EVALUATION OF MODERN GPGPU TECHNOLOGIES FOR IMAGE PROCESSING

JOACHIM MEYER
IMAGE PROCESSING
GPGPU ENGINEER

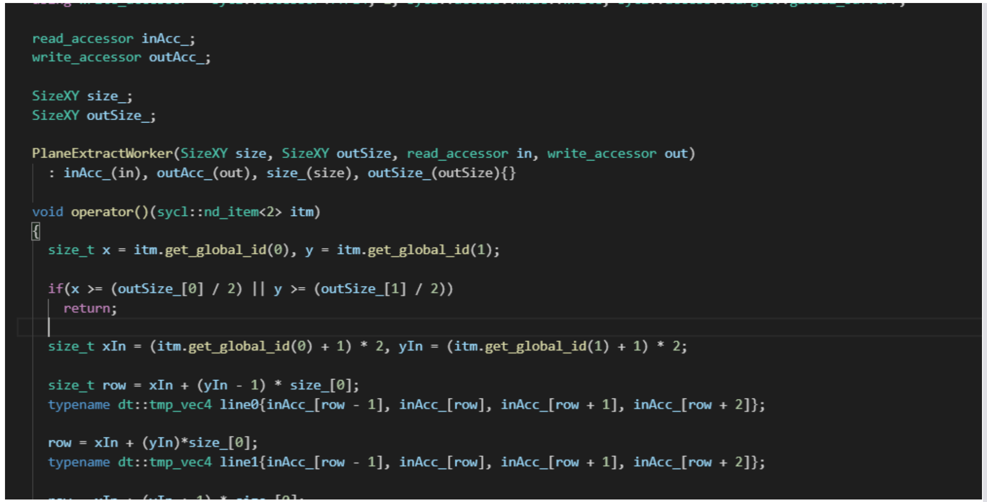
VISION.
RIGHT.
NOW.

(TOO?) MANY DIFFERENT GPGPU PROGRAMMING MODELS / APIS

Vulkan®
OpenGL|ES™
OpenGL®
SYCL™
OpenCL
NVIDIA®
CUDA®
ROCm
ON
OpenMP™
XII

(TOO?) MANY DIFFERENT GPGPU PROGRAMMING MODELS / APIS

Which one fits this awesome new project?



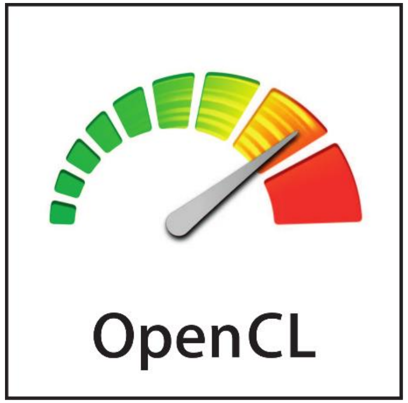
```
read_accessor inAcc_;
write_accessor outAcc_;

SizeXY size_;
SizeXY outSize_;

PlaneExtractWorker(SizeXY size, SizeXY outSize, read_accessor in, write_accessor out)
  : inAcc_(in), outAcc_(out), size_(size), outSize_(outSize){}

void operator()(sycl::nd_item<2> itm)
{
  size_t x = itm.get_global_id(0), y = itm.get_global_id(1);

  if(x >= (outSize_[0] / 2) || y >= (outSize_[1] / 2))
    return;

  size_t xIn = (itm.get_global_id(0) + 1) * 2, yIn = (itm.get_global_id(1) + 1) * 2;

  size_t row = xIn + (yIn - 1) * size_[0];
  typename dt::tmp_vec4 line0{inAcc_[row - 1], inAcc_[row], inAcc_[row + 1], inAcc_[row + 2]};

  row = xIn + (yIn)*size_[0];
  typename dt::tmp_vec4 line1{inAcc_[row - 1], inAcc_[row], inAcc_[row + 1], inAcc_[row + 2]};
```

AGENDA

- SELECTION OF COMPARED APIS
- EVALUATION SETUP
- PERFORMANCE
- USABILITY
- PLATFORM INDEPENDENCE
- CONCLUSION
- FUTURE PROSPECTS

BASICS

Selection of APIs



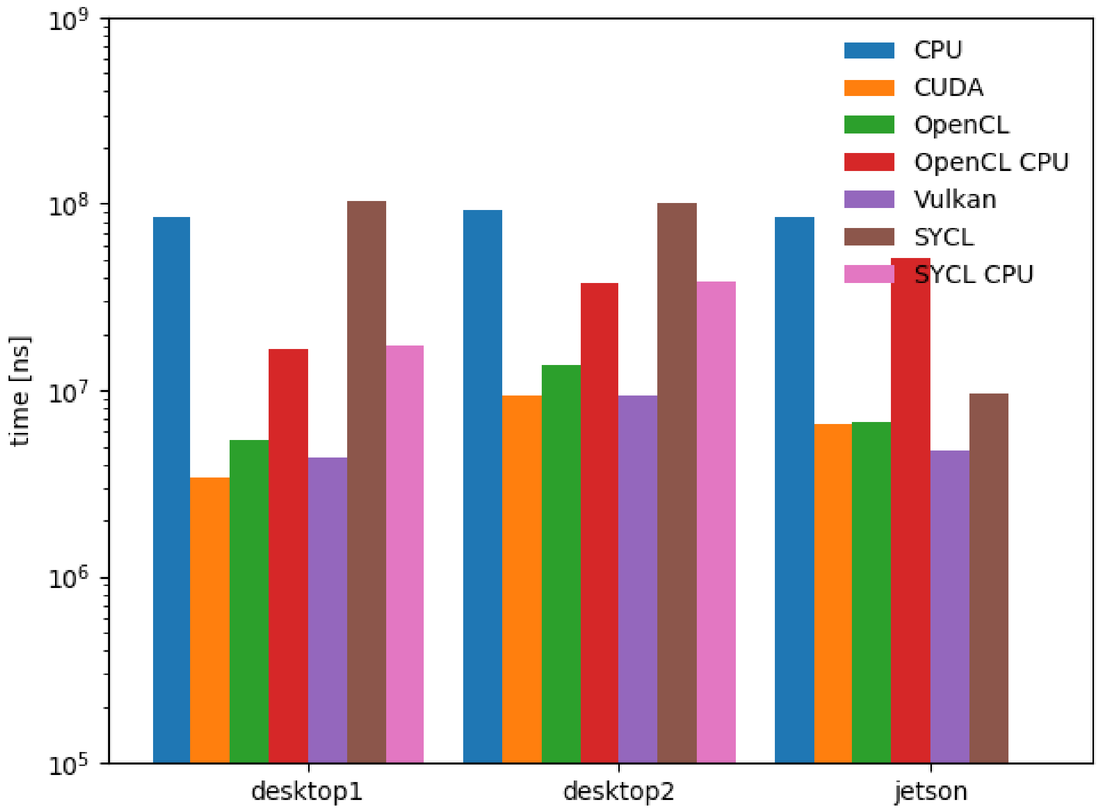
BASICS

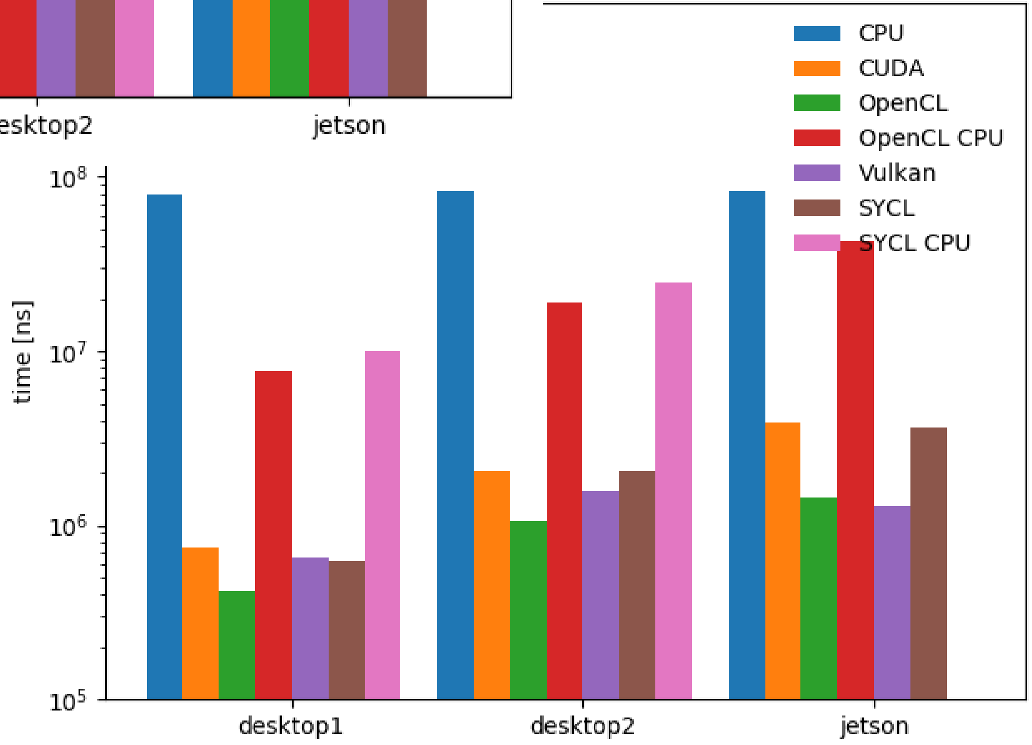
Test Project setup

- Targeting all 4 APIs + CPU reference implementation
- Targeted devices: CPUs & GPUs
- OSs: Windows & Linux 64bit
- Implementations:
  - CUDA 10.1
  - Vulkan 1.1
  - OpenCL 1.2
  - ComputeCpp (Win) & hipSYCL (Linux)
- Algorithms for polarization camera image processing

It’s comparable.

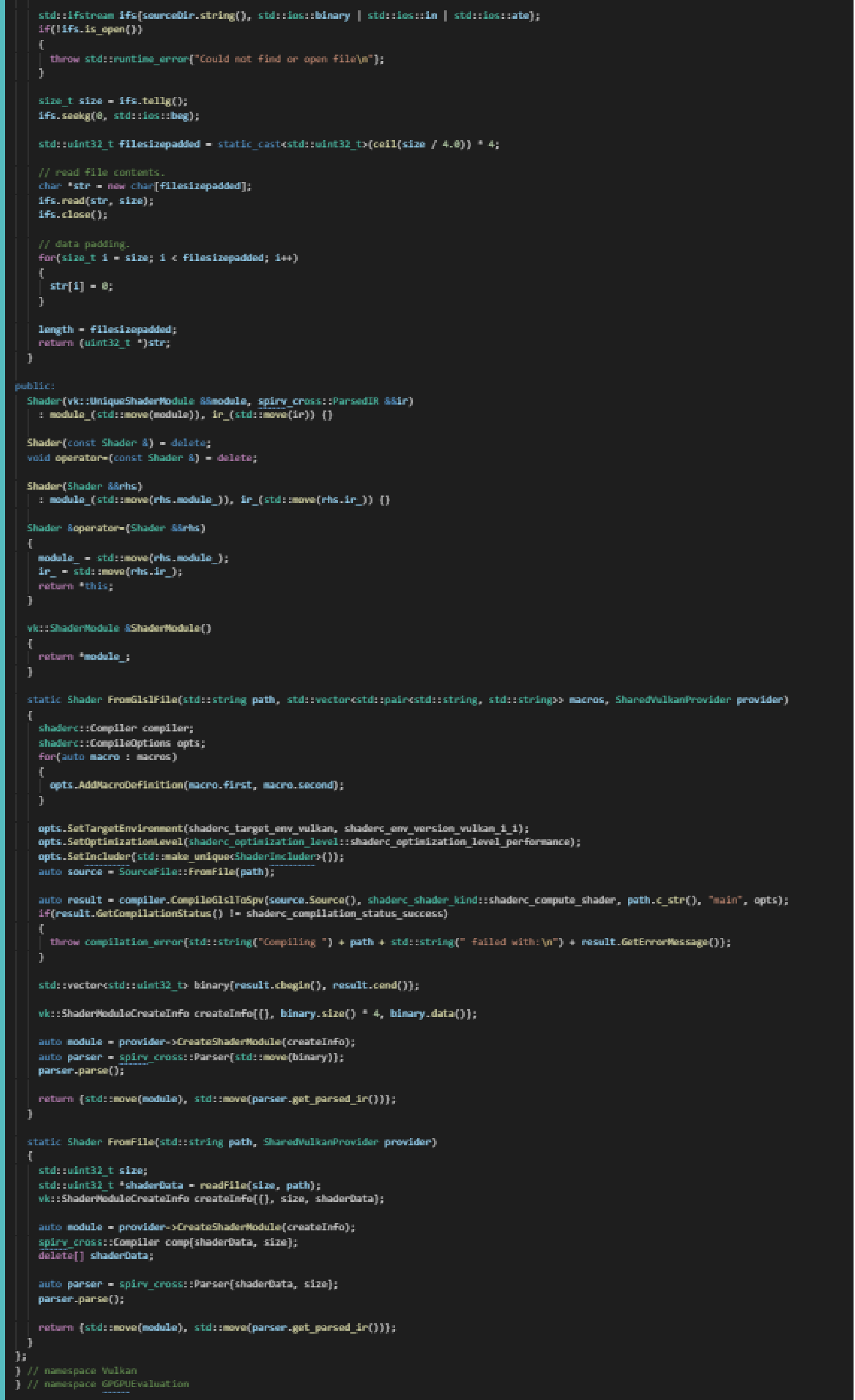
PERFORMANCE



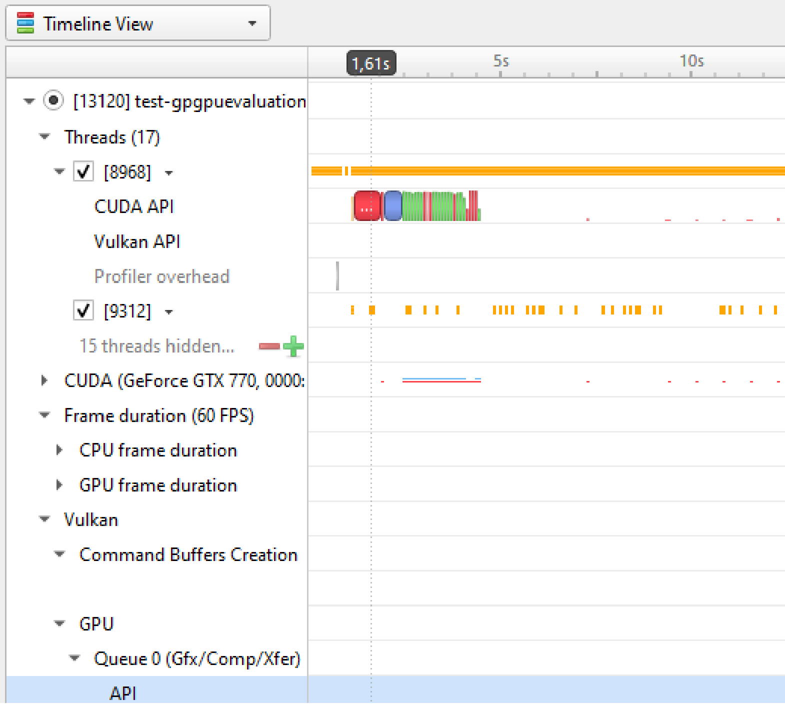


How hard is it?

USABILITY



```
    std::ifstream ifs(sourceDir.string(), std::ios::binary | std::ios::in | std::ios::ate);
    if(!ifs.is_open())
    {
      throw std::runtime_error("Could not find or open file\n");
    }

    size_t size = ifs.tellg();
    ifs.seekg(0, std::ios::beg);

    std::uint32_t filesizepadded = static_cast<std::uint32_t>(ceil(size / 4.0)) * 4;

    // read file contents.
    char *str = new char[filesizepadded];
    ifs.read(str, size);
    ifs.close();

    // data padding.
    for(size_t i = size; i < filesizepadded; i++)
    {
      str[i] = 0;
    }

    length = filesizepadded;
    return (uint32_t *)str;
  }

public:
  Shader(vk::UniqueShaderModule &&module, spirv_cross::ParsedIR &&ir)
    : module_(std::move(module)), ir_(std::move(ir)) {}

  Shader(const Shader &) = delete;
  void operator=(const Shader &) = delete;

  Shader(Shader &&rhs)
    : module_(std::move(rhs.module_)), ir_(std::move(rhs.ir_)) {}

  Shader &operator=(Shader &&rhs)
  {
    module_ = std::move(rhs.module_);
    ir_ = std::move(rhs.ir_);
    return *this;
  }

  vk::ShaderModule &ShaderModule()
  {
    return *module_;
  }

  static Shader FromGlslFile(std::string path, std::vector<std::pair<std::string, std::string>> macros, SharedVulkanProvider provider)
  {
    shaderc::Compiler compiler;
    shaderc::CompileOptions opts;
    for(auto macro : macros)
    {
      opts.AddMacroDefinition(macro.first, macro.second);
    }

    opts.SetTargetEnvironment(shaderc_target_env_vulkan, shaderc_env_version_vulkan_1_1);
    opts.SetOptimizationLevel(shaderc_optimization_level::shaderc_optimization_level_performance);
    opts.SetIncluder(std::make_unique<ShaderIncluder>());
    auto source = SourceFile::FromFile(path);

    auto result = compiler.CompileGlslToSpv(source.Source(), shaderc_shader_kind::shaderc_compute_shader, path.c_str(), "main", opts);
    if(result.GetCompilationStatus() != shaderc_compilation_status_success)
    {
      throw compilation_error(std::string("Compiling ") + path + std::string(" failed with:\n") + result.GetErrorMessage());
    }

    std::vector<std::uint32_t> binary(result.cbegin(), result.cend());

    vk::ShaderModuleCreateInfo createInfo({}, binary.size() * 4, binary.data());

    auto module = provider->CreateShaderModule(createInfo);
    auto parser = spirv_cross::Parser(std::move(binary));
    parser.parse();

    return {std::move(module), std::move(parser.get_parsed_ir())};
  }

  static Shader FromFile(std::string path, SharedVulkanProvider provider)
  {
    std::uint32_t size;
    std::uint32_t *shaderData = readFile(size, path);
    vk::ShaderModuleCreateInfo createInfo({}, size, shaderData);

    auto module = provider->CreateShaderModule(createInfo);
    spirv_cross::Compiler comp(shaderData, size);
    delete[] shaderData;

    auto parser = spirv_cross::Parser(shaderData, size);
    parser.parse();

    return {std::move(module), std::move(parser.get_parsed_ir())};
  }
};
} // namespace Vulkan
} // namespace GPGPUEvaluation
```



WHAT'S THE IMPLEMENTATION COST?

	CUDA	SYCL	OpenCL	Vulkan
LoC basic setup	4	5	6	65
LoC realistic setup	25	27	34	128 (+ 25 GLSL→SPIRV)
LoC / new kernel	4	5	6	11
C++ kernels	✓	✓	✓	
Implicit asynchronity	✓	✓	✓	
Taskgraph	✓	✓		

ANY TOOLS TO HELP?

CUDA

- Solid dev tooling:
  - kernel debugging
  - profiling
  - IDE integration

SYCL

- Hardly any specific tools, but native OpenCL / HIP tools usable
- Host-device enables native IDE debugging

OpenCL

- Mostly vendor specific dev tools
- LPGPU^2 CodeXL: generalization of AMD project

Vulkan

- Mainly graphics focused tooling
- Validation layers
- Emulator (Talvos)

LIBRARIES?

CUDA

- Many optimized libraries
  - FFT, BLAS, image processing, …

SYCL

- Some libraries
  - BLAS, DNN, RNG, Parallel STL, image processing
  - Native (OpenCL / HIP) libraries usable

OpenCL

- Number of libraries with some device-specific optimization
  - FFT, BLAS, DGEMM, image processing, ...

Vulkan

- Hardly any Compute specific libraries

APPLICATION ADOPTION AS INDICATOR FOR COMMUNITY SUPPORT

HELP ANYONE?

CUDA

- Widely used by scientists and application devs
  - De-facto standard in ML libraries

- SO Questions: 12.380

SYCL

- Few applications known
  - Tensorflow
  - Eigen

- SO Questions: 28

OpenCL

- Wide adoption in consumer applications
  - Adobe Creative Cloud
  - Final Cut Pro

- SO Questions: 5.040

Vulkan

- Increasing adoption for mobile device support / combined with graphics
  - Adobe Premier Rush
  - OcataneRender

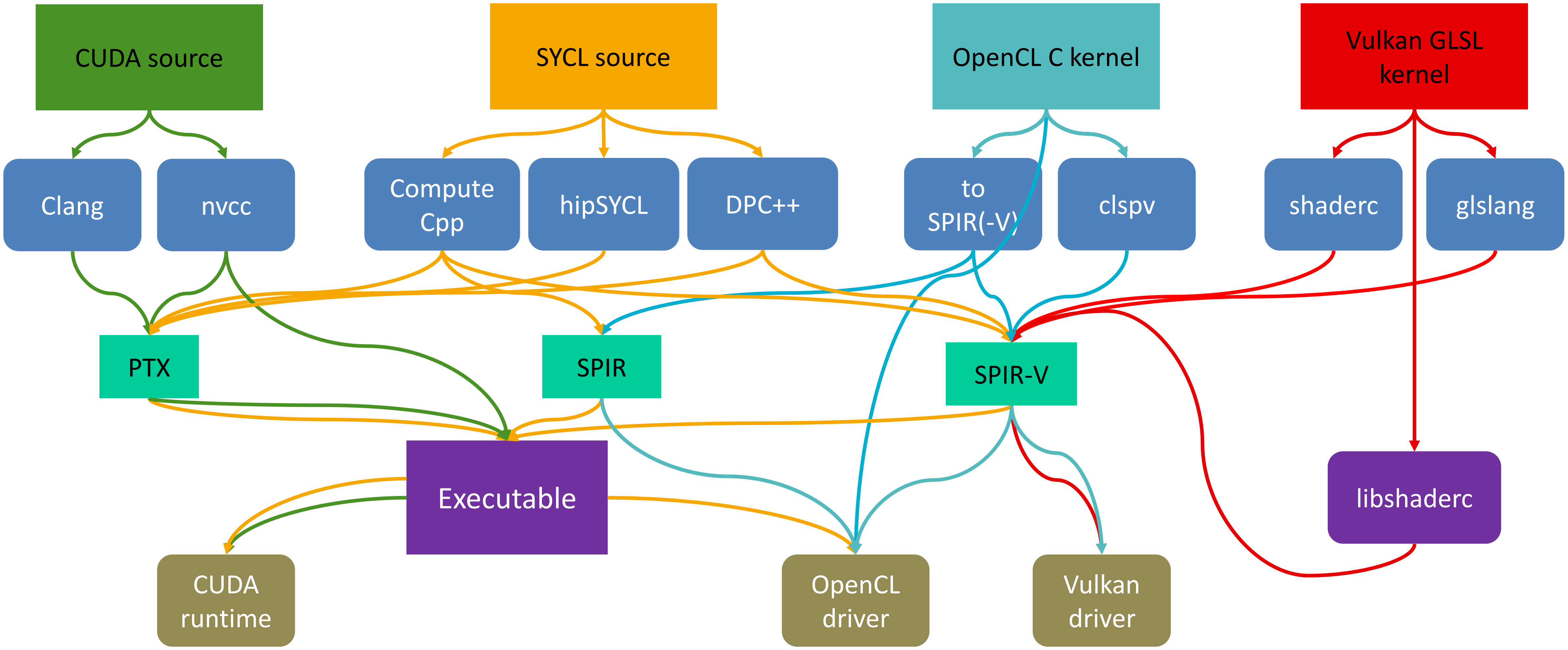
- SO Questions: 1.020

Stack Overflow Question tags counted from 2. March 20

EACH API HAS ITS OWN (KERNEL) COMPILATION WORKFLOW

STEMMER® IMAGING

HOW DOES THE CODE COME TO LIFE?



HOW TO HANDLE DYNAMIC DATA TYPES?

CUDA

Generic programming

SYCL

Generic programming

OpenCL

(Dynamic) online compilation with required data type as macro

Preprocessor programming to dispatch temporary data types

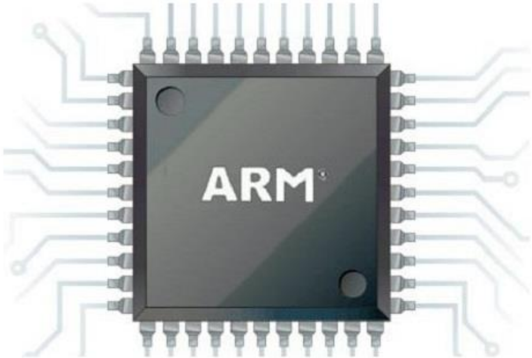
Vulkan

Online / offline compilation with required data type as macro / in shader names

Preprocessor programming to dispatch temporary data types

Can it target XYZ?

PLATFORM INDEPENDENCE



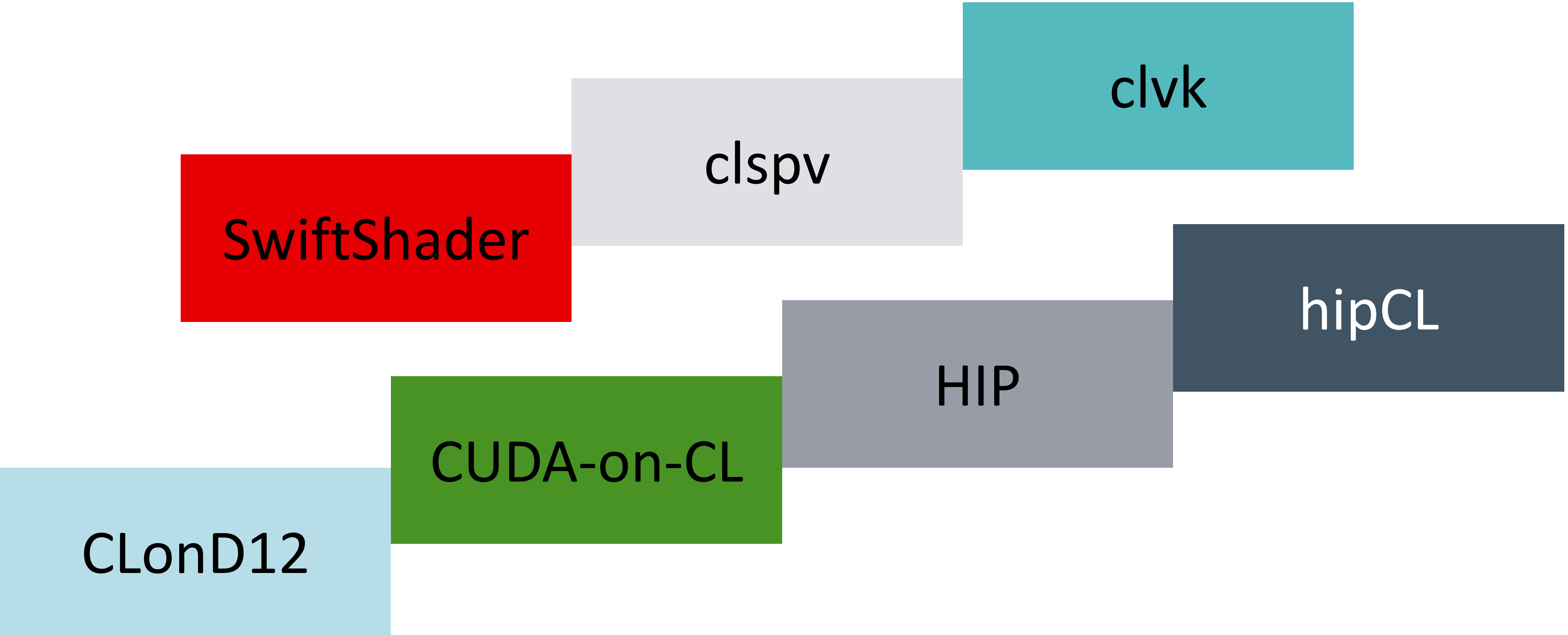
WHAT HARDWARE CAN BE TARGETED USING WHICH VERSION?

CAN IT TARGET XYZ?

	CUDA	SYCL	OpenCL	Vulkan
Most recent version	10.2	1.2.1	2.2	1.2
Nvidia	10.2	1.2.1	1.2	1.2
AMD	HIP	1.2.1	2.0	1.2
Intel		1.2.1	2.1	1.2
ARM		1.2.1	2.1	1.2
Windows	✓	✓	✓	✓
Linux	✓	✓	✓	✓
macOS	✓		✓	✓
Android			✓	✓
CPU		✓	✓	✓
FPGAs		✓	✓	

PROJECTS INCREASING PORTABILITY OF THE APIS

PORTABILITY INITIATIVES



SwiftShader

clspv

clvk

CLonD12

CUDA-on-CL

HIP

hipCL

CONCLUSION

So which API should be used?

CUDA

Single-source programming

Highly optimized and powerful libraries and tools

Vendor lock-in acceptable?

(Maybe use HIP instead?)

SYCL

Single-source programming

Multi-platform (incl. FPGAs,..)

Tools for underlying implementation usable

Emerging SYCL-specific tool and library support

OpenCL

Cross-platform (incl. FPGAs,..)

Mature libraries

Big community

Not-up-to-date implementations

Vulkan

Fully OS and GPU-vendor independent

High setup cost but possibility to optimize

Lack of compute specific tooling & libraries

Full decision matrix: doi.org/10.1145/3388333.3388645

FUTURE PROSPECTS

SOME POSSIBLE DEVELOPMENTS

WHAT‘S UP NEXT?

CUDA

- Extended ARM & data-center support
- Continuous optimization and feature updates
- Fast support for new GPU features
- HIP porting to Windows?

SYCL

- Maturing and optimization of implementations
- Extended hardware and OS support
- Removal of OpenCL as conformance required backend
- Specific tooling and libraries
- News @ IWOCL
- SYCL-on-Vulkan?

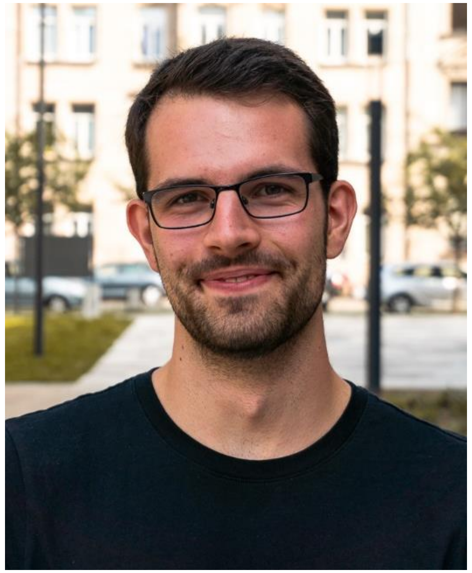
OpenCL

- (Hopefully) improved vendor support with OpenCL Next
- Updated to new hardware features
- Higher-level kernel language support
- News @ IWOCL

Vulkan

- Continued wide support
- Extended compute capabilities to serve as portability backend for other APIs
- Compute specific libraries & tools
- Fast support for new GPU features

THANK YOU VERY MUCH FOR YOUR ATTENTION



JOACHIM MEYER

STEMMER IMAGING AG
J.MEYER@STEMMER-IMAGING.COM
STEMMER-IMAGING.COM
JOAMEYER.DE