, not using metadata as in llvm

```
OpEntryPoint Fragment %4 "main" %31 %33 %42 %57
OpExecutionMode %4 OriginLowerLeft
```

  - Specified how the entry point should be treated.
  - Represents the different shader stages in Vulkan: Tessalation, Geometry, Vertex, Fragment or the different compute options, GLCompute/ Kernel – for OpenCL.
  - ExecutionMode specifies extra semantic information about the mode an entry point will execute in. E.g. LocalSize for Kernel execution mode.

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

Structured control flow

- Vulkan shaders require structure control flow.
  - Loops have a single exit block and a single back edge
  - No gotos.
- OpenCL only requires reducible control flow
- LLVM optimizations can change the layout to produce unstructured or irreducible control flow

Example irreducible control flow

- Duff's device

```
switch(b%8) {
   case 0: do { *dst++ = *src++;
   case 7:      *dst++ = *src++;
   case 6:      *dst++ = *src++;
   case 5:      *dst++ = *src++;
   case 4:      *dst++ = *src++;
   case 3:      *dst++ = *src++;
   case 2:      *dst++ = *src++;
   case 1:      *dst++ = *src++;
    } while (--b>0);
}
```

Reliance on metadata

- OpenCL relies on metadata to express semantics, SPIR-V represents those semantics as SPIR-V opcodes.

Reliance on metadata

- OpenCL relies on metadata to express semantics, SPIR-V represents those semantics as SPIR-V opcodes.

```
__kernel void entry(
__global int *inout_3
)
{
    inout_3[get_global_id(0)] = get_global_id(0);
}
```

Reliance on metadata

- OpenCL relies on metadata to express semantics, SPIR-V represents those semantics as SPIR-V opcodes.

```
; Function Attrs: nounwind
define spir_kernel void @entry(i32 addrspace(1)* %inout_3) #0 {
  %call = call spir_func i64 @_Z13get_global_idj(i32 0) #1
  %conv = trunc i64 %call to i32
  %call1 = call spir_func i64 @_Z13get_global_idj(i32 0) #1
  %arrayidx = getelementptr inbounds i32, i32 addrspace(1)* %inout_3, i64 %call1
  store i32 %conv, i32 addrspace(1)* %arrayidx, align 4
  ret void
}

!opencl.kernels = !{!0}
!llvm.ident = !{!8}

!0 = !{void (i32 addrspace(1)*)* @entry, !1, !2, !3, !4, !5, !6, !7}
!1 = !{!"kernel_arg_addr_space", i32 1}
!2 = !{!"kernel_arg_access_qual", !"none"}
!3 = !{!"kernel_arg_type", !"int*"}
!4 = !{!"kernel_arg_base_type", !"int*"}
!5 = !{!"kernel_arg_type_qual", !""}
!6 = !{!"kernel_arg_name", !"inout_3"}
!7 = !{!"attrs", !""}
!8 = !{!"clang version 3.9.0 "}
```

Reliance on metadata

- OpenCL relies on metadata to express semantics, SPIR-V represents those semantics as SPIR-V opcodes.

```
; SPIR-V
; Version: 1.0
; Generator: Khronos LLVM/SPIR-V Translator;
14
; Bound: 19
; Schema: 0
               OpCapability Addresses
               OpCapability Linkage
               OpCapability Kernel
               OpCapability Int64
          %1 = OpExtInstImport "OpenCL.std"
               OpMemoryModel Physical64 OpenCL
               OpEntryPoint Kernel %10 "entry"
               OpSource OpenCL_C 102000
               OpName %5
"__spirv_BuiltInGlobalInvocationId"
               OpName %11 "inout_3"
               OpName %14 "call"
               OpName %15 "conv"
               OpName %17 "call1"
               OpName %18 "arrayidx"
               OpDecorate %5 BuiltIn
GlobalInvocationId
               OpDecorate %5 Constant
               OpDecorate %5 LinkageAttributes
"__spirv_BuiltInGlobalInvocationId" Import
          %2 = OpTypeInt 64 0
          %7 = OpTypeInt 32 0
          %3 = OpTypeVector %2 3
          %4 = OpTypePointer UniformConstant
%3
          %6 = OpTypeVoid
          %8 = OpTypePointer CrossWorkgroup %7
          %9 = OpTypeFunction %6 %8
          %5 = OpVariable %4 UniformConstant
         %10 = OpFunction %6 None %9
         %11 = OpFunctionParameter %8
         %12 = OpLabel
         %13 = OpLoad %3 %5
         %14 = OpCompositeExtract %2 %13 0
         %15 = OpUConvert %7 %14
         %16 = OpLoad %3 %5
         %17 = OpCompositeExtract %2 %16 0
         %18 = OpInBoundsPtrAccessChain %8 %11
%17
               OpStore %18 %15 Aligned 4
               OpReturn
               OpFunctionEnd
```


ARM

Uniform control flow

- SPIR-V has native opcodes to represent barrier operation and cross workgroup / subgroup operations
- LLVM introduces convergence function attribute to represent a similar concept

Composible types

- Built up using multiple SPIR-V “instructions”
- Example

```
struct MyStruct { int4 a; float arr[3]; };
```

```
%uint                = OpTypeInt 32 0
%uint_3              = OpConstant %uint 3
%v3uint              = OpTypeVector %uint 3
%v4uint              = OpTypeVector %uint 4
%float               = OpTypeFloat 32
%_arr_float_uint_3 = OpTypeArray %float %uint_3
%struct_MyStruct     = OpTypeStruct %v4uint %_arr_float_uint_3
```

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

SPIR-V Tooling

- SPIRV-Tools – open source tools by Khronos:
  - spirv-opt – optimizer
  - spirv-as – assembler
  - spirv-dis – disassembler
  - spirv-val – validator
  - spirv-cfg – Control Flow Graph viewer

- glslang - reference shader compiler by Khronos
  - Shader compiler (graphics and compute)
  - Binary ←→ textual representation conversions
  - spirv-remap – improves readablility of SPIR-V files by assigning similar ids to similar opcodes

SPIR-V Tooling

Optimizations currently implemented in spirv-opt:

- Strip debug info
- Set spec constant default value
- Freeze spec constant
- Fold OpSpecConstantOp and OpSpecConstantComposite
- Unify constants
- Eliminate dead constants

ARM

SPIRV-LLVM converter

- Conversion from LLVM to SPIR-V and from SPIR-V to LLVM
- OpenCL-only
- Forked and put inside the LLVM tree in llvm/lib/SPIRV
- Based on LLVM 3.6 (+ unfinished version based on 3.8)

ARM

SPIRV-LLVM converter – OpenCL to LLVM

```
__kernel void test(__global int* in1, __global int* in2, __global int* out) {
  int id = get_global_id(0);
  out[id] = in1[id] * in2[id];
}
```

```
; ModuleID = 'hw.cl'
target datalayout = "e-p:32:32-i64:64-v16:16-v24:32-v32:32-v48:64-v96:128-
v192:256-v256:256-v512:512-v1024:1024-n8:16:32:64"
target triple = "spir-unknown-unknown"

; Function Attrs: nounwind
define spir_kernel void @test(i32 addrspace(1)* nocapture readonly %in1, i32
addrspace(1)* nocapture readonly %in2, i32 addrspace(1)* nocapture %out) #0 {
entry:
  %call = tail call spir_func i32 @_Z13get_global_idj(i32 0) #2
  %arrayidx = getelementptr inbounds i32 addrspace(1)* %in1, i32 %call
  %0 = load i32 addrspace(1)* %arrayidx, align 4, !tbaa !10
  %arrayidx1 = getelementptr inbounds i32 addrspace(1)* %in2, i32 %call
  %1 = load i32 addrspace(1)* %arrayidx1, align 4, !tbaa !10
  %mul = mul nsw i32 %1, %0
  %arrayidx2 = getelementptr inbounds i32 addrspace(1)* %out, i32 %call
  store i32 %mul, i32 addrspace(1)* %arrayidx2, align 4, !tbaa !10
  ret void
}
```


SPIRV-LLVM converter – LLVM to SPIR-V

```
119734787 65536 393230 22 0
2 Capability Addresses
2 Capability Linkage
2 Capability Kernel
5 ExtInstImport 1 "OpenCL.std"
3 MemoryModel 1 2
5 EntryPoint 6 9 "test"
3 Source 3 200000
11 Name 5 "__spirv_BuiltInGlobalInvocationId"
3 Name 10 "in1"
3 Name 11 "in2"
3 Name 12 "out"
4 Name 13 "entry"
4 Name 15 "call"
5 Name 16 "arrayidx"
5 Name 18 "arrayidx1"
3 Name 20 "mul"
5 Name 21 "arrayidx2"
4 Decorate 5 BuiltIn 28
3 Decorate 5 Constant
13 Decorate 5 LinkageAttributes
"__spirv_BuiltInGlobalInvocationId" Import
4 TypeInt 2 32 0
4 TypeVector 3 2 3
4 TypePointer 4 0 3
```

```
2 TypeVoid 6
4 TypePointer 7 5 2
6 TypeFunction 8 6 7 7 7
4 Variable 4 5 0

5 Function 6 9 0 8
3 FunctionParameter 7 10
3 FunctionParameter 7 11
3 FunctionParameter 7 12

2 Label 13
4 Load 3 14 5
5 CompositeExtract 2 15 14 0
5 InBoundsPtrAccessChain 7 16 10 15
6 Load 2 17 16 2 4
5 InBoundsPtrAccessChain 7 18 11 15
6 Load 2 19 18 2 4
5 IMul 2 20 17 19
5 InBoundsPtrAccessChain 7 21 12 15
5 Store 21 20 2 4
1 Return

1 FunctionEnd
```


ARM

SPIRV-LLVM converter – SPIR-V back to LLVM

Original

```
; ModuleID = 'hw.cl'
target datalayout = "e-p:32:32-i64:64-v16:16-v24:32-v32:32-v48:64-v96:128-v192:256-v256:256-v512:512-v1024:1024-n8:16:32:64"
target triple = "spir-unknown-unknown"

; Function Attrs: nounwind
define spir_kernel void @test(i32 addrspace(1)* nocapture readonly %in1, i32 addrspace(1)* nocapture readonly %in2, i32 addrspace(1)* nocapture %out) #0 {
entry:
  %call = tail call spir_func i32 @_Z13get_global_idj(i32 0) #2
  %arrayidx = getelementptr inbounds i32 addrspace(1)* %in1, i32 %call
  %0 = load i32 addrspace(1)* %arrayidx, align 4, !tbaa !10
  %arrayidx1 = getelementptr inbounds i32 addrspace(1)* %in2, i32 %call
  %1 = load i32 addrspace(1)* %arrayidx1, align 4, !tbaa !10
  %mul = mul nsw i32 %1, %0
  %arrayidx2 = getelementptr inbounds i32 addrspace(1)* %out, i32 %call
  store i32 %mul, i32 addrspace(1)* %arrayidx2, align 4, !tbaa !10
  ret void
}

; Function Attrs: nounwind readnone
declare spir_func i32 @_Z13get_global_idj(i32) #1

attributes #0 = { nounwind "less-precise-fpmad"="false" "no-frame-pointer-elim"="false" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-realign-stack" "stack-protector-buffer-size"="8" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { nounwind readnone "less-precise-fpmad"="false" "no-frame-pointer-elim"="false" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-realign-stack" "stack-protector-buffer-size"="8" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #2 = { nounwind readnone }
```

After roundtrip

```
; ModuleID = 'hw.cl'
target datalayout = "e-p:32:32-i64:64-v16:16-v24:32-v32:32-v48:64-v96:128-v192:256-v256:256-v512:512-v1024:1024"
target triple = "spir-unknown-unknown"

; Function Attrs: nounwind
define spir_kernel void @test(i32 addrspace(1)* nocapture %in1, i32 addrspace(1)* nocapture %in2, i32 addrspace(1)* nocapture %out) #0 {
entry:
  %call = call spir_func i32 @_Z13get_global_idj(i32 0) #1
  %arrayidx = getelementptr inbounds i32 addrspace(1)* %in1, i32 %call
  %0 = load i32 addrspace(1)* %arrayidx, align 4
  %arrayidx1 = getelementptr inbounds i32 addrspace(1)* %in2, i32 %call
  %1 = load i32 addrspace(1)* %arrayidx1, align 4
  %mul = mul i32 %1, %0
  %arrayidx2 = getelementptr inbounds i32 addrspace(1)* %out, i32 %call
  store i32 %mul, i32 addrspace(1)* %arrayidx2, align 4
  ret void
}

; Function Attrs: nounwind readnone
declare spir_func i32 @_Z13get_global_idj(i32) #1

attributes #0 = { nounwind }
attributes #1 = { nounwind readnone }
```


ARM

SPIR-V community

- Public GitHub group with numerous repositories
- Khronos public forums
- Khronos Group paid membership
- Conference calls

SPIR-V community – textual representations

glslang

```
                    2:                TypeVoid
                    3:                TypeFunction 2
                    6:                TypeInt 32 0
                    7:                TypePointer Function 6(int)
                    9:                TypeVector 6(int) 3
                   10:                TypePointer Input 9(ivec3)
11(gl_GlobalInvocationID):             10(ptr) Variable Input
                   12:        6(int) Constant 0
                   13:                TypePointer Input 6(int)
                   16:                TypeFloat 32
                   17:                TypeVector 16(float) 4
                   18:                TypeRuntimeArray 17(fvec4)
           19(Output):                TypeStruct 18
                   20:                TypePointer Uniform 19(Output)
      21(output_data):       20(ptr) Variable Uniform
                   22:                TypeInt 32 1
                   23:       22(int) Constant 0
                   25:                TypeRuntimeArray 17(fvec4)
           26(Input0):                TypeStruct 25
                   27:                TypePointer Uniform 26(Input0)
      28(input_data0):       27(ptr) Variable Uniform
                   30:                TypePointer Uniform 17(fvec4)
                   33:                TypeRuntimeArray 17(fvec4)
           34(Input1):                TypeStruct 33
                   35:                TypePointer Uniform 34(Input1)
      36(input_data1):       35(ptr) Variable Uniform
                   42:        6(int) Constant 128
                   43:        6(int) Constant 1
                   44:      9(ivec3) ConstantComposite 42 43 43
             4(main):             2 Function None 3
```

SPIR-V community – textual representations

SPIRV-LLVM

```
5 EntryPoint 6 9 "test"
3 Source 3 200000
11 Name 5 "__spirv_BuiltInGlobalInvocationId"
3 Name 10 "in1"
3 Name 11 "in2"
3 Name 12 "out"
4 Name 13 "entry"
4 Name 15 "call"
5 Name 16 "arrayidx"
5 Name 18 "arrayidx1"
3 Name 20 "mul"
5 Name 21 "arrayidx2"
4 Decorate 5 BuiltIn 28
3 Decorate 5 Constant
13 Decorate 5 LinkageAttributes
"__spirv_BuiltInGlobalInvocationId" Import
4 TypeInt 2 32 0
4 TypeVector 3 2 3
4 TypePointer 4 0 3
2 TypeVoid 6
4 TypePointer 7 5 2
6 TypeFunction 8 6 7 7 7
4 Variable 4 5 0

5 Function 6 9 0 8
3 FunctionParameter 7 10
3 FunctionParameter 7 11
3 FunctionParameter 7 12
```

SPIR-V community – textual representations

SPIRV-tools

```
; SPIR-V
; Version: 1.0
; Generator: Khronos LLVM/SPIR-V Translator; 14
; Bound: 14
; Schema: 0
               OpCapability Addresses
               OpCapability Kernel
          %1 = OpExtInstImport "OpenCL.std"
               OpMemoryModel Physical32 OpenCL
               OpEntryPoint Kernel %6 "test"
               OpSource OpenCL_C 200000
               OpName %in "in"
               OpName %out "out"
               OpName %entry "entry"
               OpName %arrayidx "arrayidx"
               OpName %arrayidx1 "arrayidx1"
       %uint = OpTypeInt 32 0
     %uint_0 = OpConstant %uint 0
       %void = OpTypeVoid
%_ptr_CrossWorkgroup_uint = OpTypePointer CrossWorkgroup %uint
          %5 = OpTypeFunction %void %_ptr_CrossWorkgroup_uint %_ptr_CrossWorkgroup_uint
          %6 = OpFunction %void None %5
         %in = OpFunctionParameter %_ptr_CrossWorkgroup_uint
        %out = OpFunctionParameter %_ptr_CrossWorkgroup_uint
      %entry = OpLabel
   %arrayidx = OpInBoundsPtrAccessChain %_ptr_CrossWorkgroup_uint %in %uint_0
         %12 = OpLoad %uint %arrayidx Aligned 4
  %arrayidx1 = OpInBoundsPtrAccessChain %_ptr_CrossWorkgroup_uint %out %uint_0
               OpStore %arrayidx1 %12 Aligned 4
               OpReturn
               OpFunctionEnd
```

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

SPIR-V in Compute

- Fragmentation: support for 2 compute execution models: GLCompute (compute shaders), Kernel (OpenCL)

- 2 different specifications of Extended Instruction Sets
- Different ways of dealing with memory (e.g. Storage Classes, function parameter passing)

- Both execution models require appropriate driver support

ARM

SPIR-V in compute - limitations

- No way of expressing function pointers

- Recursion representable, but may fail at runtime
- Problems with non-reducible control flow
- No native exception support

- Stack size possibly limited:
  - Problems with "deep" call-chains

ARM

Agenda

- Introduction to Khronos
- Introduction to SPIR-V
- SPIR-V Structure
- LLVM and SPIR-V differences
- SPIR-V Tooling
- SPIR-V for Compute
- Summary

ARM

Summary

- SPIR-V introduces some interesting concepts for an IR
- LLVM could be extended to support SPMD/ SIMD concept natively
- Good for LLVM to support more diverse hardware targets
- SPIR-V in LLVM tip of tree

ARM

Thanks

Neil Hickey neil.hickey@arm.com - ARM

Jakub (Kuba) Kuderski kubakuderski@gmail.com- PUT

ARM