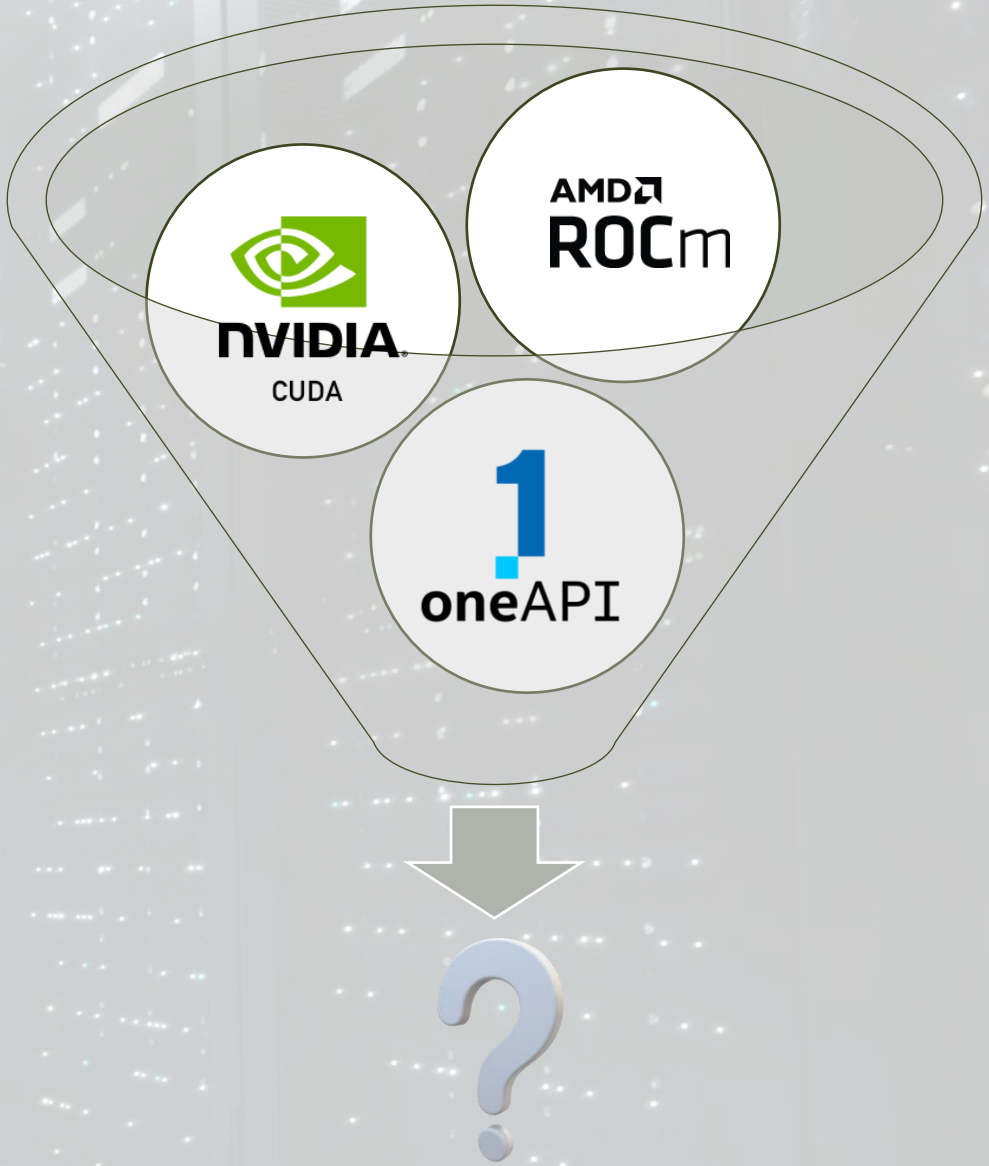
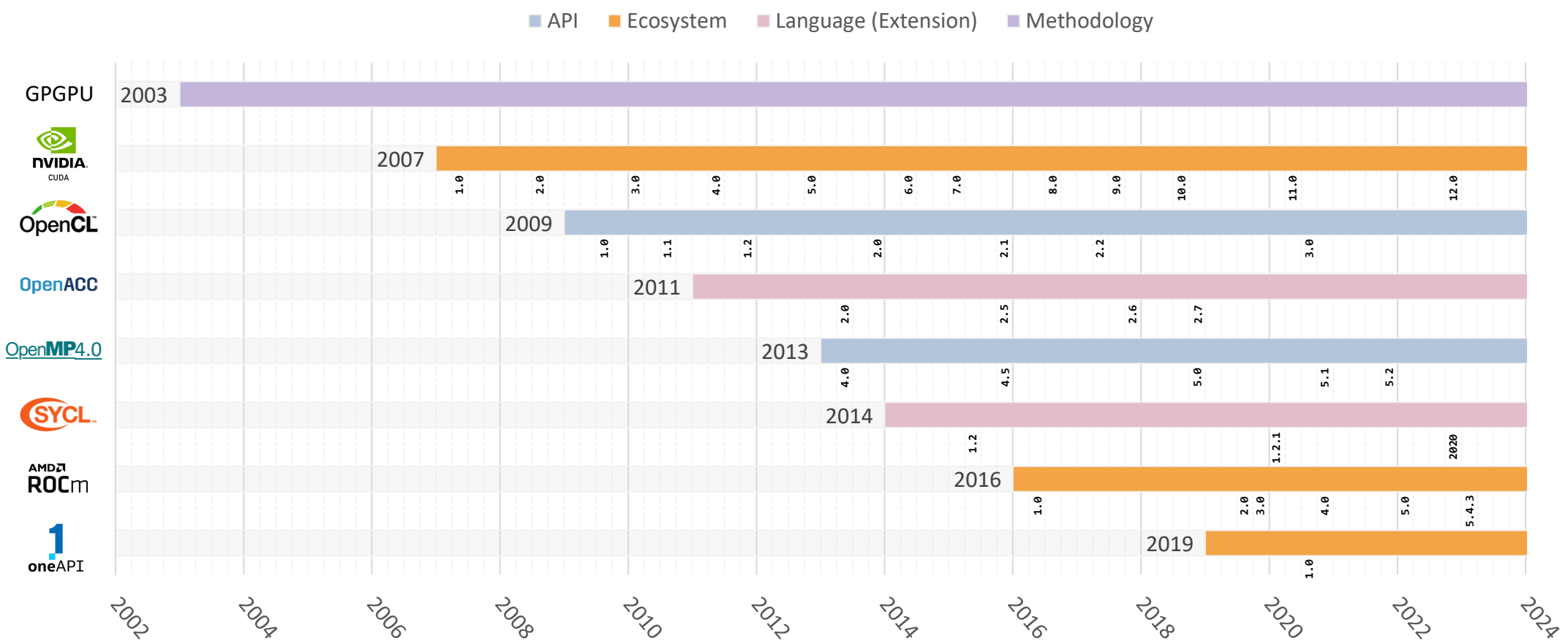
CUDA, ROCm, oneAPI

All for One
or
One for All?

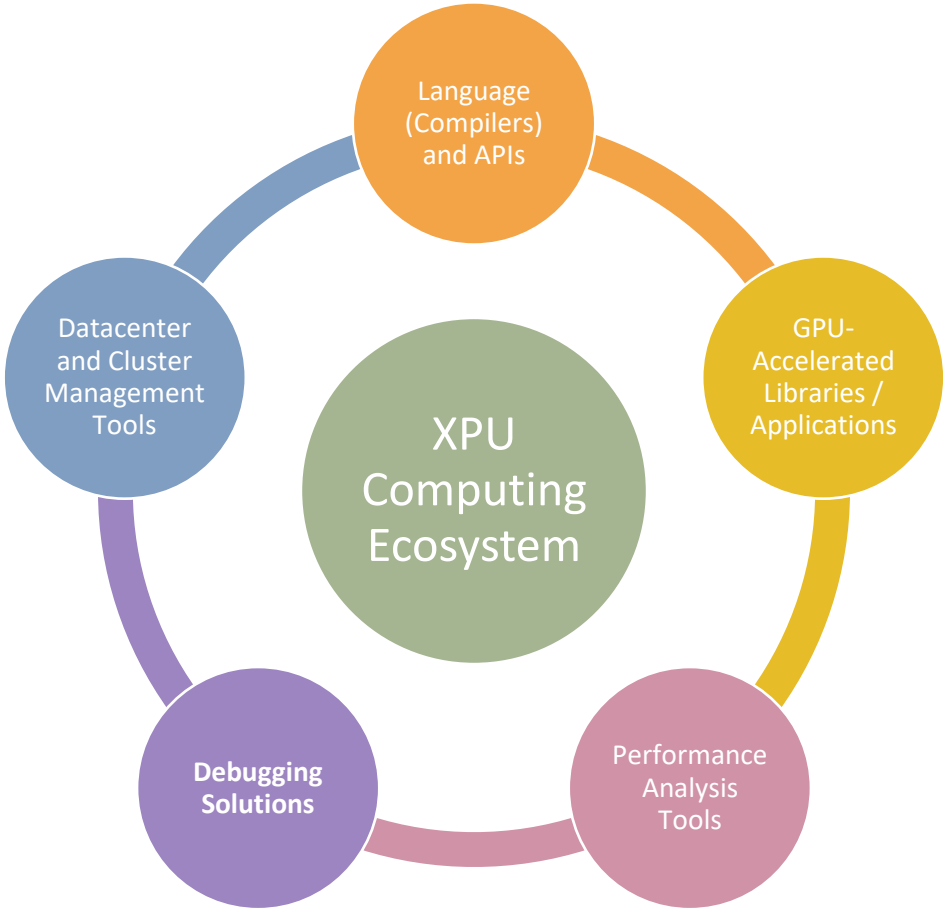
Armin Sobhani
asobhani@sharcnet.ca
https://staff.sharcnet.ca/asobhani
SHARCNET | Compute Ontario
HPC Technical Consultant
OntarioTech UNIVERSITY



GPU Computing Timeline



The XPU/ Heterogenous Computing Ecosystem



CUDA Ecosystem

Deployment Tools	NVIDA SMI		Data Center GPU Manager (DCGM)		GPU REST Engine (GRE)
Libraries	cuBLAS	cuFFT	cuSPARSE	cuSOLVER	AMG-X
	Thrust	CUB	cuDNN	cuRAND	NCCL
Compilers & Tools	Compilers nvcc, nvc, nvc++, nvfortran	CUDA-GDB	NVIDIA Nsight	NVIDIA Visual Profiler	PAPI CUDA
Programming Models	CUDA	OpenMP API	OpenACC	OpenCL	PyCUDA
Drivers / Runtimes	*Linux* and *Windows* Device Drivers and Runtime (no macOS anymore)				

ROCm Ecosystem

AMD ROCm

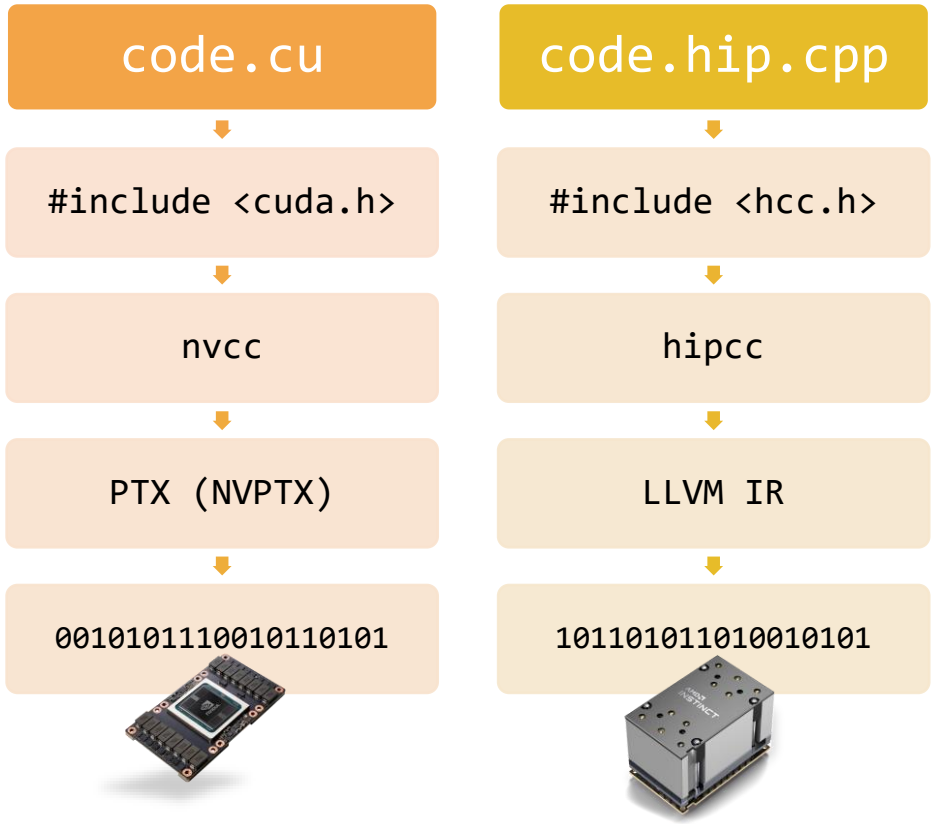
Deployment Tools	ROCm SMI		ROCm Data Center Tool		ROCm Validation Suite
Libraries	rocBLAS	rocFFT	rocSPARSE	rocSOLVER	rocALLUTION
	rocThrust	rocPRIM	MIOpen	rocRAND	RCCL
Compilers & Tools	Compilers hipcc, hipfc	rocGDB	rocProfiler	hipify gpufort	TENSILE
Programming Models	HIP API		OpenMP API		OpenCL
Drivers / Runtimes	*Linux* (RedHat, SLES and Ubuntu) Device Drivers and Runtime				

oneAPI Ecosystem

Deployment Tools	sycl-ls		Intel® Cluster Checker		Intel® MPI Library
Libraries	oneMKL	oneMKL	oneMKL	oneMKL	oneMKL
	oneDPL	oneTBB	oneDNN	oneMKL	oneCCL
Compilers & Tools	Compilers icx, icpx, ifx, dpcpp	Intel® GDB	VTune Profiler	c2s (dpct)	Intel® Inspector
Programming Models	SYCL		OpenMP API		OpenCL
Drivers / Runtimes	*Linux*, *Windows* and *macOS* Device Drivers and Runtime				

Compile-Time vs. Run-Time Platform Targeting

Compile-time (CUDA / ROCm)



Run-time (oneAPI / SYCL / OpenCL)

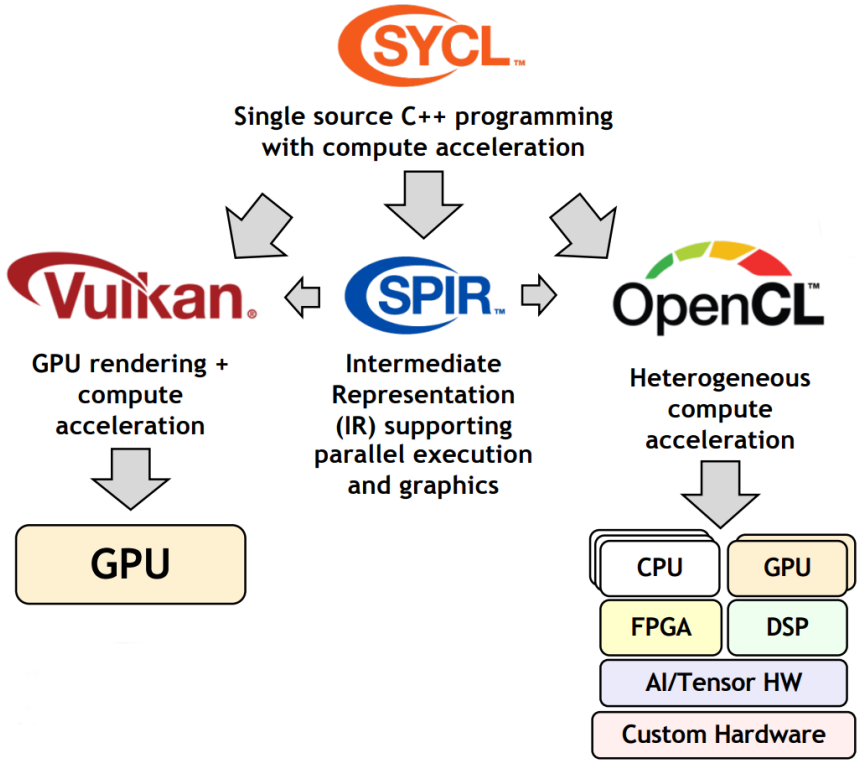


Image courtesy of khronos.org

Running SYCL Programs on NVIDIA/AMD GPUs

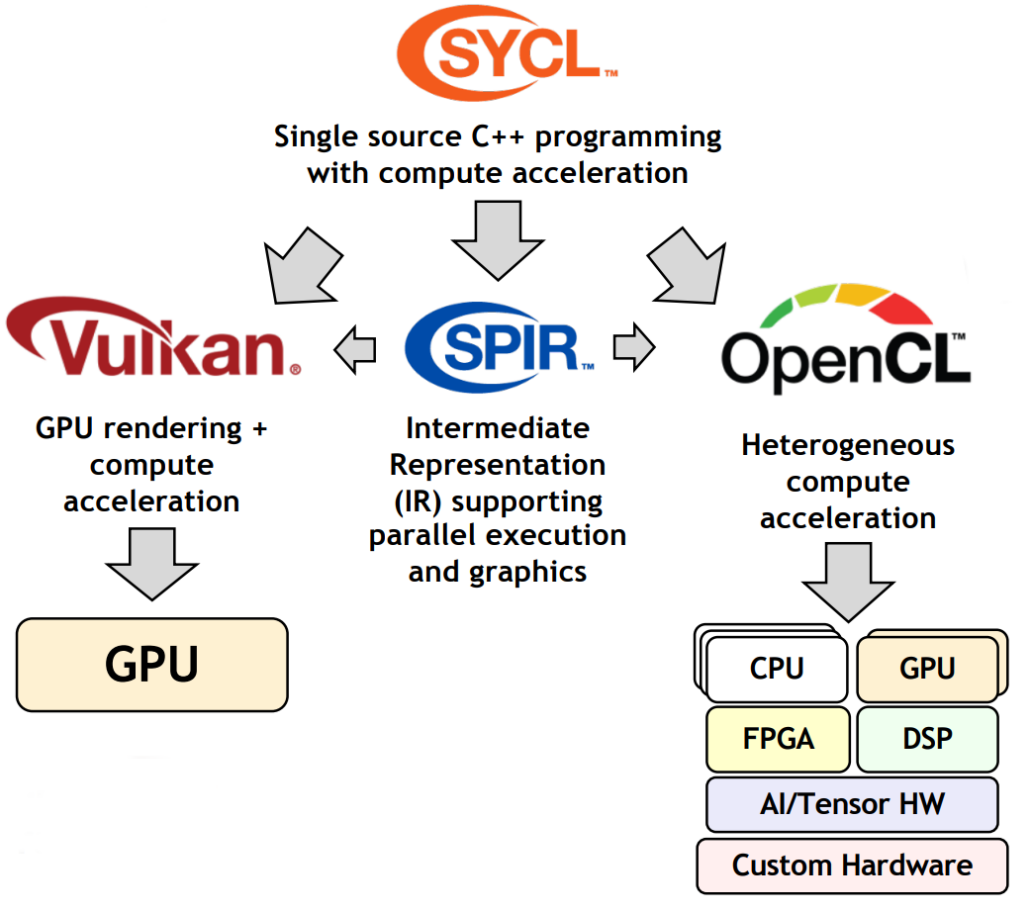
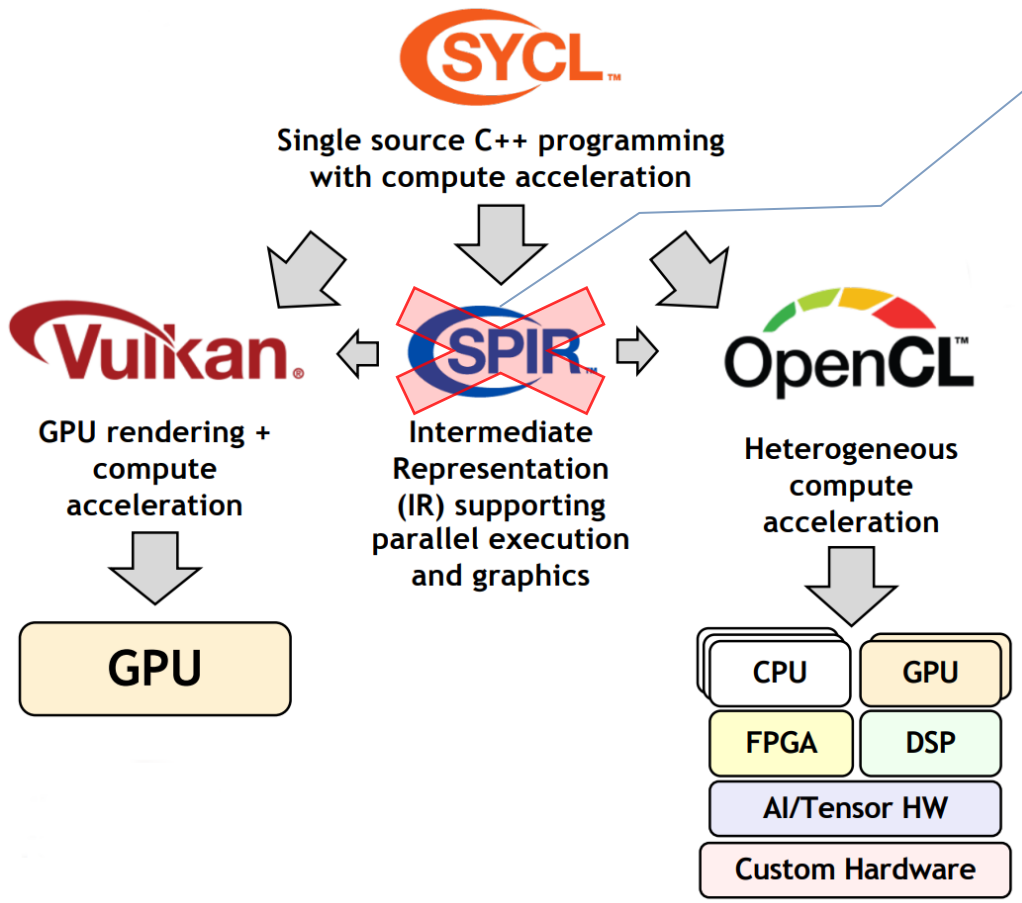


Image courtesy of khronos.org

Running SYCL Programs on NVIDIA/AMD GPUs



Neither NVIDIA's nor AMD's OpenCL have never supported SPIR(-V)

Image courtesy of khronos.org

But that doesn't mean it's Impossible…

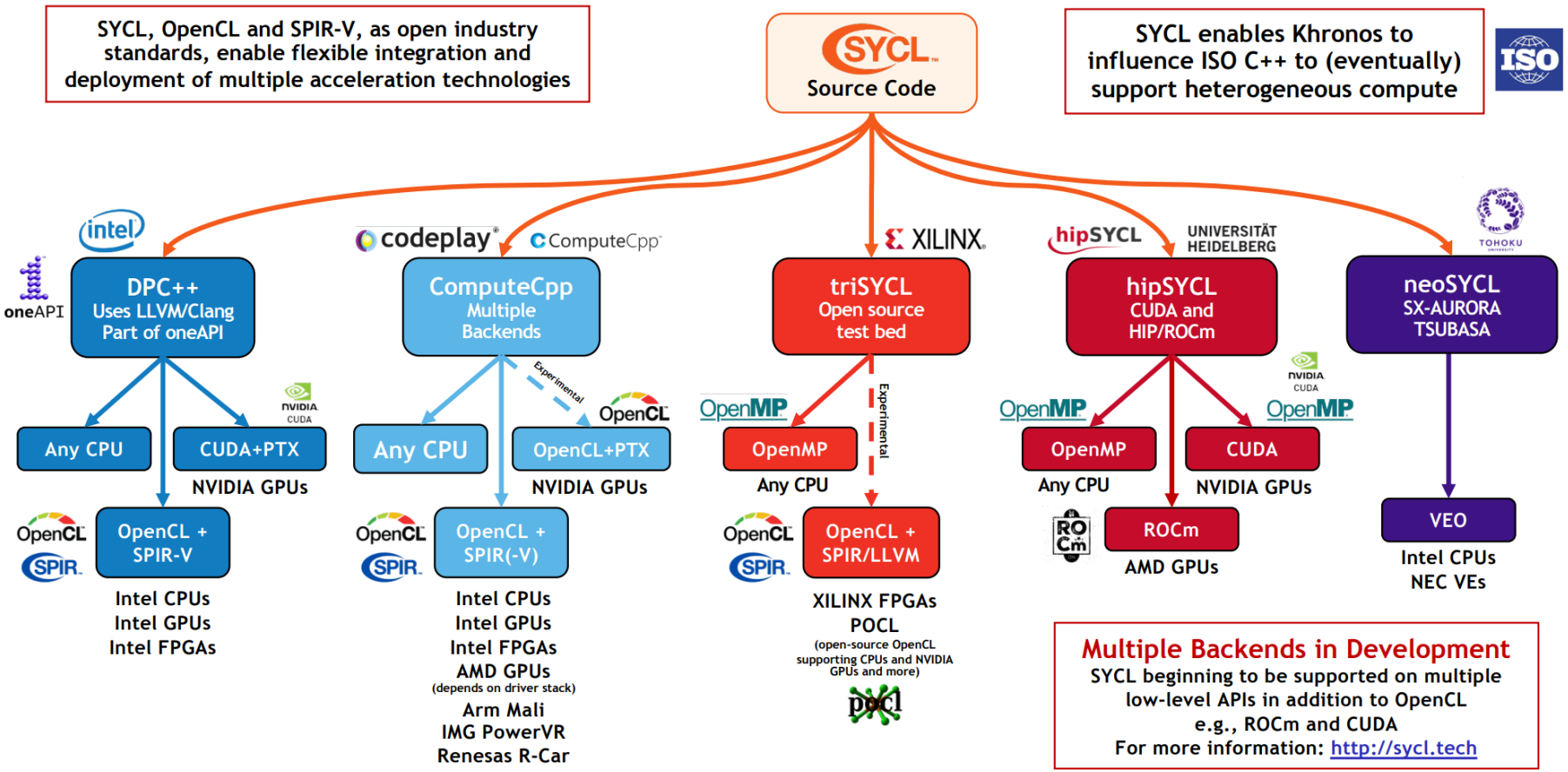


Image courtesy of khronos.org

But that doesn't mean it's Impossible…

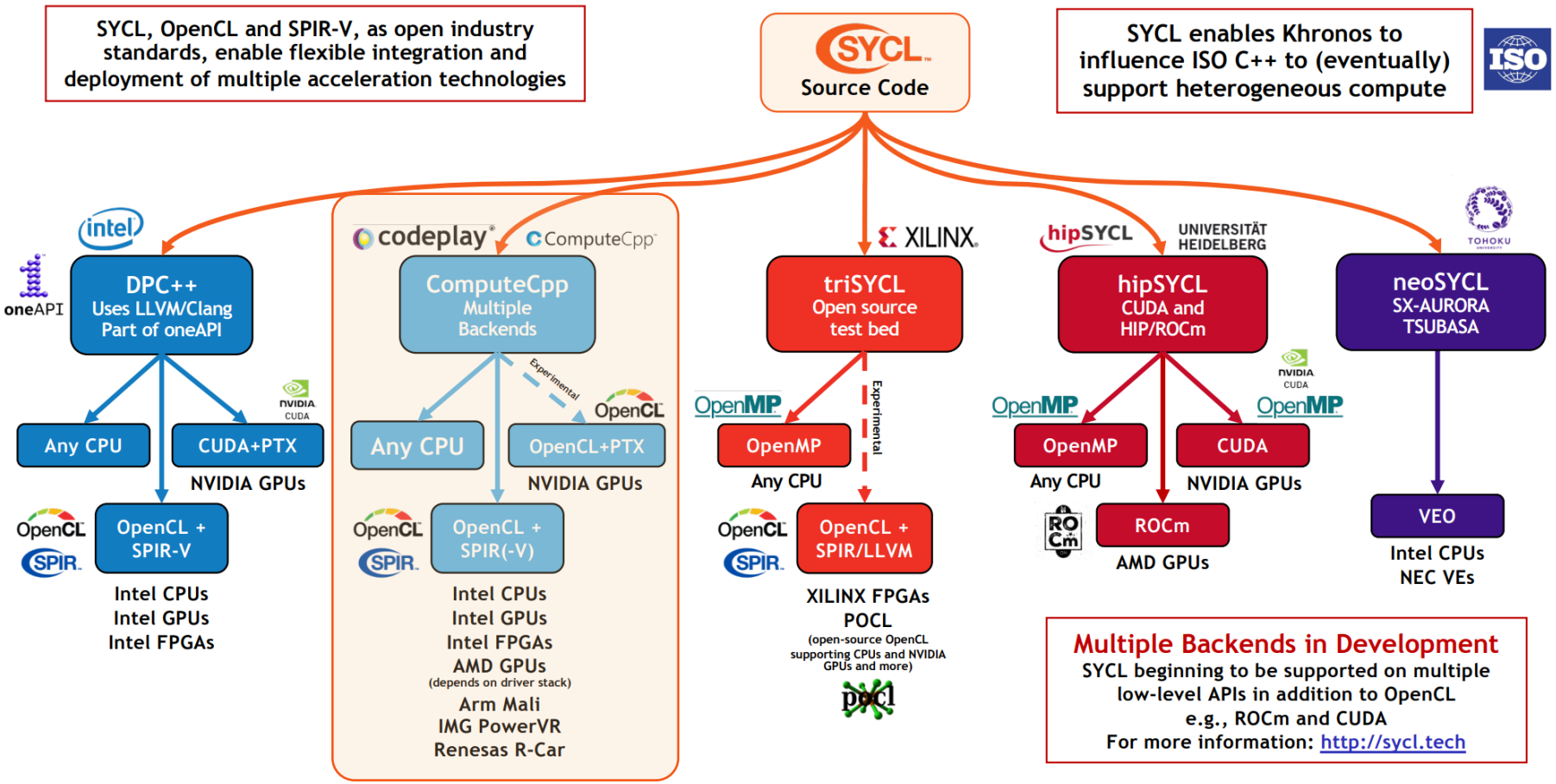


Image courtesy of khronos.org

oneAPI for NVIDIA and AMD GPUs from Codeplay

Plugin for NVIDIA GPUs

https://developer.codeplay.com/products/oneapi/nvidia

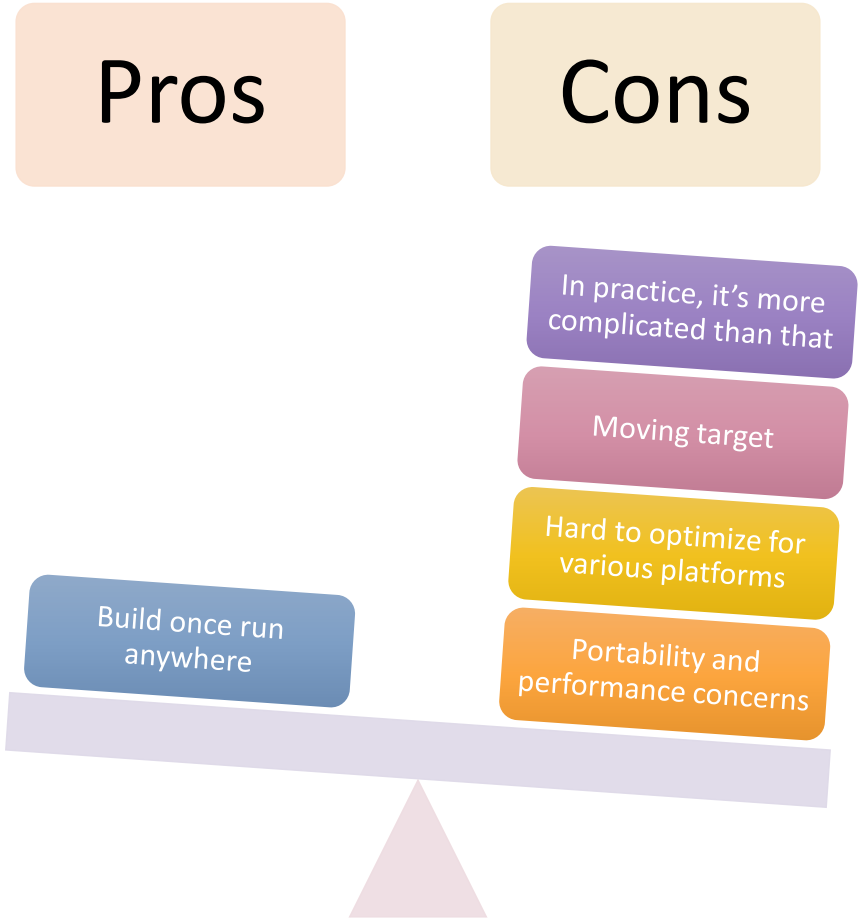
Plugin for AMD GPUs

https://developer.codeplay.com/products/oneapi/amd

One or All?

Shall we go with oneAPI and ditch other musketeers...

TL;DR – Not Now! Revisit in 5 Years…



One or All?

Why not go with all of them then?

The more the merrier!

Terminology and Syntax 'Cheat Sheet'

TERM	CUDA	ROCm	OpenCL	SYCL
Device	int deviceId	int deviceId	cl_device	sycl::device
Queue	cudaStream_t	hipStream_t	cl_command_queue	sycl::queue
Event	cudaEvent_t	hipEvent_t	cl_event	sycl::event
Memory	void*	void*	cl_mem	sycl::buffer
Grid of threads	grid	grid	NDRange	NDRange
Subgroup of threads	block	block	work-group	work-group
Thread	thread	thread	work-item	work-item
Scheduled execution	warp	Warp	sub-group (warp, wavefront, etc.)	sub-group (warp, wavefront, etc.)
Thread-index	threadIdx.{x,y,z}	hipThreadIdx.{x,y,z}, threadIdx.{x,y,z}*	get_local_id({0,1,2})	sycl::nd_item::get_local_id({0,1,2})
Block-index	blockIdx.{x,y,z}	hipBlockIdx.{x,y,z}, blockIdx.{x,y,z}*	get_group_id({x,y,z})	sycl::nd_item::get_group({x,y,z})
Block-dim	blockDim.{x,y,z}	hipBlockDim.{x,y,z}, blockDim.{x,y,z}*	get_local_size({x,y,z})	sycl::nd_item::get_local_range({x,y,z})
Grid-dim	gridDim.{x,y,z}	hipGridDim.{x,y,z}, gridDim.{x,y.z}*	get_num_group({0,1,2})	sycl::nd_item::get_num_group({0,1,2})
Device Kernel	__global__	__global__	__kernel__	C++ lambda, sycl::kernel
Device Function	__device__	__device__	N/A. Implied in device compilation	N/A. Implied in device compilation
Host Function	__host__ (default)	__host__ (default)	N/A. Implied in host compilation	N/A. Implied in host compilation
Host + Device Function	__host__ __device__	__host__ __device__	N/A	N/A
Kernel Launch	<<< >>>	hipLaunchKernel, <<< >>>*	clEnqueueNDRangeKernel	sycl::queue::submit()
Global Memory	__global__	__global__	__global	__global
Group Memory	__shared__	__shared__	__local	__local
Private Memory	(default)	(default)	__private	__private
Constant	__constant__	__constant__	__constant	__constant
Thread Synchronisation	__syncthreads	__syncthreads	barrier(CLK_LOCAL_MEMFENCE)	sycl::queue::wait()
Precise Math	cos(f)	cos(f)	cos(f)	cos(f)
Fast Math	__cos(f)	__cos(f)	native_cos(f)	native_cos(f)

* Since ROCm 3.5

Code Convertors

CUDA to HIP

Hipify-perl

- Easiest to use
- Very simple string replacement technique
- May require manual post-processing

Hipify-clang

- More robust translation of the code
- Generates warnings and assistance for additional analysis
- High quality translation

gpuFORT

- Conversion tool to translate directive-based code to direct kernel

CUDA to SYCL

c2s

- Available in oneAPI Base Toolkit
- dpct is an alias for it

```
$ c2s --help
USAGE: c2s [options] [<source0> ... <sourceN>]
```

SYCLomatic

- It's the name of the above project on GitHub: https://github.com/oneapi-src/SYCLomatic

Library 'Cheat Sheet'

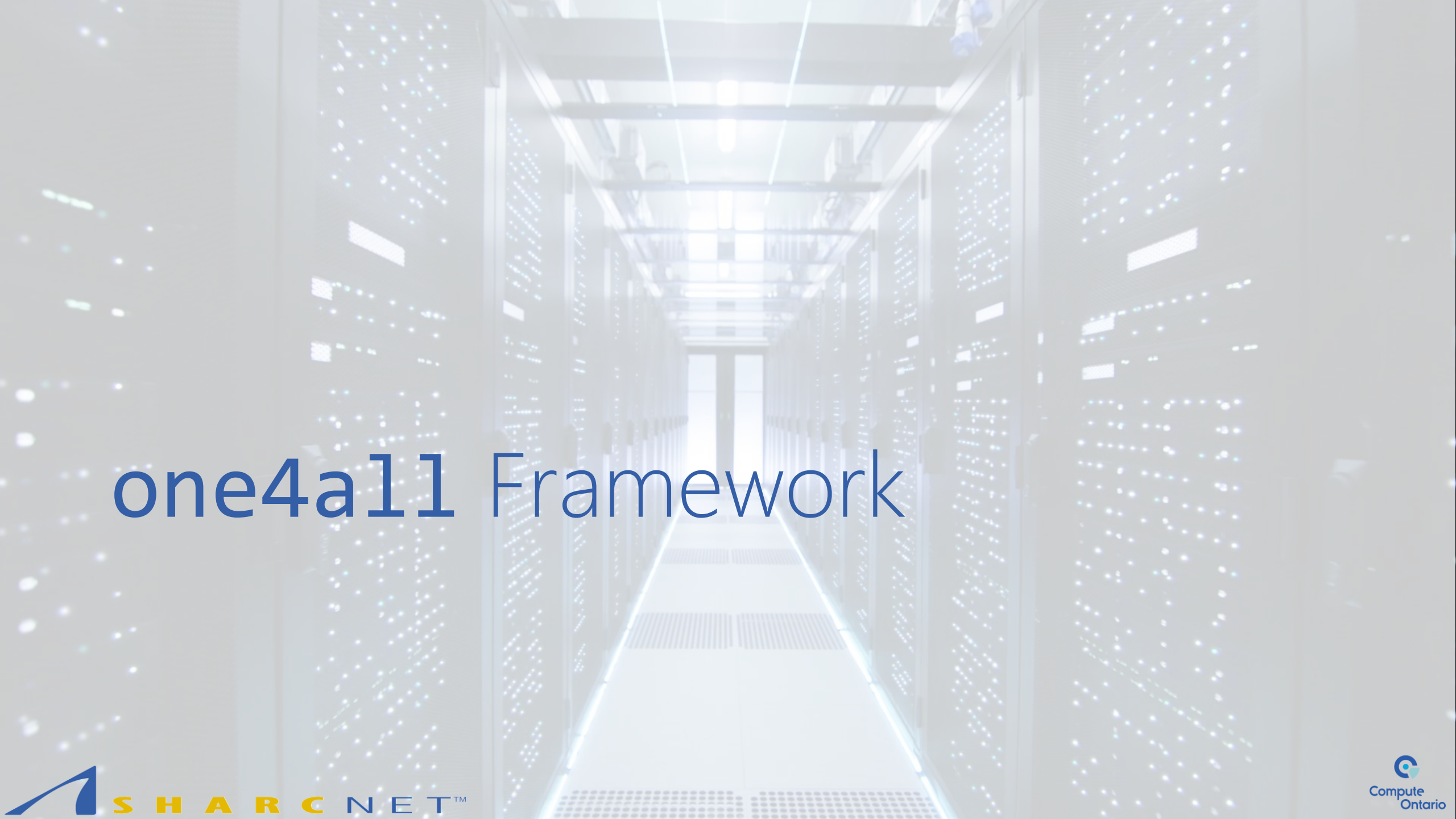
CUDA Library	ROCm Library	oneAPI Library	Description
cuBLAS	rocBLAS	oneMKL	Basic Linear Algebra Subroutines
cuFFT	rocFFT	oneMKL	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	oneMKL	Sparse BLAS + SPMV
cuSolver	rocSOLVER	oneMKL	Lapack library
AMG-X	rocALUTION	oneMKL	Sparse iterative solvers and preconditioners with Geometric and Algebraic MultiGrid
Thrust	rocThrust	oneDPL	C++ parallel algorithms library
CUB	rocPRIM	oneTBB	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	oneDNN	Deep learning Solver Library
cuRAND	rocRAND	oneMKL	Random Number Generator Library
EIGEN	EIGEN – HIP port	oneMKL	C++ template library for linear algebra: matrices, vectors, numerical solvers
NCCL	RCCL	oneCCL	Communications Primitives Library based on the MPI equivalents

CMake Support

Support is already included in CMake

- CMake 3.18 is the recommended minimum

Both provide CMake config files for downstream applications



one4all Framework

one4all

https://github.com/arminms/one4all

Features

- Support four target APIs
  - CUDA
  - oneAPI
  - ROCm
  - STL Parallel Algorithms
- All the configurations are automatically done by CMake
- Support unit testing with Catch2
- Support Google Benchmark
- Two (kernel and Thrust/oneDPL) sample algorithms are already included

Algorithm Launching Kernel – `generate_table()`

[-10,-5]	[-5,-1]	[-1,0]	[0,1]	[1,5]	[5,10]	[10,15]	[15,20]
-5.98057	-1.76911	-0.64359	0.69031	3.92408	6.54551	11.57777	17.53321
-5.66571	-3.28884	-0.34985	0.78536	4.03271	6.24854	10.63636	15.21093
-7.61223	-4.61874	-0.51897	0.50821	3.83865	7.67846	13.92863	17.38602
-6.18005	-1.65404	-0.57231	0.65792	3.94176	7.89860	10.99230	17.92224
-7.93094	-4.61098	-0.29945	0.69920	1.41541	8.05536	10.93716	15.03498
-7.46060	-4.93828	-0.84652	0.88615	4.62336	6.17031	10.71456	19.19021
-6.68157	-4.95061	-0.01087	0.87937	1.02586	7.37018	14.62476	18.53668
-8.80097	-2.85922	-0.75668	0.00395	4.94050	6.33347	14.23037	19.65340
-5.33858	-4.62868	-0.66382	0.29509	2.41884	5.76090	12.93356	15.86466
…	…	…	…	…	…	…	…

Algorithm Launching Kernel – `generate_table()`

Doesn't rely on `cuRAND/rocRAND/oneMKL`

Based on PCG family of random number generators

- https://pcg-random.org
- Compatible with `C++11` random library interface
- Supports *block-splitting* for parallel implementation (but not *leap-frog*)

The uniform distribution code is based on `trng4` library

- https://github.com/rabauke/trng4
- Faster than `C++11` version
- Supports both host and device

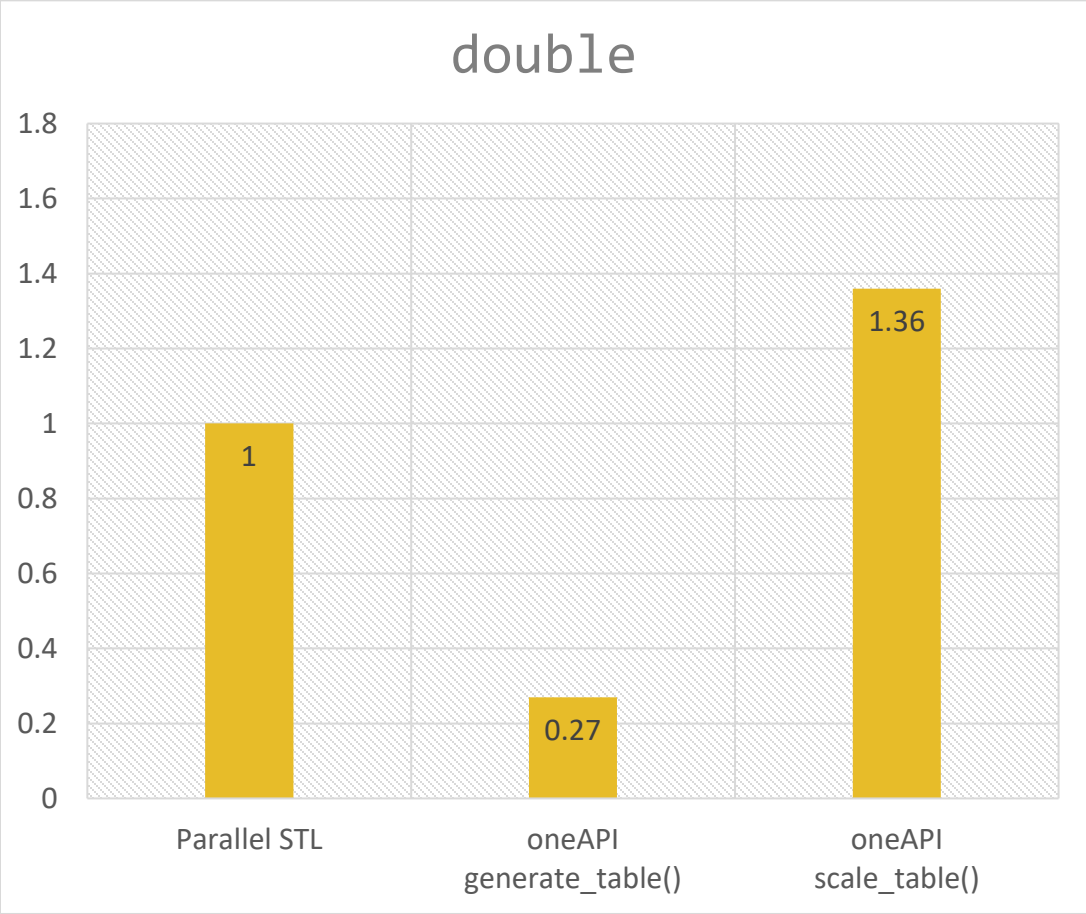
Using the same seed you get the exact same random numbers on all platforms

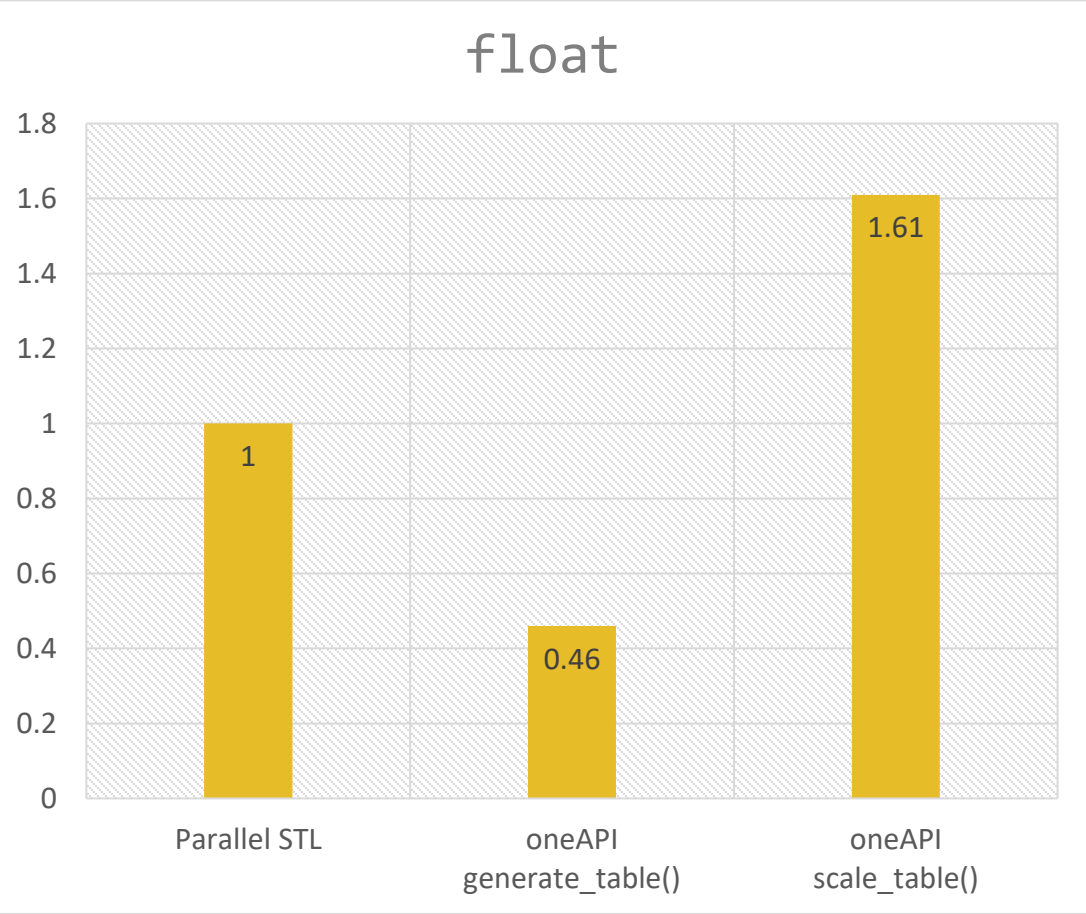
Algorithm using Thrust/oneDPL – `scale_table()`

[-1,1]	[-1,1]	[-1,1]	[-1,1]	[-1,1]	[-1,1]	[-1,1]	[-1,1]
0.60777	0.61544	-0.28718	0.38062	0.46204	-0.38179	-0.36889	0.01328
0.73372	-0.14442	0.30029	0.57072	0.51635	-0.50059	-0.74545	-0.91563
-0.04489	-0.80937	-0.03794	0.01641	0.41933	0.07139	0.57145	-0.04559
0.52798	0.67298	-0.14462	0.31585	0.47088	0.15944	-0.60308	0.16890
-0.17238	-0.80549	0.40109	0.39841	-0.79230	0.22214	-0.62513	-0.98601
0.01576	-0.96914	-0.69304	0.77230	0.81168	-0.53188	-0.71418	0.67608
0.32737	-0.97531	0.97826	0.75874	-0.98707	-0.05193	0.84990	0.41467
-0.52039	0.07039	-0.51336	-0.99211	0.97025	-0.46661	0.69215	0.86136
0.86457	-0.81434	-0.32764	-0.40982	-0.29058	-0.69564	0.17342	-0.65414
…	…	…	…	…	…	…	…

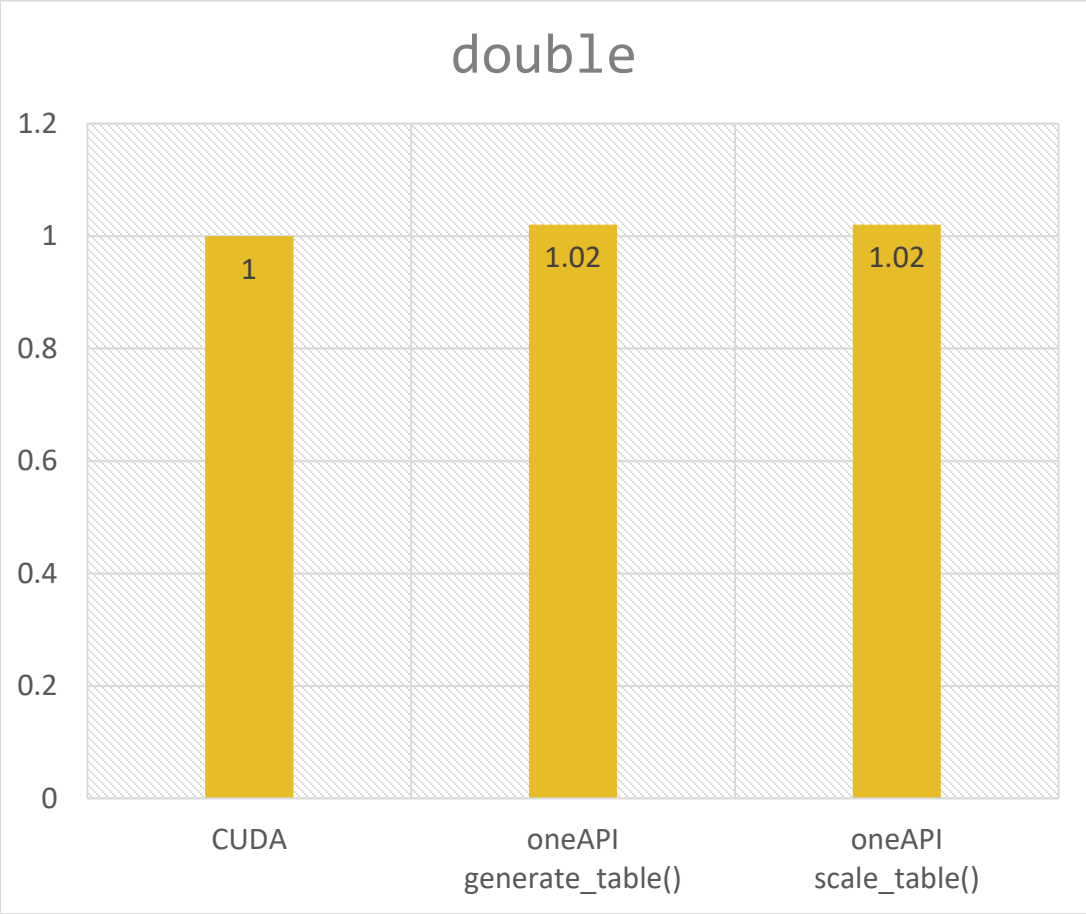
Benchmark Results

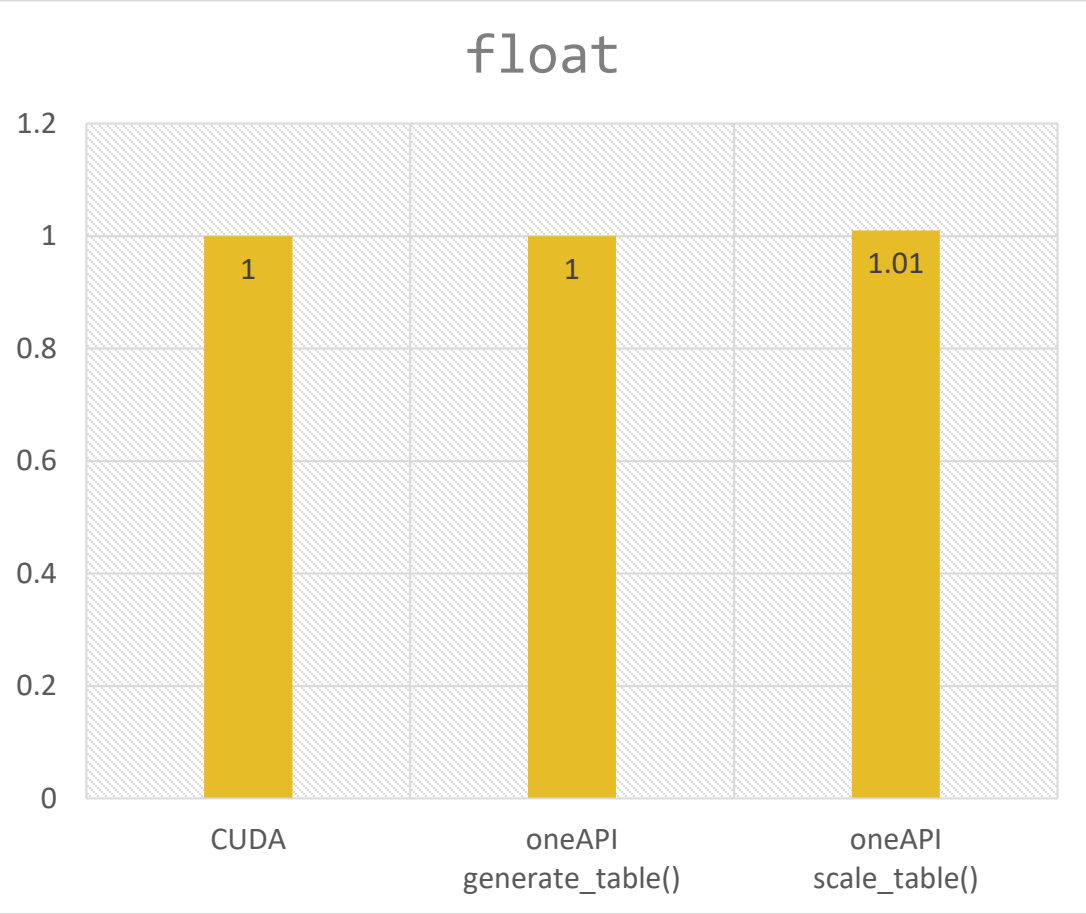
Parallel STL vs. oneAPI (2x AMD EPYC 7543 128 cores)



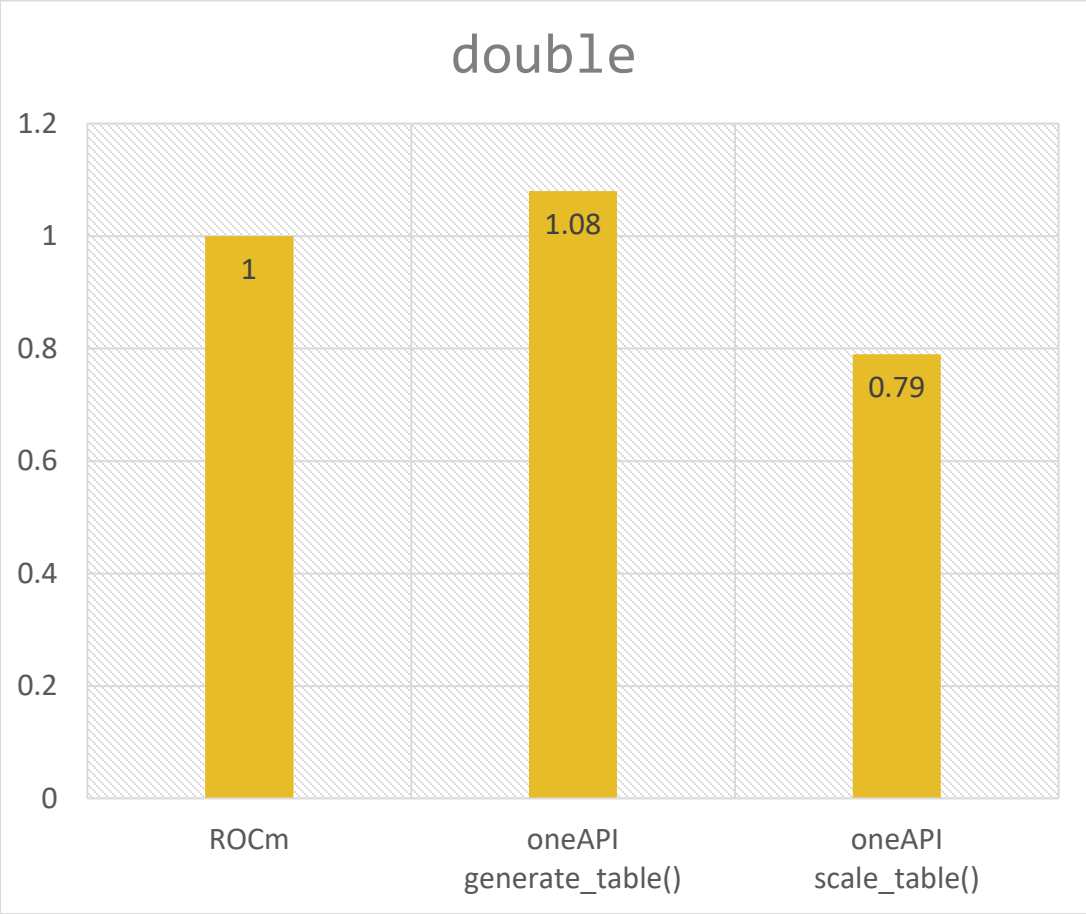


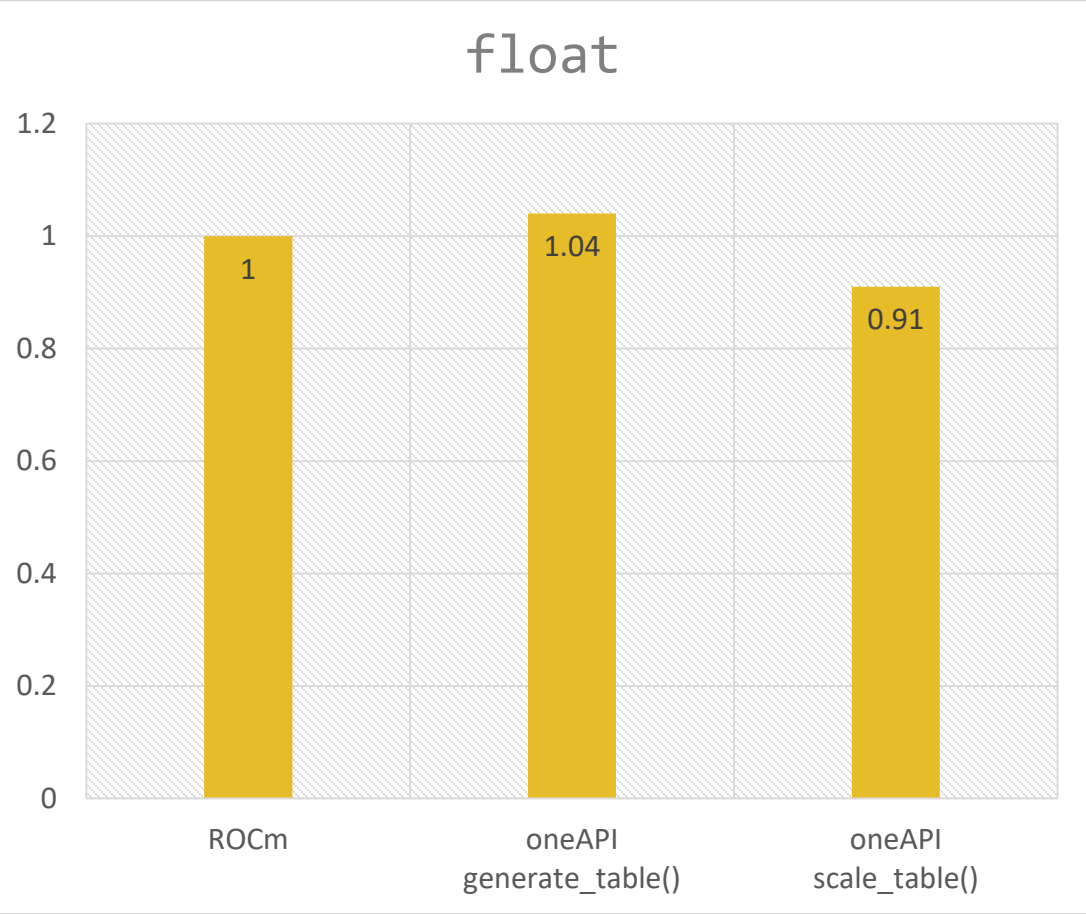
CUDA vs. oneAPI (NVIDIA A100-SXM4-40GB)



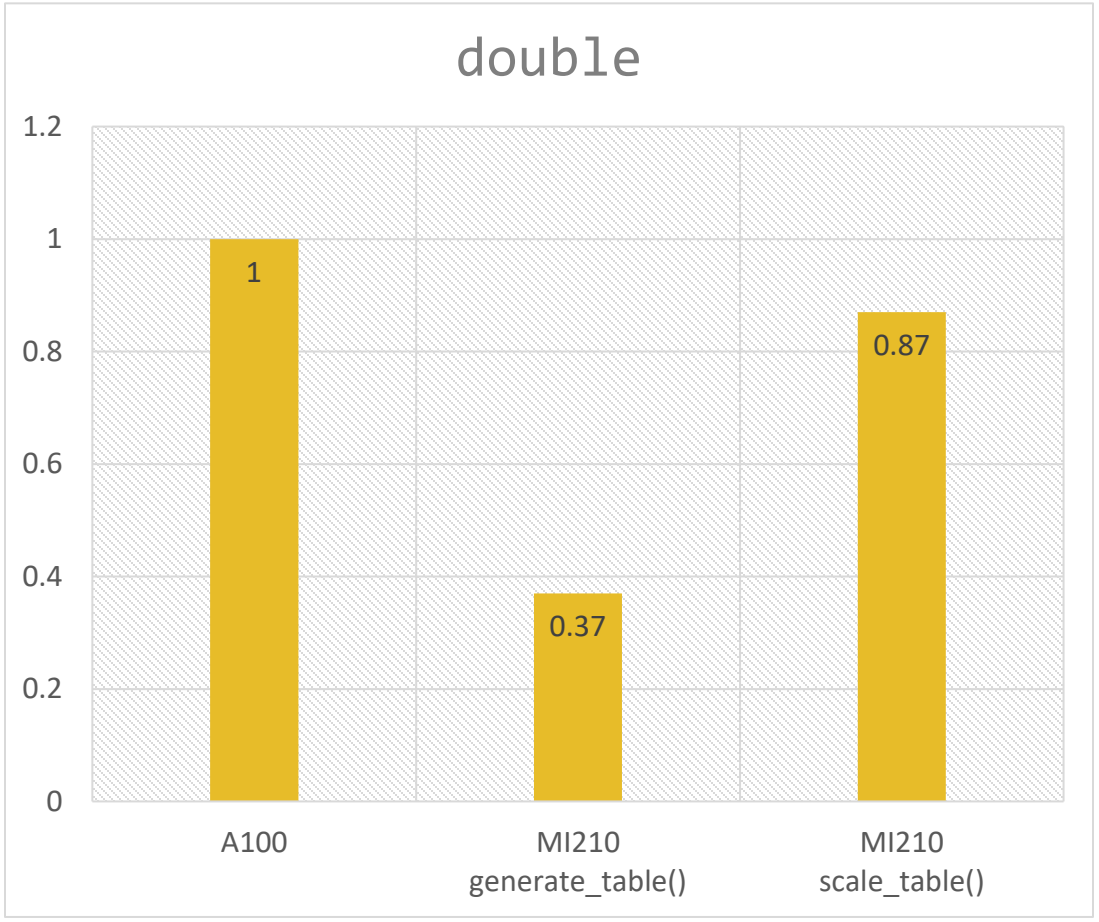


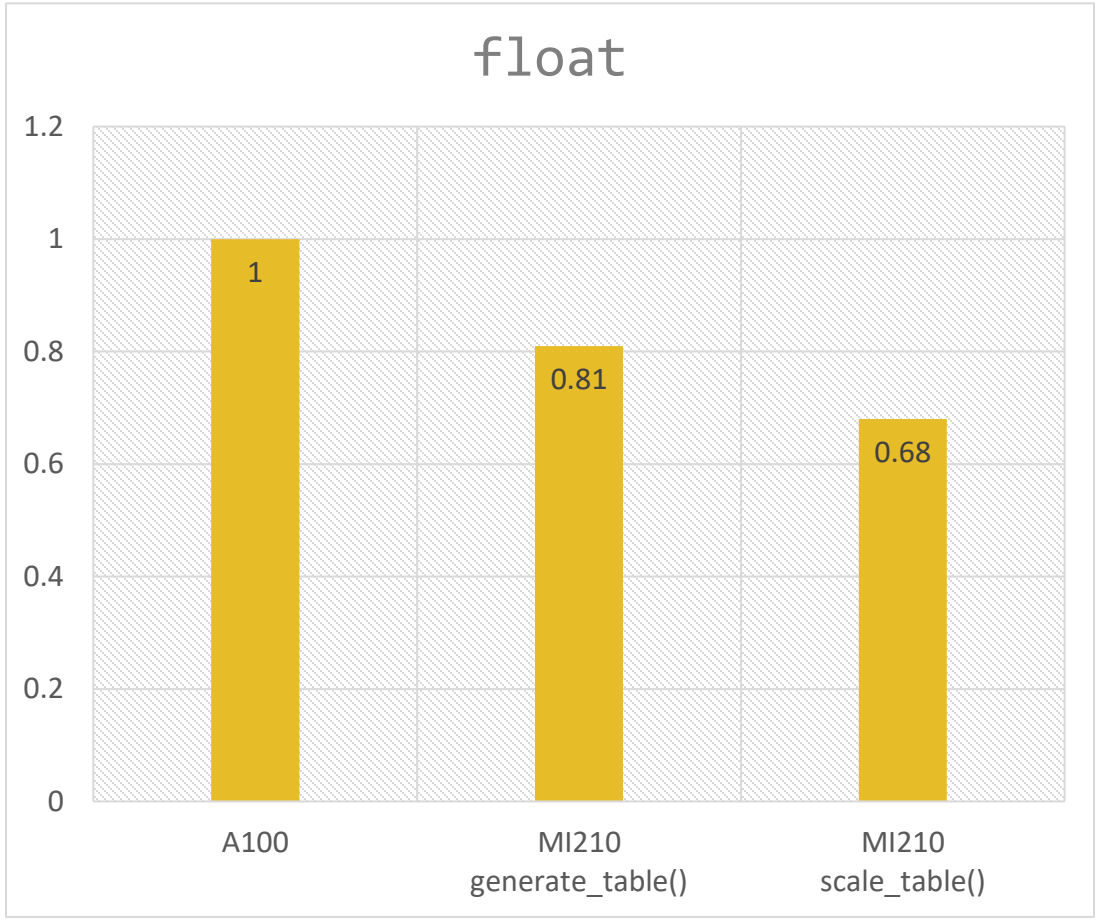
ROCm vs. oneAPI (AMD Instinct MI210)





NVIDIA A100 (SM=108) vs. AMD MI210 (SM=104)





Live Session

https://github.com/arminms/one4all

SHARCNET™

Compute Ontario