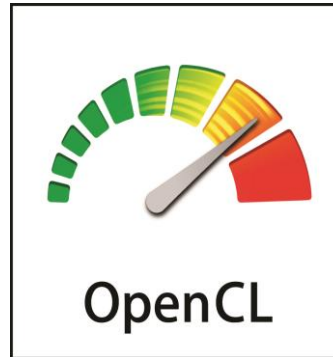




Neil Trevett
Vice President Mobile Ecosystem at NVIDIA
President of Khronos and Chair of the OpenCL Working Group
SIGGRAPH, July 2016

Khronos Connects Software to Silicon

Open Consortium creating
ROYALTY-FREE, OPEN STANDARD
APIs for hardware acceleration

Defining the roadmap for
low-level silicon interfaces
needed on every platform

Graphics, compute
and vision processing

Rigorous specifications AND
conformance tests for cross-
vendor portability

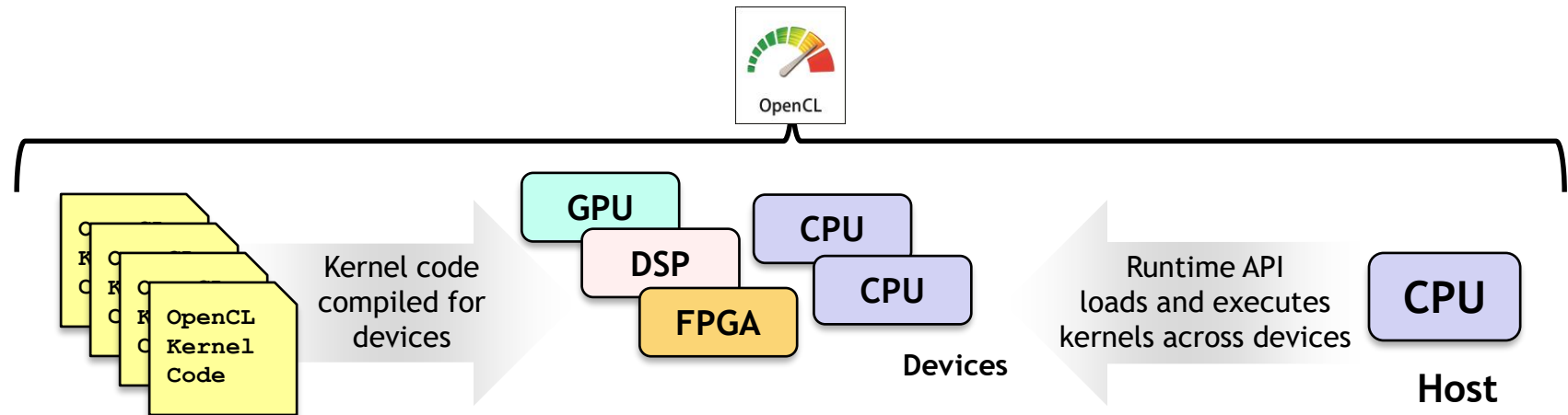
*Acceleration APIs
BY the Industry
FOR the Industry*



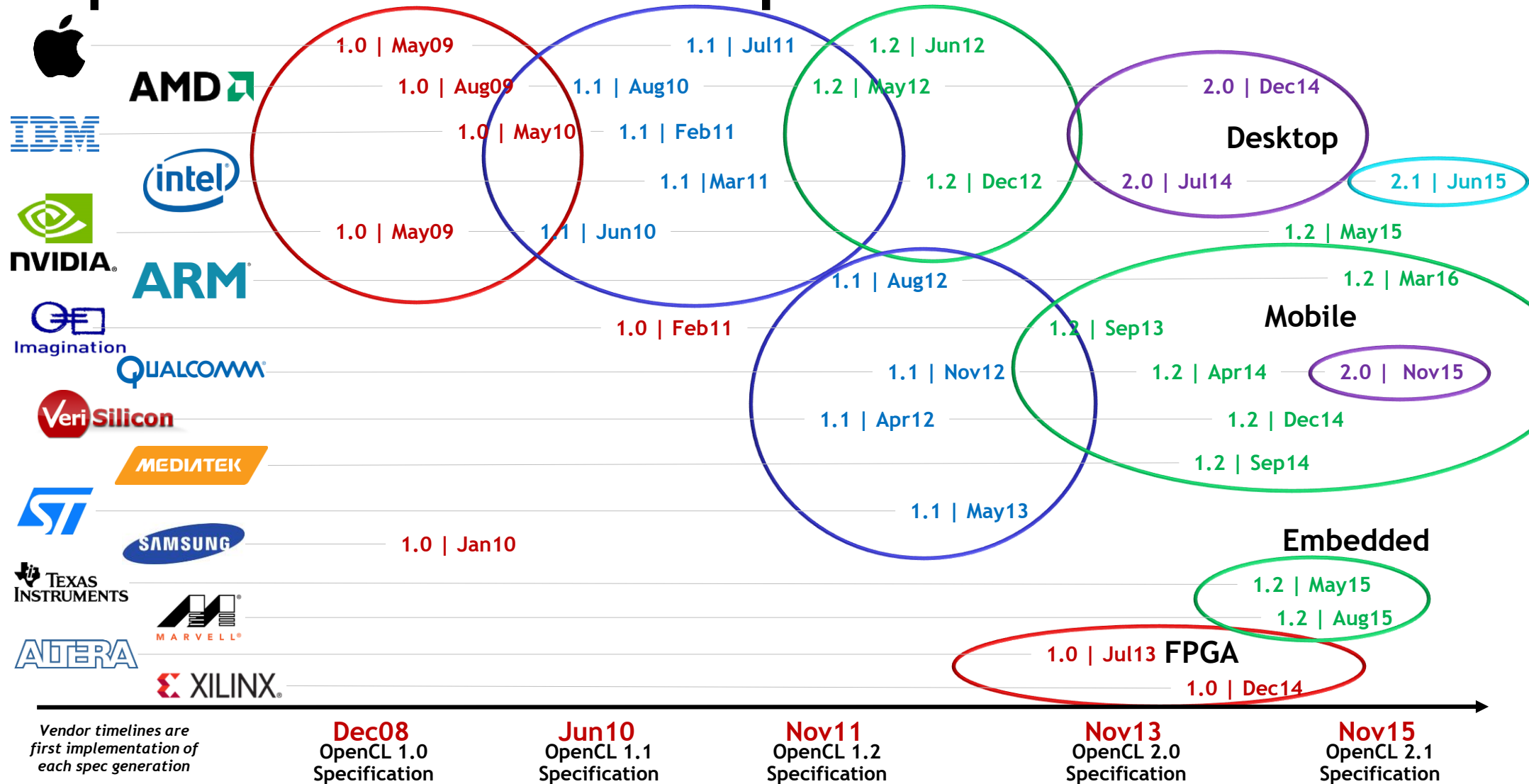
**Well over a *BILLION* people use Khronos APIs
Every Day...**

OpenCL

- **Heterogeneous parallel programming of diverse compute resources**
 - One code tree can be executed on CPUs, GPUs, DSPs and FPGA
- **OpenCL = Two APIs and Two Kernel languages**
 - C Platform Layer API to query, select and initialize compute devices
 - C Runtime API to build and execute kernels across multiple devices
 - OpenCL C and OpenCL C++ kernel languages
- **New in OpenCL 2.2 - OpenCL C++ kernel language is a static subset of C++14**
 - Adaptable and elegant sharable code - great for building libraries
 - Templates enable meta-programming for highly adaptive software
 - Lambdas used to implement nested/dynamic parallelism



OpenCL Conformant Implementations



OpenCL 2.2 - Top to Bottom C++



Single Source C++ Programming

Full support for features in C++14-based Kernel Language



API and Language Specs

Brings C++14-based Kernel Language into core specification



Portable Kernel Intermediate Language

Support for C++14-based kernel language e.g. constructors/destructors

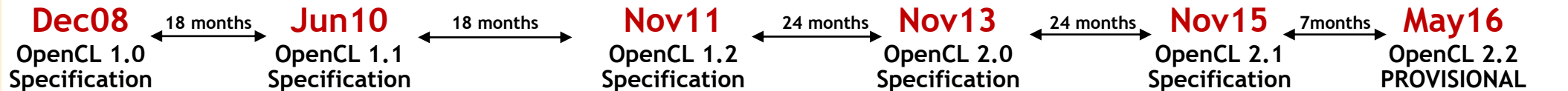
OpenCL C++ Kernel Language
SPIR-V 1.1 with C++ support
SYCL 2.2 for single source C++

3-component vectors
Additional image formats
Multiple hosts and devices
Buffer region operations
Enhanced event-driven execution
Additional OpenCL C built-ins
Improved OpenGL data/event interop

Device partitioning
Separate compilation and linking
Enhanced image support
Built-in kernels / custom devices
Enhanced DX and OpenGL Interop

Shared Virtual Memory
On-device dispatch
Generic Address Space
Enhanced Image Support
C11 Atomics
Pipes
Android ICD

SPIR-V in Core
Subgroups into core
Subgroup query operations
clCloneKernel
Low-latency device
timer queries



SPIR-V Ecosystem



Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

<https://github.com/KhronosGroup/SPIRV-Tools>

SPIR-V (Dis)Assembler

SPIR-V Validator

Other Intermediate Forms

SPIR-V

- Khronos defined and controlled cross-API intermediate language
 - Native support for graphics and parallel constructs
 - 32-bit Word Stream
 - Extensible and easily parsed
 - Retains data object and control flow information for effective code generation and translation

Third party kernel and shader Languages

HLSL

GLSL

OpenCL C

OpenCL C++

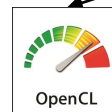
New with
OpenCL 2.2
And SPIR-V 1.1

LLVM to SPIR-V
Bi-directional
Translator

LLVM

SPIR-V Magic #: 0x07230203
SPIR-V Version 99
Builder's Magic #: 0x051a00BB
<id> bound is 50
0
OpMemoryModel
Logical
GLSL450
OpEntryPoint
Fragment shader
function <id> 4
OpTypeVoid
<id> is 2
OpTypeFunction
<id> is 3
return type <id> is 2
OpFunction
Result Type <id> is 2
Result <id> is 4
0
Function Type <id> is 3
SPIR

IHV Driver
Runtimes



SYCL for OpenCL

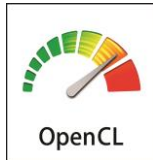
- Single-source heterogeneous programming using STANDARD C++
 - Use C++ templates and lambda functions for host & device code
- Aligns the hardware acceleration of OpenCL with direction of the C++ standard
 - C++14 with open source C++17 Parallel STL hosted by Khronos

Developer Choice

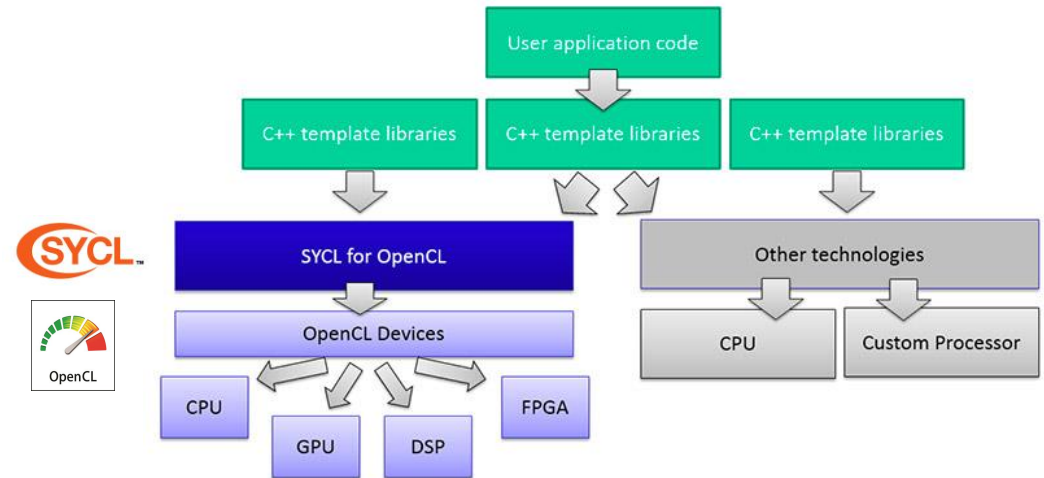
The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language

Low Level Control
'GPGPU'-style separation of
device-side kernel source
code and host code



Single-source C++
Programmer Familiarity
Approach also taken by
C++ AMP and OpenMP



OpenCL Roadmap Discussions...

Desktop

Use cases: Video and Image Processing, Gaming Compute

Roadmap: Vulkan interop, arbitrary precision for increased performance, pre-emption, collective programming and improved execution model



Vulkan Compute can leverage OpenCL?

Gaming Compute, Pixel Processing, Inference

Fine grain graphics and compute (no interop needed)

SPIR-V for shading language flexibility - C/C++

Low-latency, fine grain run-time

Google Android adoption

Competes well with Metal (=C++/OpenCL 1.2)

Roadmap: types, precision and accuracy

Pointers and address spaces, execution model



Mobile

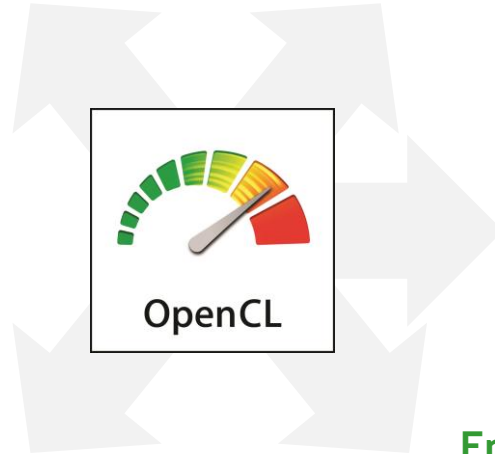
Use case: Photo and Vision Processing

Roadmap: arbitrary precision for inference engine and pixel processing efficiency, pre-emption and QoS scheduling for power efficiency

HPC, SciViz, Datacenter

Use case: Numerical Simulation, Virtualization

Roadmap: enhanced streaming processing, enhanced library support



FPGAs

Use cases: Network and Stream Processing

Roadmap: enhanced execution model, self-synchronized and self-scheduled graphs, fine-grained synchronization between kernels, DSL in C++

Embedded

Use cases: Signal and Pixel Processing

Roadmap: arbitrary precision for power efficiency, hard real-time scheduling, asynch DMA

Possible learnings from Vulkan Philosophy

1. Explicit - provide direct access to hardware capabilities with thin driver
2. Feature Sets - enable diverse architectures to ship just relevant features
3. Open source conformance tests for deep community engagement

Speakers

Neil Trevett	NVIDIA	OpenCL 2.2 and Ecosystem Update
Ralph Potter	Codeplay	SPIR-V 1.1
Ben Ashbaugh	Intel	OpenCL C++
Luke Iwanski	Codeplay	SYCL 2.2
Matias Koskela	Tampere University of Technology	POCL - Open Source OpenCL
Alex Bourd Geoffrey Wenger	Qualcomm	SceneDetect on Snapdragon 820
Ben Ashbaugh	Intel	Intel Code Builder



SPIR-V: A shader IR for OpenCL, OpenGL and Vulkan

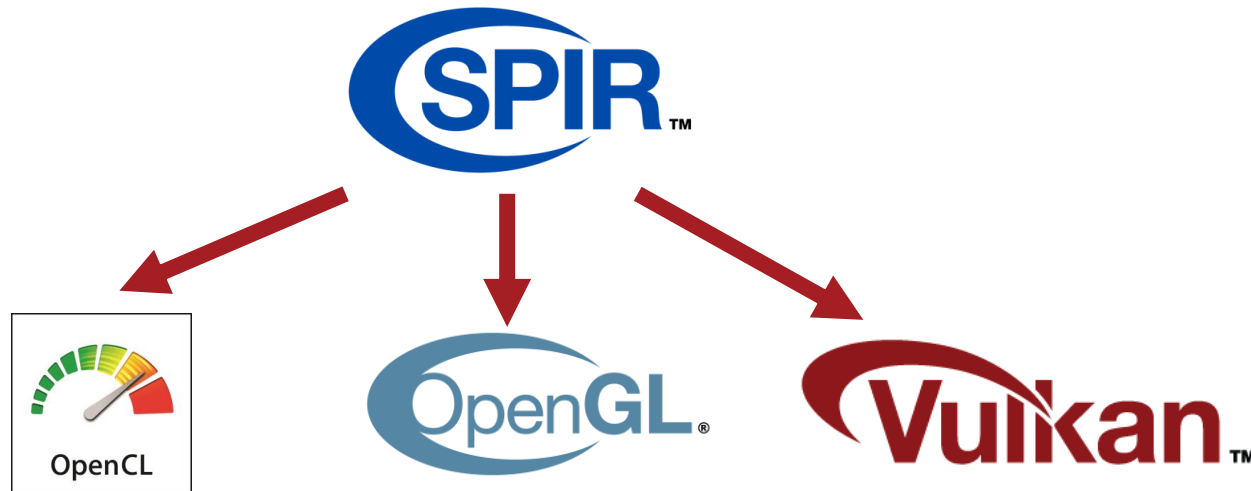
Ralph Potter
Staff Software Engineer - Research, Codeplay

Overview

- What is SPIR-V?
- SPIR vs SPIR-V
- Supporting multiple APIs with a single IR
- Example
- Generating, manipulating and consuming SPIR-V
- What's new in SPIR-V 1.1?

What is SPIR-V?

- Compiler IR for OpenCL and Vulkan as core and OpenGL via ARB_gl_spirv
- Easy to parse. Just a stream of words
- Easy to transform. Designed to be easy to convert to and from LLVM IR
- Easy to manipulate and optimize. Single-Static Assignment form
- 1.0 announced at SC15 (Nov 15), 1.1 at IWOCL (April 16)



SPIR vs SPIR-V

- **SPIR**

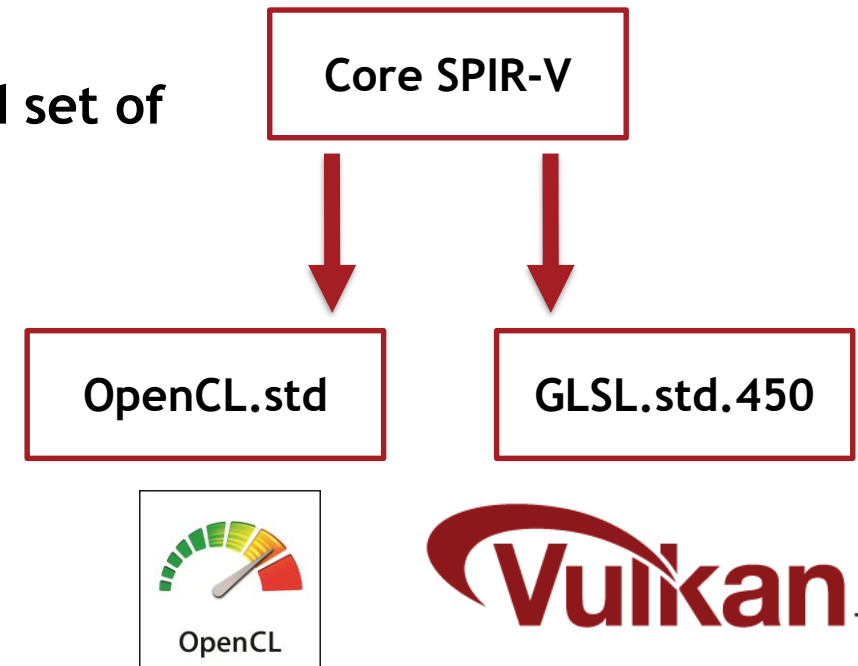
- OpenCL only
- Derived from LLVM 3.2 IR, with restrictions and extensions
- Requires the optional `cl_khr_spir` extension

- **SPIR-V**

- New IR completely defined by Khronos
- Shared core IR, with extended instructions for OpenCL and Vulkan
- SPIR-V consumption is core and non-optional for OpenCL and Vulkan
- Available via the `ARB_gl_spirv` extension for OpenGL

Supporting OpenCL and Vulkan

- Each runtime API defines an Environment
 - Sets of Capability flags supported by each runtime
- Every instruction defines a set of required Capabilities
 - Your module must not use instructions
- Your module must match the memory model of the runtime, enabled by OpMemoryModel
- Additionally, runtime APIs define an extended set of instructions, which may be imported using OpExtInstImport

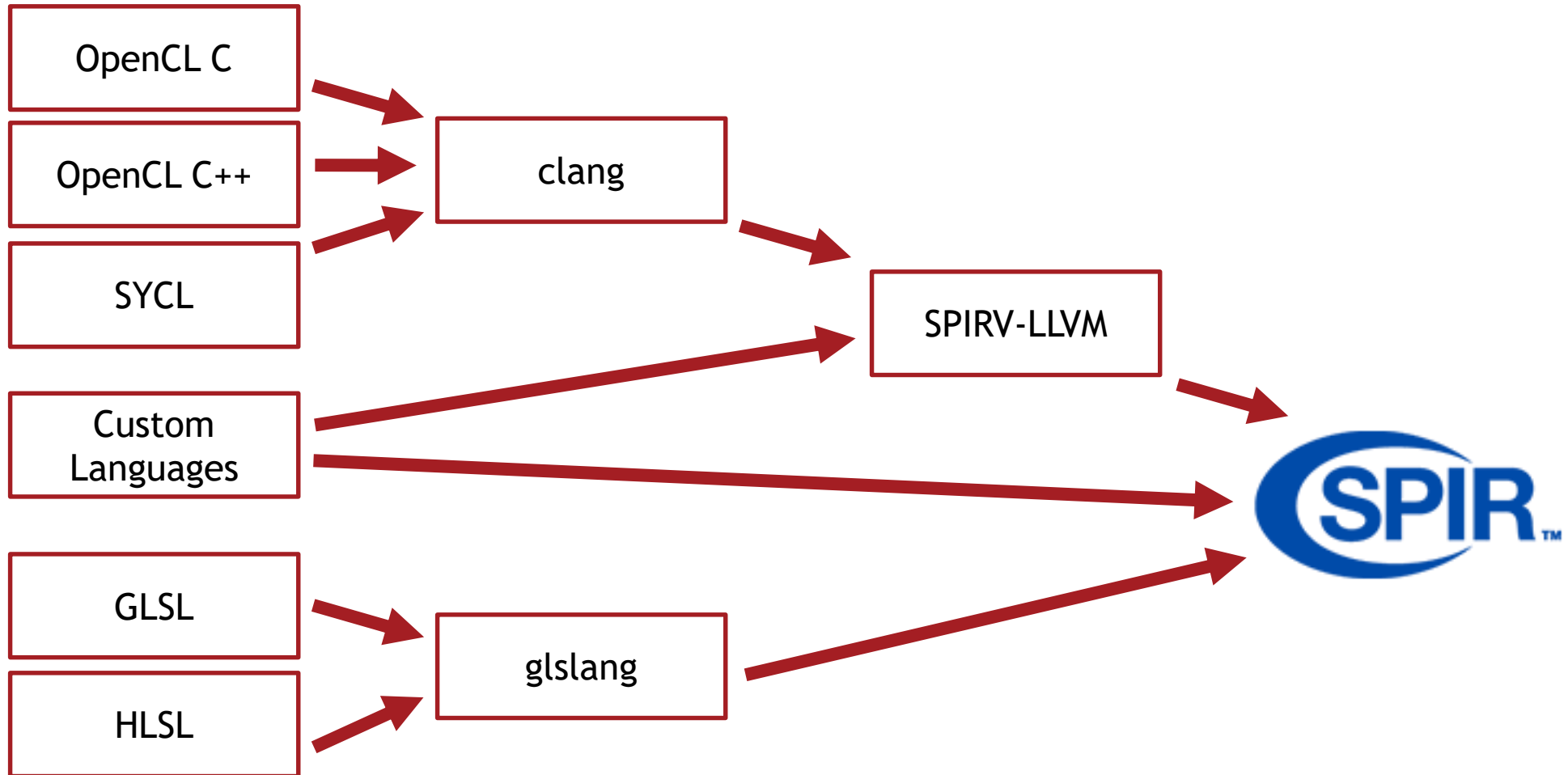


SPIR-V Example

```
kernel void f(global float *a,
              global float *b,
              global float *c) {
    int i = get_global_id(0);
    a[i] = b[i] + c[i];
}
```

```
OpCapability Addresses
OpCapability Linkage
OpCapability Kernel
%1 = OpExtInstImport "OpenCL.std"
OpMemoryModel Physical32 OpenCL
OpDecorate %16 GlobalInvocationId
OpDecorate %23 FuncParamAttr NoCapture
%23 = OpDecorationGroup
OpDecorate %10 LinkageAttributes "f" Export
OpGroupDecorate %23 %11 %12 %13
%2 = OpTypeInt 32 0
%3 = OpTypeFunction %2 %2
%6 = OpTypeVoid
%7 = OpTypeFloat 32
%8 = OpTypePointer CrossWorkgroup %7
%9 = OpTypeFunction %6 %8 %8 %8
%16 = OpVariable %2 Builtin
%10 = OpFunction %6 None %9
%11 = OpFunctionParameter %8
%12 = OpFunctionParameter %8
%13 = OpFunctionParameter %8
%14 = OpLabel
%17 = OpInBoundsPtrAccessChain %8 %12 %16
%18 = OpLoad %7 %17 Aligned 4
%19 = OpInBoundsPtrAccessChain %8 %13 %16
%20 = OpLoad %7 %19 Aligned 4
%21 = OpFAdd %7 %18 %20
%22 = OpInBoundsPtrAccessChain %8 %11 %16
OpStore %22 %21 Aligned 4
OpReturn
OpFunctionEnd
```

Generating SPIR-V

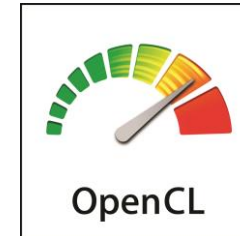


Parsing and Manipulating SPIR-V

- **Language Bindings for SPIR-V**
 - C, C++, Lua, Python, JSON
 - <https://github.com/KhronosGroup/SPIRV-Headers>
- **SPIR-V Tools**
 - Assembler, Disassembler, Validator, Binary SPIR-V Parser
 - Both library and command-line tools
 - <https://github.com/KhronosGroup/SPIRV-Tools>
- **SPIR-V LLVM Bi-directional Translator**
 - Transform to and from LLVM IR
 - <https://github.com/KhronosGroup/SPIRV-LLVM>

Consuming SPIR-V

- OpenCL 2.1 added `clCreateProgramWithIL`
- Vulkan uses `vkCreateShaderModule`.
 - SPIR-V passed as a member of `VkShaderModuleCreateInfo`
- ARB_gl_spirv extends `glShaderBinary` with binary format = `GL_SHADER_BINARY_FORMAT_SPIR_V_ARB`



What's New in SPIR-V 1.1?

- **Primarily OpenCL 2.2 features**
- **Named barriers**
 - More flexible barriers, allowing waits at subgroup granularity
- **Subgroup Dispatch**
 - Instructions for querying and constraining workgroup size based on number of subgroups
- **Pipe storage**
 - Instructions for program scope pipes
- **Initialiser and Finaliser module entry points for global C++ constructors**
- **Support OpenCL-C++ ivdep to aid loop optimisation**

Summary

- What is SPIR-V?
- SPIR vs SPIR-V
- Supporting multiple APIs with a single IR
- Example
- Generating, manipulating and consuming SPIR-V
- What's new in SPIR-V 1.1?



OpenCL C++ Kernel Language

Ben Ashbaugh
Intel

SIGGRAPH, August 2016

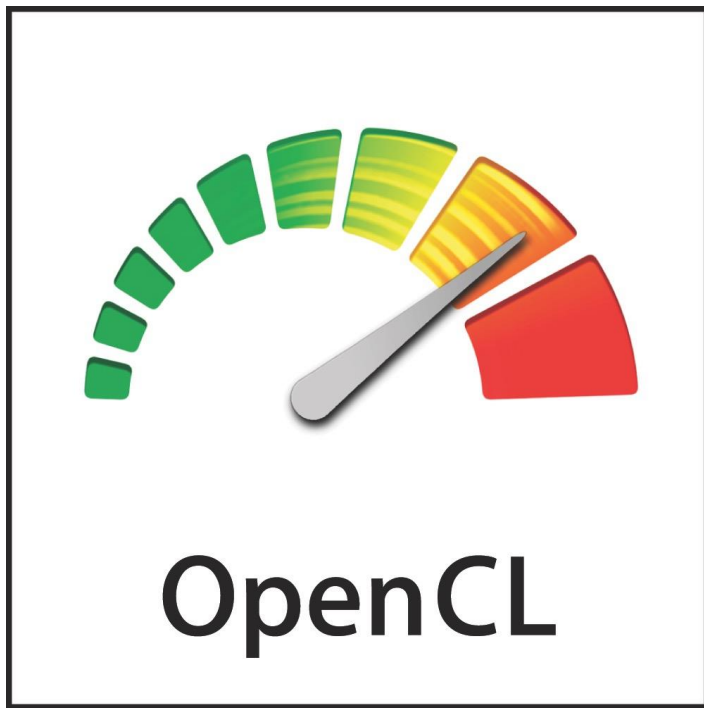
OpenCL C++ Kernel Language Overview

- **A “Static Subset” of C++14**
 - Frees developers from low-level coding details without sacrificing performance
- **In: Classes, templates, lambdas, function and operator overloading, more...**
 - Reusable device libraries and containers - fast and elegant sharable code
 - Templates enables meta-programming for highly adaptive software
- **In: Upgraded OpenCL Standard Library**
 - Leverages C++ standard library features to improve type safety, simplify syntax
 - Examples: atomics, images, device queues, math functions
- **Out: Virtual Functions, Exceptions, Type Identification, C++ Standard Library...**

Q: I know OpenCL C and C++, how much do I have to learn?

A: If you know both languages, OpenCL C++ will be familiar, intuitive, and easy to learn!

A: If you know either OpenCL C or C++, OpenCL C++ is a natural extension!



K H R O N O S[™]
G R O U P

C++ Example #1: Hello, World! (Array Copy)

C++: Example 1: Array Copy

OpenCL C

```
kernel void myKernel(global int *in,
                     global int *out)
{
    size_t tid = get_global_id(0);
    out[tid] = in[tid];
}
```

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
```

```
kernel void myKernel(cl::global_ptr<int[]> in,
                     cl::global_ptr<int[]> out)
{
    size_t tid = cl::get_global_id(0);
    out[tid] = in[tid];
}
```

C++: Example 1: Array Copy

OpenCL C

```
kernel void myKernel(global int *in,  
                      global int *out)  
{  
    size_t tid = get_global_id(0);  
    out[tid] = in[tid];  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_memory>
```

```
kernel void myKernel(cl::global_ptr<int[]> in,  
                     cl::global_ptr<int[]> out)  
{  
    size_t tid = cl::get_global_id(0);  
    out[tid] = in[tid];  
}
```

C++: Example 1: Array Copy

OpenCL C

```
kernel void myKernel(global int *in,  
                      global int *out)  
{  
    size_t tid = get_global_id(0);  
    out[tid] = in[tid];  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_memory>
```

```
kernel void myKernel(cl::global_ptr<int[]> in,  
                      cl::global_ptr<int[]> out)  
{  
    size_t tid = cl::get_global_id(0);  
    out[tid] = in[tid];  
}
```

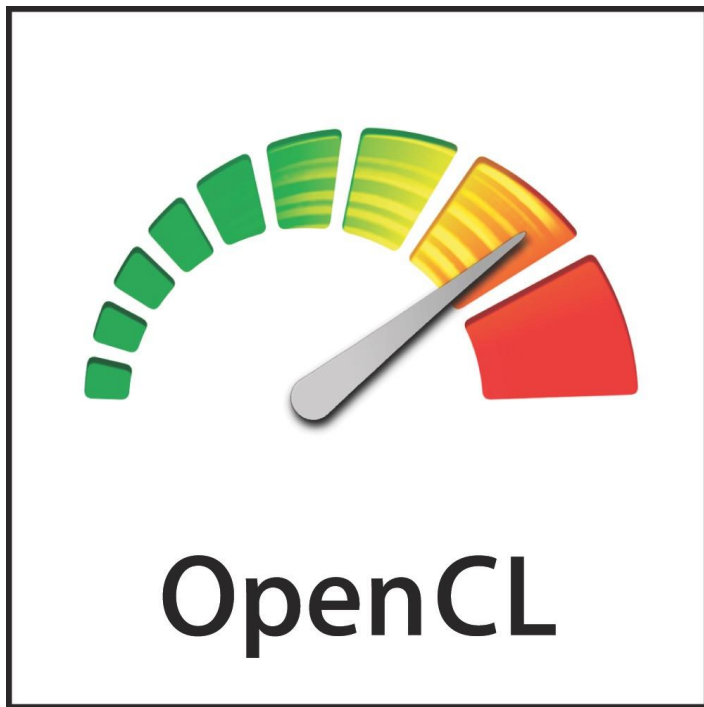
C++: Example 1: Array Copy

OpenCL C

```
kernel void myKernel(global int *in,  
                     global int *out)  
{  
    size_t tid = get_global_id(0);  
    out[tid] = in[tid];  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_memory>  
  
kernel void myKernel(cl::global_ptr<int[]> in,  
                    cl::global_ptr<int[]> out)  
{  
    size_t tid = cl::get_global_id(0);  
    out[tid] = in[tid];  
}
```



K H R O N O S[™]
G R O U P

C++ Example #2: Images

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;
```

```
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;
```

```
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```

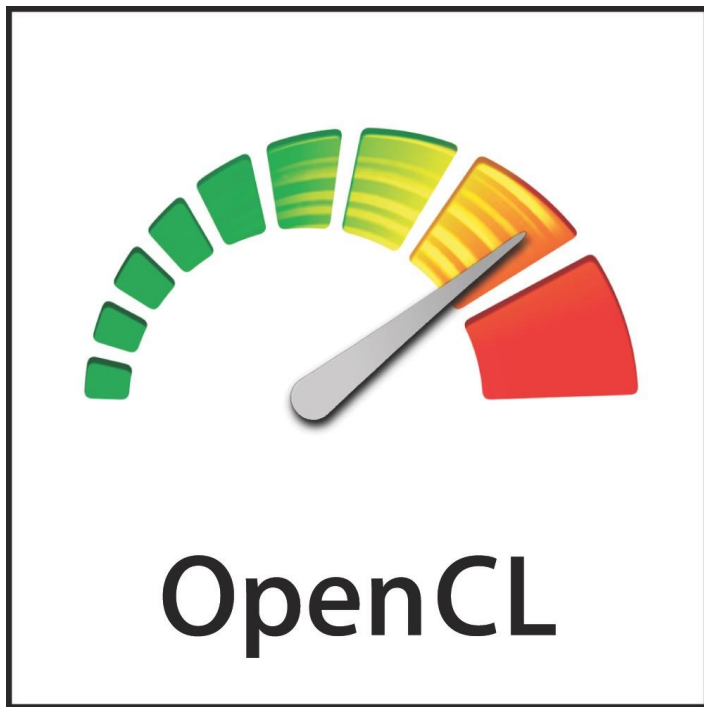
C++: Example 2: Images

OpenCL C

```
kernel void myKernel(  
    read_only image2d_t img,  
    sampler_t s,  
    global float4 *out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords = (int2)(tidX, tidY);  
    int w = get_image_width(img);  
    out[w * tidY + tidX] = read_imagef(img, s, coords);  
}
```

OpenCL C++

```
#include <opencl_work_item>  
#include <opencl_image>  
#include <opencl_memory>  
using namespace cl;  
  
kernel void myKernel(  
    image2d<float4, image_access::sample> img,  
    sampler s,  
    global_ptr<float4[ ]> out)  
{  
    size_t tidX = get_global_id(0), tidY = get_global_id(1);  
    int2 coords(tidX, tidY);  
    int w = img.width();  
    out[w * tidY + tidX] = img.sample(s, coords);  
}
```



K H R O N O S[™]
G R O U P

C++ Example #3: Templates

C++: Example 3: Templates

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
using namespace cl;

template<typename T>
void add_vectors(const T& srcA,
                const T & srcB,
                T& dst)
{
    size_t id = get_global_id(0);
    dst[id] = srcA[id] + srcB[id];
}
```

```
kernel void add_vectors_float(
    const global_ptr<float[]> srcA,
    const global_ptr<float[]> srcB,
    global_ptr<float[]> dst)
{
    add_vectors(srcA, srcB, dst);
}

kernel void add_vectors_int(
    const global_ptr<int[]> srcA,
    const global_ptr<int[]> srcB,
    global_ptr<int[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

C++: Example 3: Templates

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
using namespace cl;
```

```
template<typename T>
```

```
void add_vectors(const T& srcA,
                const T & srcB,
                T& dst)
```

```
{
    size_t id = get_global_id(0);
    dst[id] = srcA[id] + srcB[id];
}
```

```
kernel void add_vectors_float(
    const global_ptr<float[]> srcA,
    const global_ptr<float[]> srcB,
    global_ptr<float[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

```
kernel void add_vectors_int(
    const global_ptr<int[]> srcA,
    const global_ptr<int[]> srcB,
    global_ptr<int[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

C++: Example 3: Templates

OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
using namespace cl;

template<typename T>
void add_vectors(const T& srcA,
                const T & srcB,
                T& dst)
{
    size_t id = get_global_id(0);
    dst[id] = srcA[id] + srcB[id];
}
```

```
kernel void add_vectors_float(
    const global_ptr<float[]> srcA,
    const global_ptr<float[]> srcB,
    global_ptr<float[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

```
kernel void add_vectors_int(
    const global_ptr<int[]> srcA,
    const global_ptr<int[]> srcB,
    global_ptr<int[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

C++: Example 3: Templates

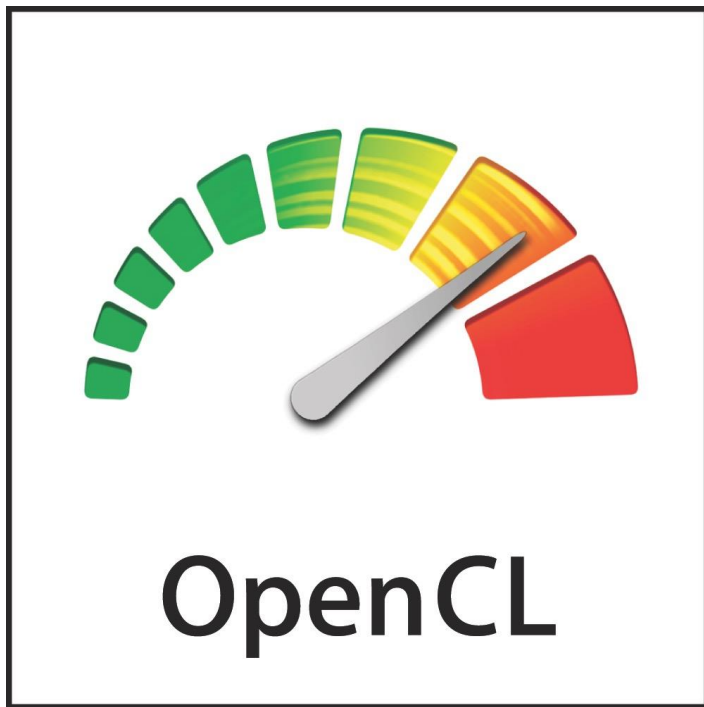
OpenCL C++

```
#include <opencl_work_item>
#include <opencl_memory>
using namespace cl;

template<typename T>
void add_vectors(const T& srcA,
                const T & srcB,
                T& dst)
{
    size_t id = get_global_id(0);
    dst[id] = srcA[id] + srcB[id];
}
```

```
kernel void add_vectors_float(
    const global_ptr<float[]> srcA,
    const global_ptr<float[]> srcB,
    global_ptr<float[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

```
kernel void add_vectors_int(
    const global_ptr<int[]> srcA,
    const global_ptr<int[]> srcB,
    global_ptr<int[]> dst)
{
    add_vectors(srcA, srcB, dst);
}
```

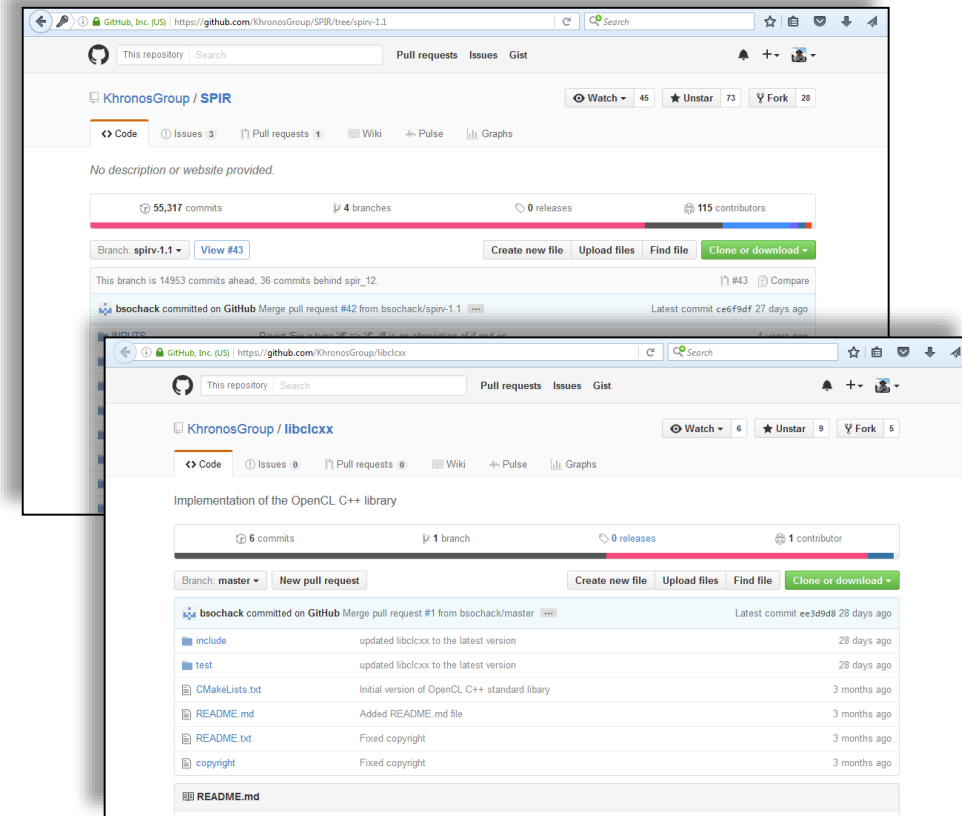


K H R O N O S[™]
G R O U P

Summary

Current Status

- OpenCL C++ Kernel Language Spec is Provisional
 - <https://www.khronos.org/registry/cl/specs/opencl-2.2-cplusplus.pdf>
- Try it out with the Open Source, Free OpenCL C++ Kernel Language Compiler!
 - <https://github.com/KhronosGroup/SPIR/tree/spirv-1.1>
 - <https://github.com/KhronosGroup/libclcxx>
- Can emit SPIR-V 1.0
 - For OpenCL 2.1 compatibility
- Or, can emit SPIR-V 1.1
 - Enables all OpenCL 2.2 functionality
 - Requires an OpenCL 2.2 driver/runtime





SYCL™ 2.2 For Software Devs

**Luke Iwanski, Staff Software Engineer,
Compute, Graphics & Vision @ Codeplay
SIGGRAPH Anaheim 2016**

SYCL 2.2

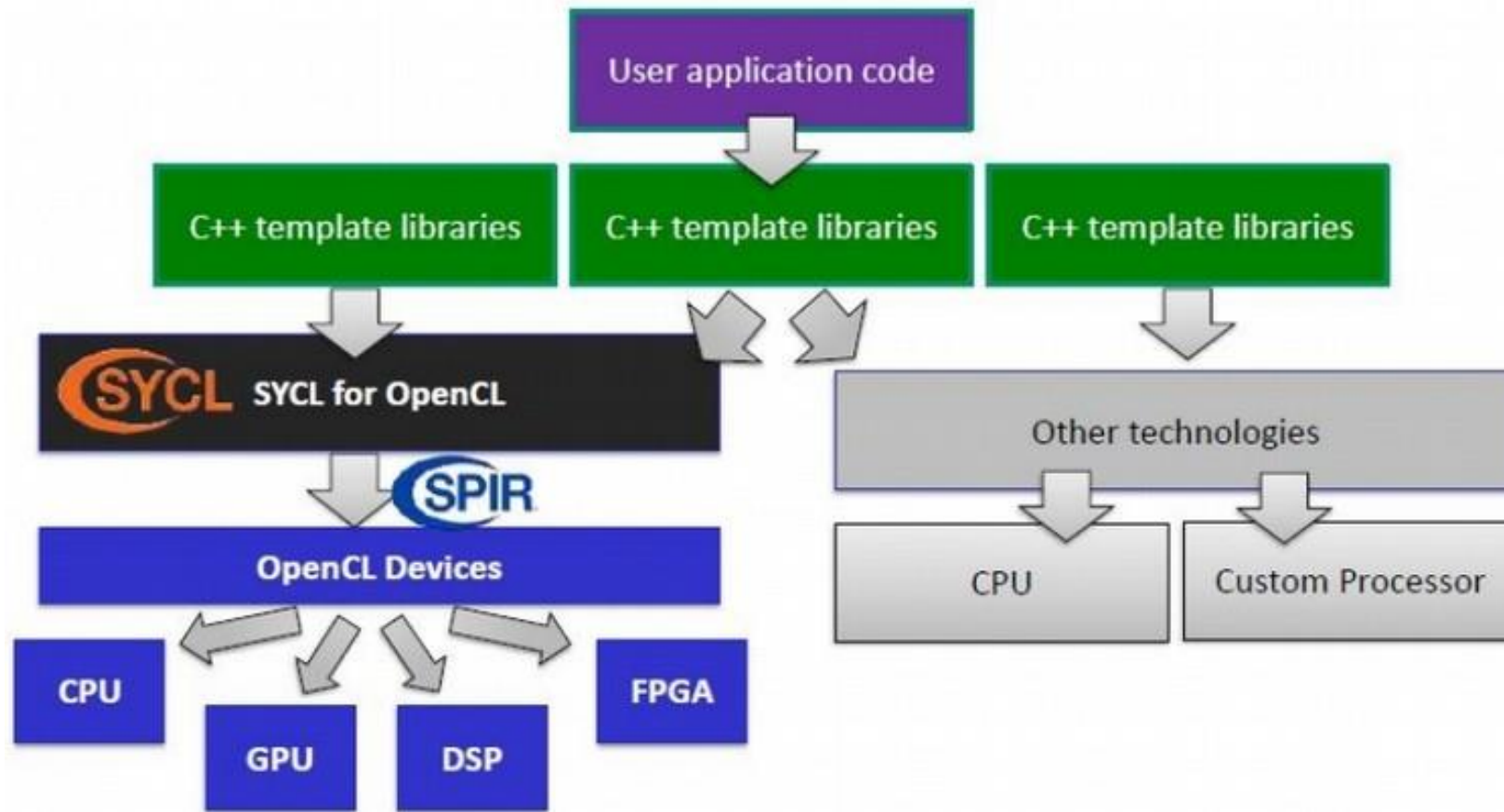
- Our Aim
- Ecosystem Overview
- New Features
- Use Cases



Our Aim

- SYCL single source, modern C++ programming model exposing OpenCL 2.2 feature-set
- Follow current C++ standard developments and enhance integration of OpenCL with developments in the C++ standard
- Enhance the Khronos ecosystem and target SPIR-V 1.1 for compute
- Full and seamless backwards compatibility with SYCL 1.2
- As usual feedback is welcome!

Ecosystem Overview



More Details: SYCL™ for OpenCL™ in a Nutshell (<https://goo.gl/fllrB2>)

SYCL 2.2: New Features

SYCL Command Groups

- All of the OpenCL commands for memory object creation, copying, mapping and synchronization operations to correctly execute a kernel on a device are defined in a functor and called from a command group
- Since SYCL 2.2 `execution_handler` takes parameter which defines OpenCL capabilities used under the hood

```
1 // wrap our data variable in a buffer
2 buffer<int, 1> resultBuf(data, range<1>(1024));
3 // create a command_group to issue commands to the queue
4 myQueue.submit([&](execution_handle<opencl22>& cgh) {
5     // request access to the buffer
6     auto writeResult = resultBuf.get_access<access::mode::write>(cgh);
7     // enqueue a parallel_for task
8     cgh.parallel_for<class simple_test>(range<1>(1024), [=](id<1> idx) {
9         writeResult[idx[0]] = idx[0];
10    }); // end of the kernel function
11 });
12 // end of our commands for this queue
```

SYCL Nested Parallelism

- Nested parallelism is a form of parallelism where more work can be submitted to the device, with the scope of the submission to be the SYCL host command group
- All of the subsequent device command groups that are submitted from the parent command group kernel will be enqueued on a device_queue and executed asynchronously in relation to the parent kernel

```
1  q.submit([&](cl::sycl::execution_handle<opencl22>& cgh) {
2      auto anAcc = aBuffer.get_access<cl::sycl::access::mode::read_write>(cgh);
3      auto anotherAcc = anotherBuffer.get_access<cl::sycl::access::mode::read_write>(cgh);
4
5      cgh.single_task<class device_side_enqueue>(cl::sycl::range<1>(1), [=](cl::sycl::id<1> idx) {
6          anAcc[idx]++;
7          // set up for device queue
8          device_queue dq = q.get_default_queue();
9          auto event = dq.submit(enqueue_policy::wait_kernel, [&](cl::sycl::device_handle& dh) {
10              int error = dh.parallel_for(range<1>(numElements2), [=](cl::sycl::id<1> idx2) {
11                  anotherAcc[idx2]--;
12              });
13          });
14      });
15  });
```

SYCL Hierarchical Parallelism

- Approach models the three levels of parallelism which are available in the OpenCL execution model through three nested parallel for function scopes
 - ✦ `parallel_for_work_group` : once per workgroup
 - ✦ `parallel_for_sub_group` : once per vector of work-items
 - ✦ `parallel_for_work_item` : once per work-item

```
1 auto flexible_command_group = [&](execution_handle& cgh) {  
2     cgh.parallel_for_work_group<class example_kernel>(range<3>(2, 2, 2), [=](group<3> my_group) {  
3         parallel_for_sub_group(myGroup, [=](sub_group<3> my_sub_group){  
4             //[sub-group code]  
5             parallel_for_work_item(my_sub_group, [=](item<3> my_item) {  
6                 //[work-item code]  
7             });  
8         });  
9         //[workgroup code]  
10    });  
11 }
```

SYCL Collective Operations

- The group and sub_group classes encapsulate the functionality within a work-group or a sub-group level of parallel execution. Since OpenCL 2.0 collective functions are available {all, any, broadcast, reduce, scan}

```
1 q.submit([&](cl::sycl::execution_handle& cgh) {
2     auto acc = buffer.get_access<cl::sycl::access::mode::read_write>(cgh);
3     cgh.single_task<class svm_sample1>(cl::sycl::nd_range<2>(range<2>(16, 16), range<2>(4, 4)),
4     [=](cl::sycl::nd_item<1> idx) {
5         group my_group = idx.get_group();
6         int x = a_function();
7         int sum = my_group<work_group_op::add> reduce(x);
8         //...
9     });
10 });
```

SYCL Shared Virtual Memory (SVM)

- SVM enables using complex data classes and pointers between host and devices, using atomics and in some cases the lifetime scope of the buffer and accessor classes as synchronization points
- There are different flavors of shared virtual memory, depending on the capabilities of the OpenCL system to share device pointers with the host

SYCL Shared Virtual Memory (SVM)

Coarse Grained SVM

A buffer in a context can be allocated as coarse-grained buffer on a device and have the same underlying raw pointer on both host and device. Accessors are needed in this case to ensure data consistency

Fine Grained Buffer Sharing SVM

Shared virtual memory with buffer sharing support denotes the capability to share memory allocations between host and device, at the granularity of each allocation. Depending on the availability of atomics; loads and stores can be atomic

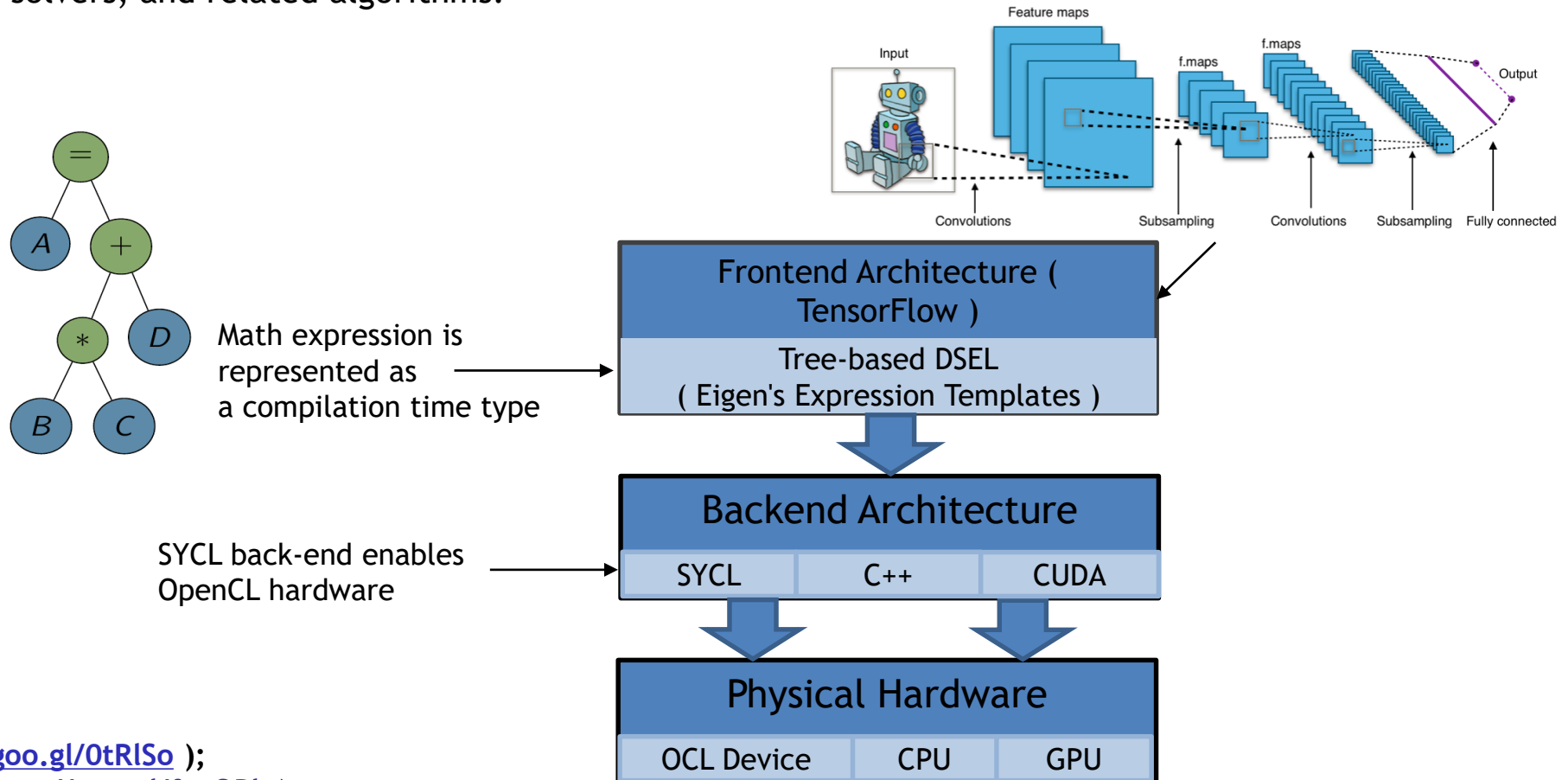
```
{
    /* Access the SVM buffer on host */
    auto hostSvmCoarsePointer =
    svmBuffer.get_access<cl::sycl::access::mode::write,
    cl::sycl::access::target::host_buffer>();
    /* Within this block it is safe to use the raw SVM pointer. */
    auto rawSvmCoarsePointer = hostSvmCoarsePointer.get();
    *rawSvmCoarsePointer = 100;
} // The underlying SVM pointer gets updated and no access on host is
// possible when the host accessor goes out of scope

q.submit([&](cl::sycl::execution_handle<svm_coarse_grain>& cgh) {
    /* Make the SVM pointer allocation available for updating on the device */
    auto deviceCoarsePointer =
    svmBuffer.get_access<cl::sycl::access::mode::read_write>(cgh);
    cgh.single_task<class svm_sample1>(<
    cl::sycl::range<1>(1), [=](cl::sycl::id<1> index) {
        int* rawSvmCoarsePointer = deviceCoarsePointer.get();
        rawSvmCoarsePointer[index] = a + 1;
    });
});
});
```

```
cl::sycl::svm_allocator<int, cl::sycl::svm_fine_grain>
fineGrainAllocator( fineGrainedContext);
/* Use the instance of the svm allocator with any custom
 * container, for example, with std::shared_ptr.
 */
std::shared_ptr<int> svmFineBuffer =
    std::allocate_shared<int>(fineGrainAllocator, numElements);
/* Initialize the pointer on the host */
*svmFineBuffer = 66;
q.submit([&](cl::sycl::execution_handler<svm_fine_grain<
cl::sycl::svm_sharing::buffer, cl::sycl::svm_atomics::none>>& cgh) {
    auto rawSvmFinePointer = svmFineBuffer.get();
    /* Register access so that dependency tracking can be possible and the
     * kernel can use the raw pointer
     */
    cgh.register_access<cl::sycl::access::mode::read_write>(rawSvmFinePointer);
    cgh.single_task<class svm_fine_grained_kernel>(range<1>(1), [=](id<1> index) {
        rawSvmFinePointer[index]++;
    });
});
```

Use Cases: TensorFlow

“TensorFlow™ is an open source software library for numerical computation using data flow graphs.”
For heavy math computation **Eigen** is used - “a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.”



WIP links:

Eigen (<https://goo.gl/0tRISo>);

TensorFlow (<https://goo.gl/fngQPb>);

Use Cases: SYCL Parallel STL

Table 1 — Table of parallel algorithms

adjacent_difference	adjacent_find	all_of	any_of
copy	copy_if	copy_n	count
count_if	equal	exclusive_scan	fill
fill_n	find	find_end	find_first_of
find_if	find_if_not	for_each	for_each_n
generate	generate_n	includes	inclusive_scan
inner_product	inplace_merge	is_heap	is_heap_until
is_partitioned	is_sorted	is_sorted_until	lexicographical_compare
max_element	merge	min_element	minmax_element
mismatch	move	none_of	nth_element
partial_sort	partial_sort_copy	partition	partition_copy
reduce	remove	remove_copy	remove_copy_if
remove_if	replace	replace_copy	replace_copy_if
replace_if	reverse	reverse_copy	rotate
rotate_copy	search	search_n	set_difference
set_intersection	set_symmetric_difference	set_union	sort
stable_partition	stable_sort	swap_ranges	transform
transform_exclusive_scan	transform_inclusive_scan	transform_reduce	uninitialized_copy
uninitialized_copy_n	uninitialized_fill	uninitialized_fill_n	unique
unique_copy			

[Note: Not all algorithms in the Standard Library have counterparts in Table 1. — end note]

Parallel STL is an implementation of the Technical Specification for C++ Extensions for Parallelism, current document number N4507

<https://github.com/KhronosGroup/SyclParallelSTL>

```
using namespace cl::sycl;
using namespace experimental::parallel::sycl;
std::vector<int> v = ...;

default_selector ds ;
queue q(ds);
sort (sycl_execution_policy(q), std::begin(v), std::end(v));
```

Thanks & References

- <https://goo.gl/fllrB2> - SYCL™ for OpenCL™ in a Nutshell
- <https://goo.gl/0tRlSo> - [WIP] Eigen bitbucket repository
- <https://goo.gl/fngQPb> - [WIP] TensorFlow github repository
- <https://github.com/KhronosGroup/SyclParallelSTL> - ParallelST repository
- <http://www.khronos.org/opencl/sycl> - SYCL specs and forums
- <http://www.codeplay.com/portal/> - Codeplay's blogs

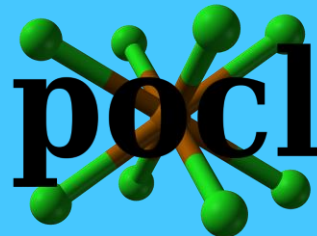


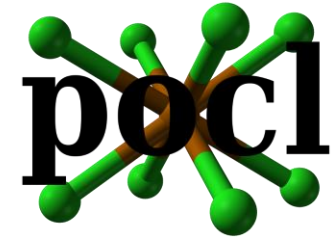
Portable Computing Language (pocl)

Matias Koskela

Customized Parallel Computing (CPC) group
Tampere University of Technology, Finland

portablecl.org





What is pocl?

- MIT-licensed open source implementation of the OpenCL standard
- Portable code base and kernel compiler
 - Homogeneous CPU and heterogeneous GPUs/DSPs/accelerators
- Upstream **Clang** as an OpenCL C frontend
- Uses **LLVM** for the kernel compiler implementation, and as a portability layer.
 - Custom LLVM passes for “SPMD to MIMD”



Uses of pocl

- Getting *OpenCL support on new devices*, which have LLVM backend. For example:
 - **Kalray** utilizes pocl for their manycores
 - **Texas Instruments'** DSPs use pocl's kernel compilation passes
 - **MIPS** CPUs received OpenCL support via pocl
- *Research platform*
 - OpenCL on customized processors (tce.cs.tut.fi)
 - High level synthesis on FPGAs
 - Performance portability of GPU kernels to CPU/DSPs
- *Open source alternative to vendor SDKs*
 - OpenCL support on “old devices” such as Intel/AMD CPUs
 - Open source means transparency which can help in optimization



Recent Work

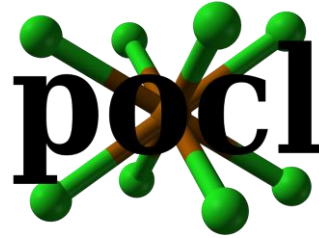
- pocl-hsa: **HSA** driver for pocl
 - Open OpenCL support for HSA-based devices
 - Funded by HSAFoundation (thanks!)
- pocl-cuda: **CUDA** driver for pocl
 - End goal is an efficient and transparent OpenCL for NVIDIA GPUs
 - Work led by James Price / University of Bristol
- Improved **offline compilation** support
 - Program binary format that removes the need for having a compiler in the host



Future Work

- OpenCL standard compliance
 - Looking for helping hands & resources
 - 1.2 very close, 2.x a bit farther
- Improve the kernel compiler's WG autovectorization
- Standardized FPGA SoC support
- Efficient execution of heterogeneous command queues





= MIT-licensed OpenCL implementation

“pocl: A Performance-Portable OpenCL Implementation”, Jääskeläinen et al.,
International Journal of Parallel Programming, Springer, October 2015, Vol 43,
Issue 5

portablecl.org
#pocl @ irc.oftc.net
Thank you!

Snapdragon Neural Processing Engine (SNPE)

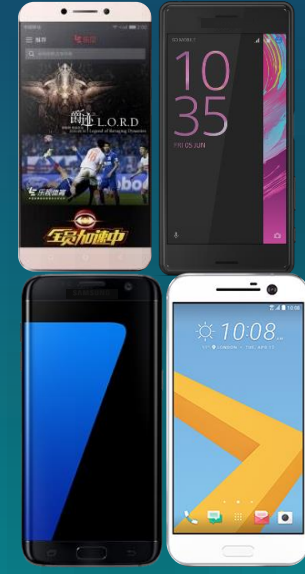


Commercial Products with OpenCL Support

2013

2014 - 2015

2016



- Snapdragon 600
- Snapdragon 400
- Snapdragon 410
- Snapdragon 800
- Snapdragon 801
- Snapdragon 805
- Snapdragon 810
- Snapdragon 615
- Snapdragon 820
- Snapdragon 652

OpenCL 1.1 Embedded

OpenCL 1.2 Full

OpenCL 2.0 Full

SNPE Overview

- **Part of Qualcomm® Zeroth™**
 - Fully on-device approaches for on-device machine learning classifiers to complement cloud solutions
 - Applications to visual understanding, image classification, video analysis, audio, and biometrics
 - **Key focus is to enable fast runtime execution of CNNs and RNNs**
- **SNPE provides an optimized runtime for the execution of arbitrary deep neural or recurrent networks**
- **SNPE accepts neural networks in a Zeroth specific format**
- **Utilizes Qualcomm Technologies' Symphony heterogeneous compute runtime, deploys the neural network onto Snapdragon HW (CPU, DSP, GPU) in order to provide efficient, low power execution of the neural network**

SNPE Performance

Run on Snapdragon 820, Android 23

	GPU Avg Latency (ms)
AlexNet	33.4
GoogLeNet	51.6
QC Scenes	16.1

SNPE Demo Overview

- “Car detector” model running on SNPE inside an Android app
- Model Information
 - 138 Layers
 - 2 billion MACs
 - GPU runtime forward propagate time is 53 ms
 - Model input is a camera image scaled down to 224x224
 - Model output is bounding box and heat map info

Thank you

All data and information contained in or disclosed by this document is confidential and proprietary information of Qualcomm Technologies, Inc. and all rights therein are expressly reserved. By accepting this material the recipient agrees that this material and the information contained therein is to be held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

© 2016 QUALCOMM Incorporated and/or its subsidiaries. All Rights Reserved.
Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries.
Other products and brand names may be trademarks or registered trademarks of their respective owners

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable.

Qualcomm Incorporated includes Qualcomm’s licensing business, QTL, and the vast majority of its patent portfolio. Qualcomm Technologies, Inc., a wholly-owned subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of Qualcomm’s engineering, research and development functions, and substantially all of its product and services businesses, including its semiconductor business.



SNPE Demo Overview

- “Scenes” model running on SNPE inside an Android app
- Model Information
 - 38 Layers
 - 500 million MACs
 - GPU runtime forward propagate time is 16.1 ms
 - Model input is a camera image scaled down to 224x224
 - Model output is one or more labels describing the entire scene

Intel® SDK for OpenCL™ – Code Builder

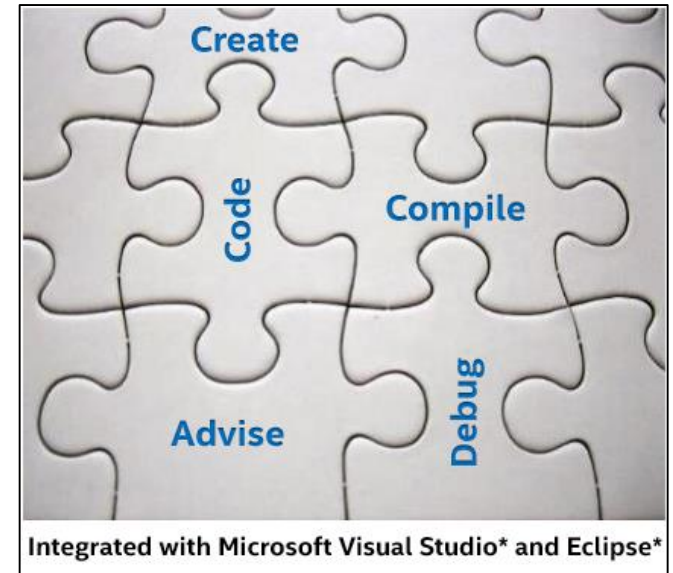
A comprehensive developer tool-chain for OpenCL™:

- Supports Each State of OpenCL™ Code Development: Build, Debug and Analyze
- Full IDE Integration: Microsoft Visual Studio* on Windows* and Eclipse* on Linux*



Now with SPIR-V support!

- Produce SPIR-V binaries for OpenCL™ C Code with the integrated **SPIR-V Generator**
- Examine the textual representation of the SPIR-V binary with the integrated **SPIR-V Disassembler**
- Develop SPIR-V based OpenCL™ applications today with the **OpenCL™ 2.1 CPU-Only Experimental Runtime**



Generating SPIR-V Binaries with the Intel® OpenCL™ Offline Compiler:

```
...\> ioc64.exe -input=program.cl -spirv32 -text-spirv32 -spirv64 -text-spirv64
```

Contains the OpenCL™ C Code

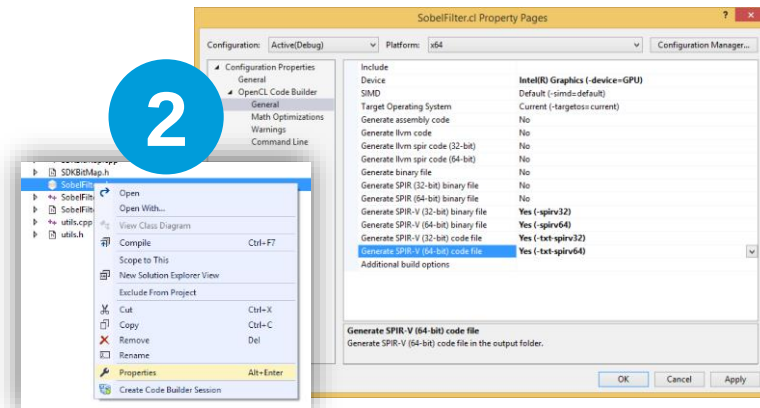
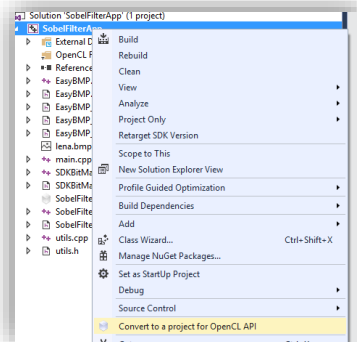
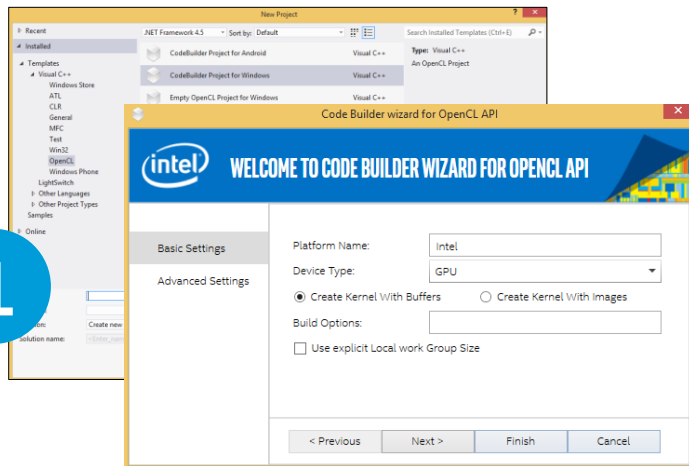
Generates a 32-bit SPIR-V binary (program_x86.spirv)

Generates a 32-bit SPIR-V Code (program_x86.txtspirv)

Generates a 64-bit SPIR-V binary (program_x64.spirv)

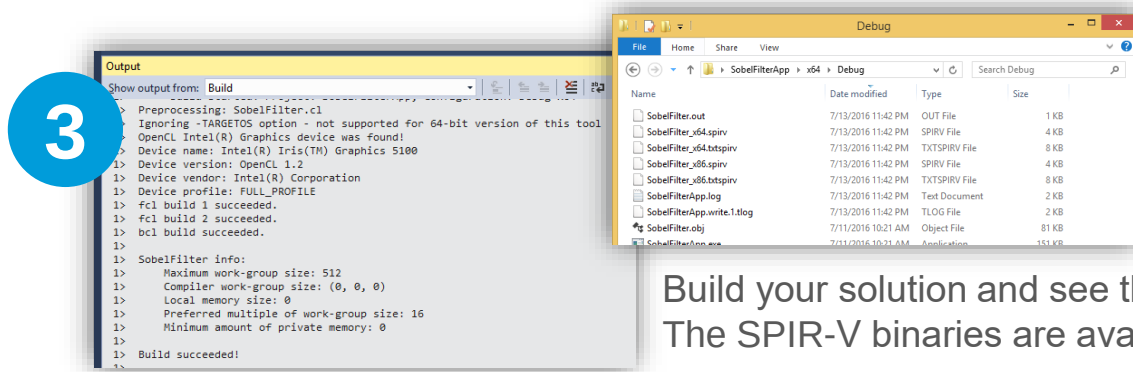
Generates a 64-bit SPIR-V Code (program_x86.txtspirv)

Generating SPIR-V Binaries through the Visual Studio* IDE:



Create new OpenCL™ project with the Visual Studio* project template wizard, or convert existing C/C++ project to OpenCL™ project.

Enable SPIR-V generation for each .cl (OpenCL™ C code) file in the project's properties.



Build your solution and see the OpenCL™ compiler build log for each .cl file. The SPIR-V binaries are available to use in the project's output folder.

SPIR-V Sample

Check out our article on Intel® Developer Zone:

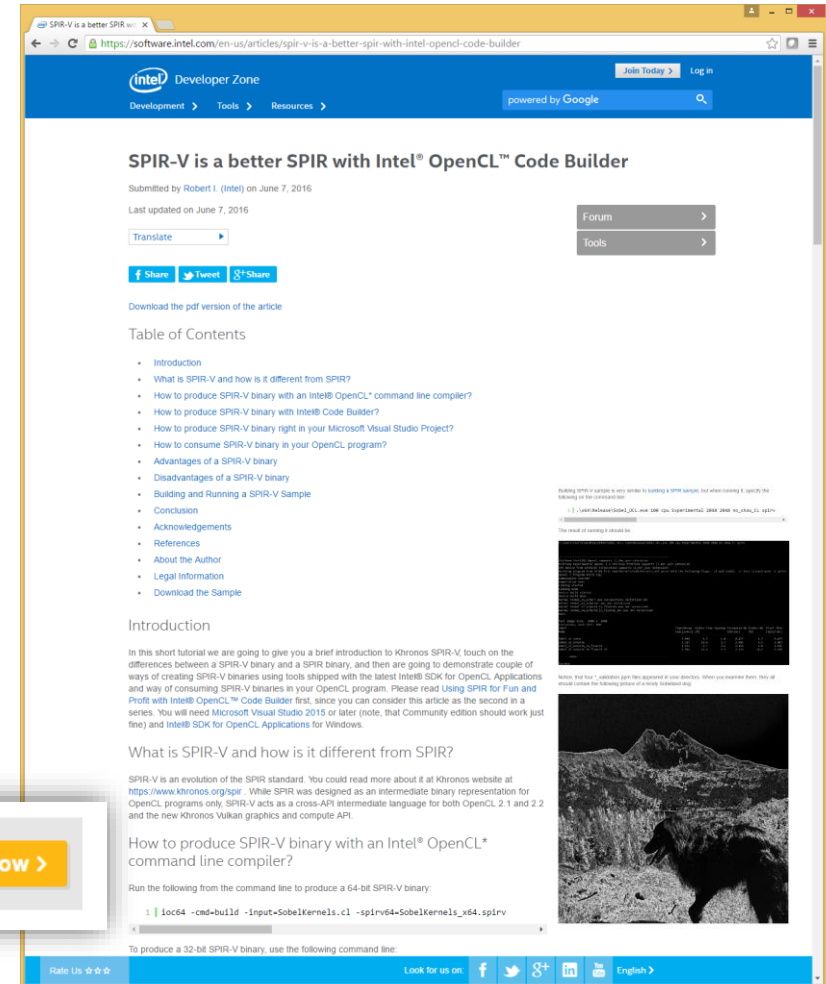
SPIR-V is a better SPIR with Intel® OpenCL™ Code Builder

<https://software.intel.com/en-us/articles/spir-v-is-a-better-spir-with-intel-opencl-code-builder>

SPIR-V introduction, code examples, tools, and tutorials are available on Intel® Developer Zone!

There are downloads available under the [Intel Sample Source Code License Agreement](#) license.

[Download Now >](#)



Call to Action

- Try it out!
- Khronos is seeking feedback on OpenCL 2.2 and OpenCL C++:
 - For OpenCL 2.2:
 - https://www.khronos.org/opencl/opencl_feedback_forum
 - For the OpenCL C++ Compiler, file a github “Issue”:
 - <https://github.com/KhronosGroup/SPIR/issues>
 - <https://github.com/KhronosGroup/libclcxx/issues>
- Reminder: Any company or organization is welcome to join Khronos for a voice and a vote in any of these standards
 - www.khronos.org

