Targeting NVIDIA Devices with Intel® oneAPI and SYCL

Intel® oneAPI 사례 발표 및 실습 핸즈온

정진우

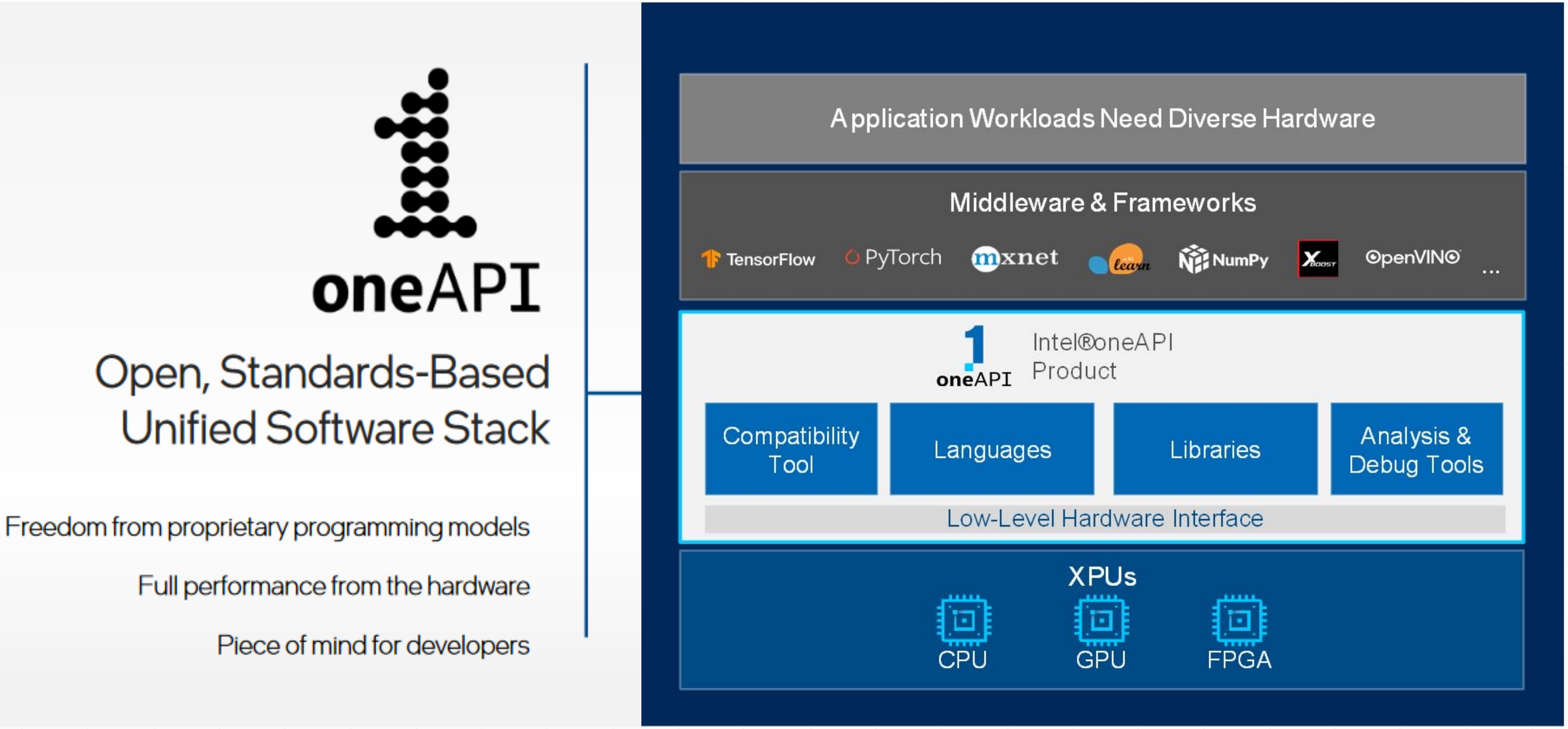
MOASYS

2023. 08. 22.

Content

- Intel® oneAPI and Heterogenous Programming Models
- NVIDIA Plugin for Intel® oneAPI Compilers
- Performance Comparison between Native CUDA and SYCL on NVIDIA GPUs
- Targeting NVIDIA GPUs with Intel® oneMKL Interfaces
- Targeting NVIDIA GPUs with Intel® oneDNN
- Conclusion

Unleash Cross-Architecture Performance with Intel® oneAPI



oneAPI

Open, Standards-Based Unified Software Stack

Freedom from proprietary programming models

Full performance from the hardware

Piece of mind for developers

Application Workloads Need Diverse Hardware

Middleware & Frameworks

Intel®oneAPI Product

Compatibility Tool

Languages

Libraries

Analysis & Debug Tools

Low-Level Hardware Interface

XPUs

CPU

GPU

FPGA

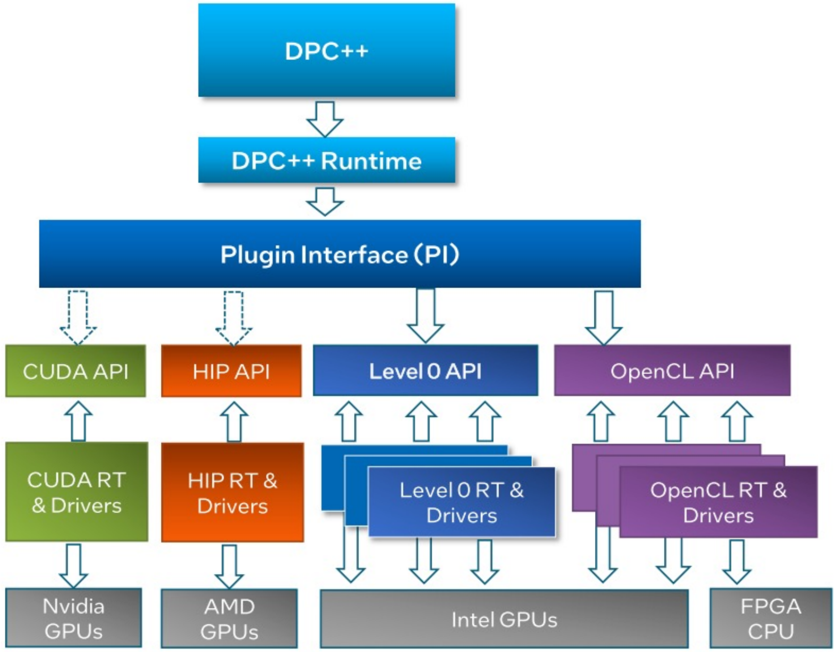
- Support diverse classes of accelerators (XPUs) such as CPUs, GPUs, and FPGA
- Continuously evolve the specifications for high-performance computing and machine learning

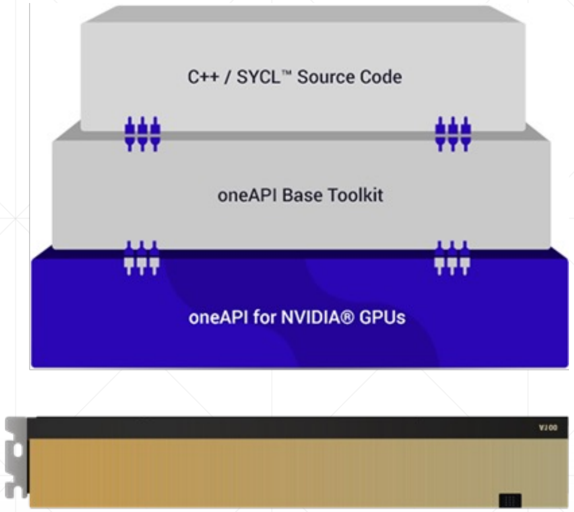
Comparison of Heterogeneous Programming Models

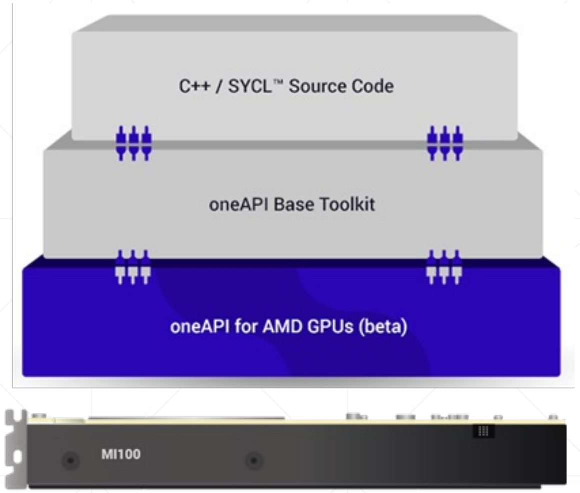
	CUDA	HIP	OpenACC	OpenMP	DPC++/SYCL
Languages	C/C++/Fortran	C++/Fortran	C/C++/Fortran	C/C++/Fortran	C++
Abstraction	Low	Low	High	High	Medium
Coding	-	-	Directive-based	Directive-based	C++ lambda
Parallelism	SIMT	SIMT	Fork-join SIMD	Fork-join SIMD	OpenCL
Offload	GPU (NVIDIA)	GPU (NVIDIA/AMD)	GPU (NVIDIA)	CPU/GPU (NVIDIA/AMD/Intel)	CPU/GPU/FPGA (NVIDIA/AMD/Intel)
Compiler	Proprietary	LLVM	PGI/CCE/GCC	PGI/CCE/GCC/LLVM/XL/Intel	LLVM
License	Proprietary	Open-source	Open-source	Open-source	Open-source

- Classic Intel compilers (icc/icpc/ifort)
  - Support OpenMP 4.0/4.5 standards
  - Do not support OpenMP offload to GPU
- New Intel compilers based on LLVM (icx/icpx/ifx)
  - Support OpenMP 5.0/5.1/5.2 standards
  - Support OpenMP offload to GPU
  - Support SYCL offload to Intel/NVIDIA/AMD devices

oneAPI Compiler Plugin for NVIDIA and AMD GPUs







- Support for Intel devices via Intel® oneAPI Base Toolkits
  - OpenCL API: for Intel CPU, GPU (Gen9, 11, Xe) and FPGA (Stratix, Aria)
  - Level Zero API : low level offload API for Intel® GPUs
- Support for 3rd party devices via binary plugin for the Intel® oneAPI 2023 compiler from Codeplay
  - https://developer.codeplay.com/products/oneapi
  - NVIDIA GPU: validated with A100-PCIe-40GB (sm_80)
  - AMD GPU (beta): validated with AMD Radeon Pro W6800 (gfx1030)

NVIDIA Plugin for oneAPI

- Plugin installation

```
$ sh oneapi-for-nvidia-gpus-2023.2.1-cuda-12.0-linux.sh --install-dir /path/to/intel/oneapi
```

- DPCPP backend enumeration

```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.16.6.0.22_223734]
[opencl:cpu:1] Intel(R) OpenCL, AMD EPYC 7543 32-Core Processor                3.0 [2023.16.6.0.22_223734]
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVIDIA A100-SXM4-80GB 8.8 [CUDA 11.6]
```

- Fat binaries for multiple targets (AMD, NVIDIA, INTEL)

```
$ icpx                                                              \
     -fsycl                                                         \
     -fsycl-targets=amdgcn-amd-amdhsa,nvptx64-nvidia-cuda,spir64    \
     -Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=gfx1030 \
     -Xsycl-target-backend=nvptx64-nvidia-cuda --offload-arch=sm_80 \
     -o sycl-app sycl-app.cpp
```

- Backend selection at runtime

```
$ ONEAPI_DEVICE_SELECTOR=cuda:0 SYCL_PI_TRACE=1 ./sycl-app
```

Benchmarking Representative Algorithms in Computation Sciences

- Microbenchmarks
  - BabelStream: GPU memory bandwidth (https://github.com/UoB-HPC/BabelStream)
- Bioinformatics
  - SneakySnake: universal genome pre-alignment filter (https://github.com/CMU-SAFARI/SneakySnake)
- Image Processing
  - aobench: ambient occlusion benchmark (https://code.google.com/archive/p/aobench/)
- Finance
  - black-scholes: Black-Scholes simulation (https://github.com/cavazos-lab/FinanceBench)
- Machine learning
  - kmeans: find cluster of data (http://lava.cs.virginia.edu/Rodinia/download_links.htm)
- Physics:
  - su3: quantum chromodynamics on SU(3) lattice (https://gitlab.com/NERSC/nersc-proxies/su3_bench)

BabelStream’s Triad Kernel: a[i] = b[i] + scalar * c[i]

OpenMP

```
template <class T>
void OMPStream<T>::triad()
{
  const T scalar = startScalar;
#ifdef OMP_TARGET_GPU
  int array_size = this->array_size;
  T *a = this->a;
  T *b = this->b;
  T *c = this->c;
  #pragma omp target teams distribute parallel for simd
#else
  #pragma omp parallel for
#endif
  for (int i = 0; i < array_size; i++)
  {
    a[i] = b[i] + scalar * c[i];
  }
}
```

OpenCL

```
 kernel void triad(
    global TYPE * restrict a,
    global const TYPE * restrict b,
    global const TYPE * restrict c)
  {
    const size_t i = get_global_id(0);
    a[i] = b[i] + scalar * c[i];
  }


template <class T>
void OCLStream<T>::triad()
{
  (*triad_kernel)(
    cl::EnqueueArgs(queue, cl::NDRange(array_size)), d_a, d_b, d_c
  );
  queue.finish();
}
```

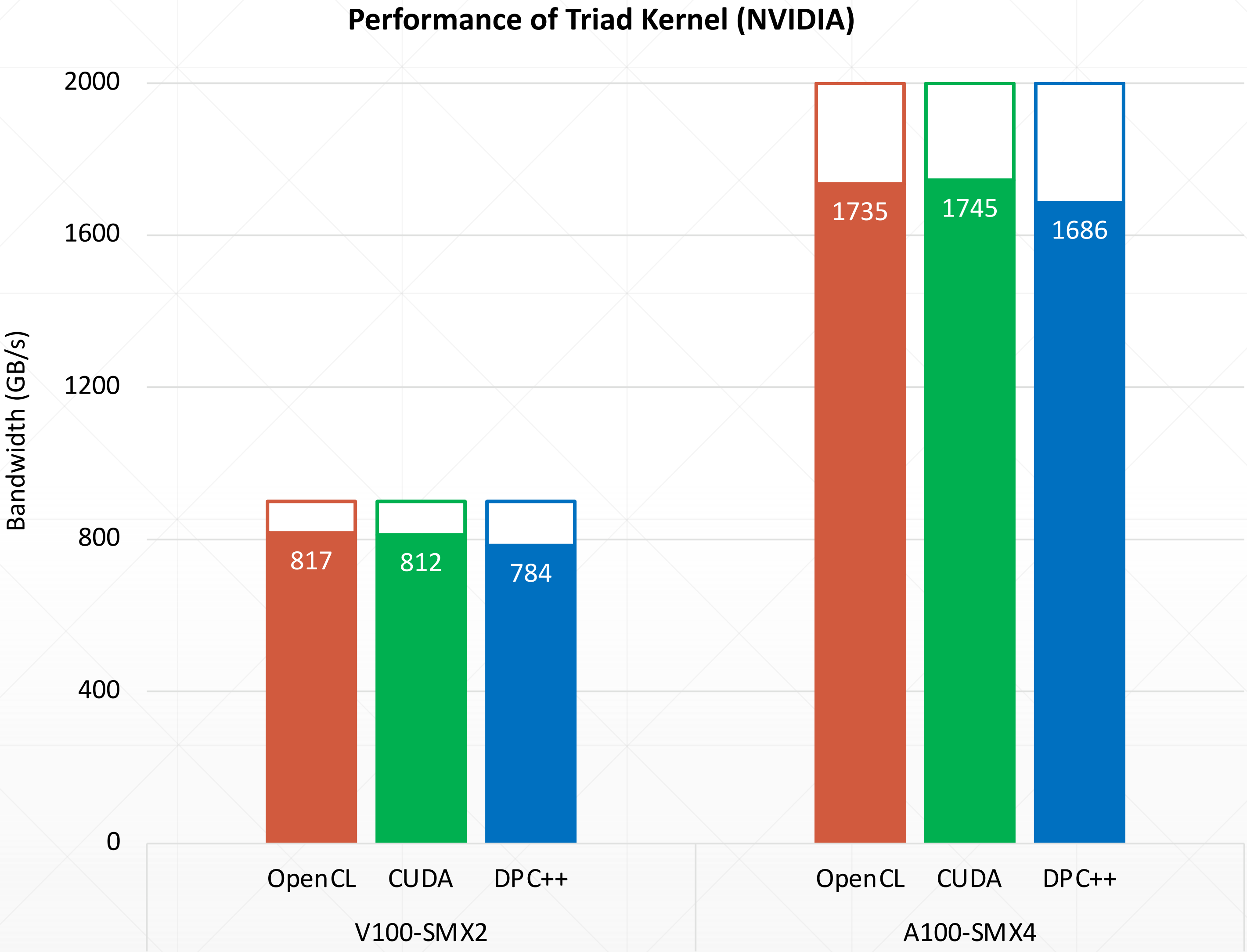
NVIDIA CUDA

```
template <typename T>
__global__ void triad_kernel(T * a, const T * b, const T * c)
{
   const T scalar = startScalar;

   const int i = blockDim.x * blockIdx.x + threadIdx.x;
   a[i] = b[i] + scalar * c[i];
}

template <class T>
void CUDAStream<T>::triad()
{
  triad_kernel<<<array_size/TBSIZE, TBSIZE>>>(d_a, d_b, d_c);
  check_error();
  cudaDeviceSynchronize();
  check_error();
}
```

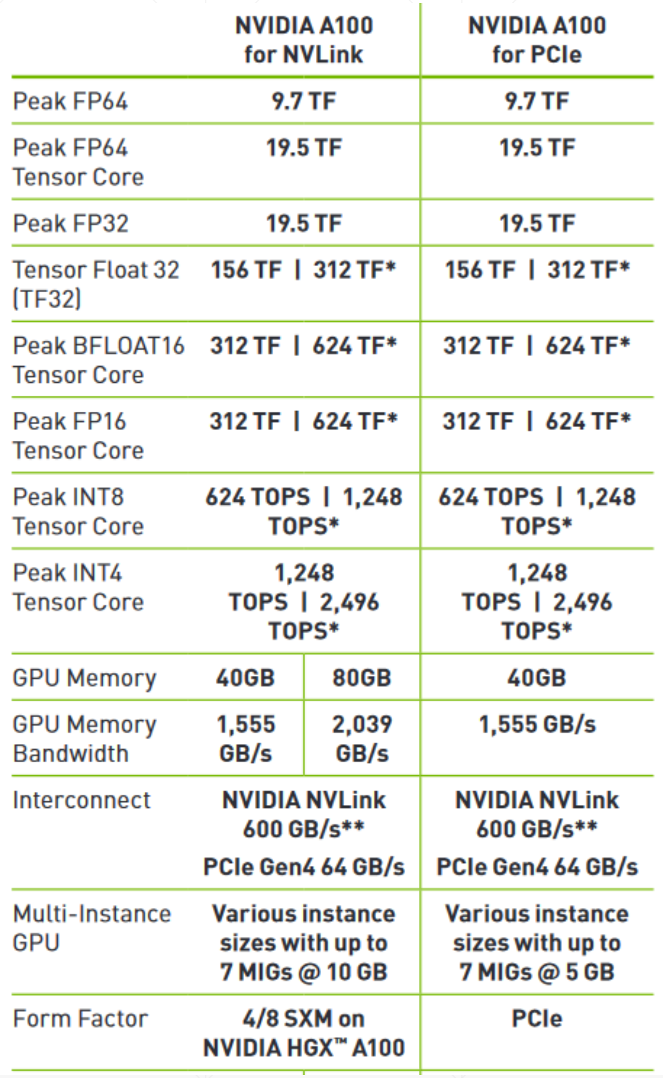
oneAPI

```
template <class T>
void SYCLStream<T>::triad()
{
  const T scalar = startScalar;

  queue->submit([&](handler &cgh)
  {
    sycl::accessor ka {d_a, cgh, sycl::write_only};
    sycl::accessor kb {d_b, cgh, sycl::read_only};
    sycl::accessor kc {d_c, cgh, sycl::read_only};
    cgh.parallel_for<triad_kernel>(range<1>{array_size}, [=](id<1> idx)
    {
       ka[idx] = kb[idx] + scalar * kc[idx];
    });
});
queue->wait();
}
```

Memory Bandwidth: V100/A100

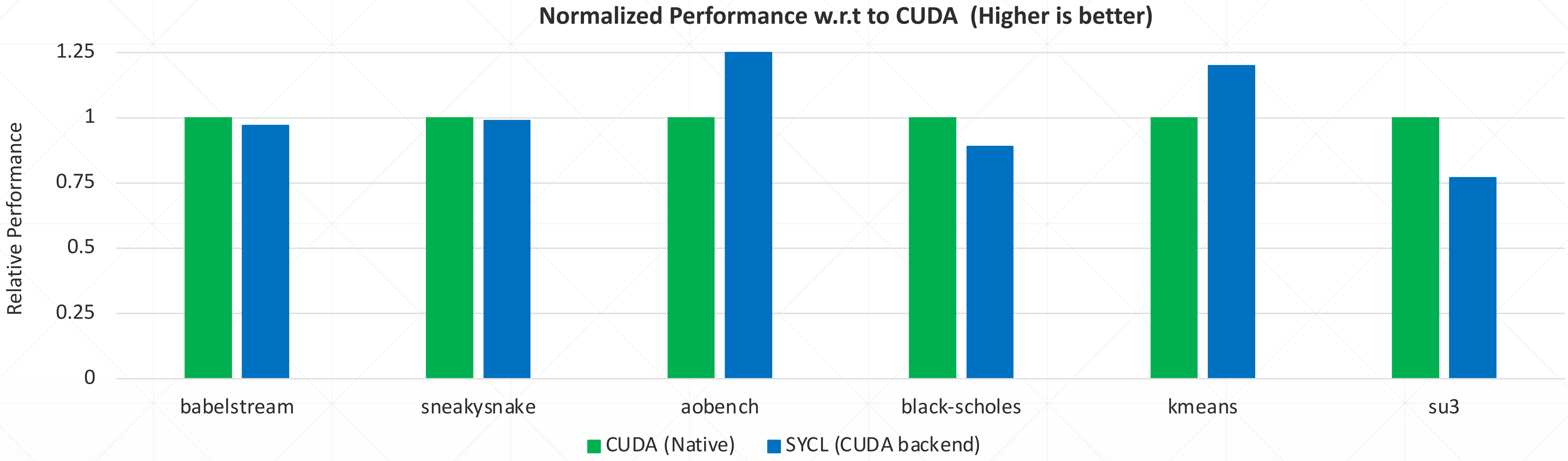


Performance of Triad Kernel (NVIDIA)



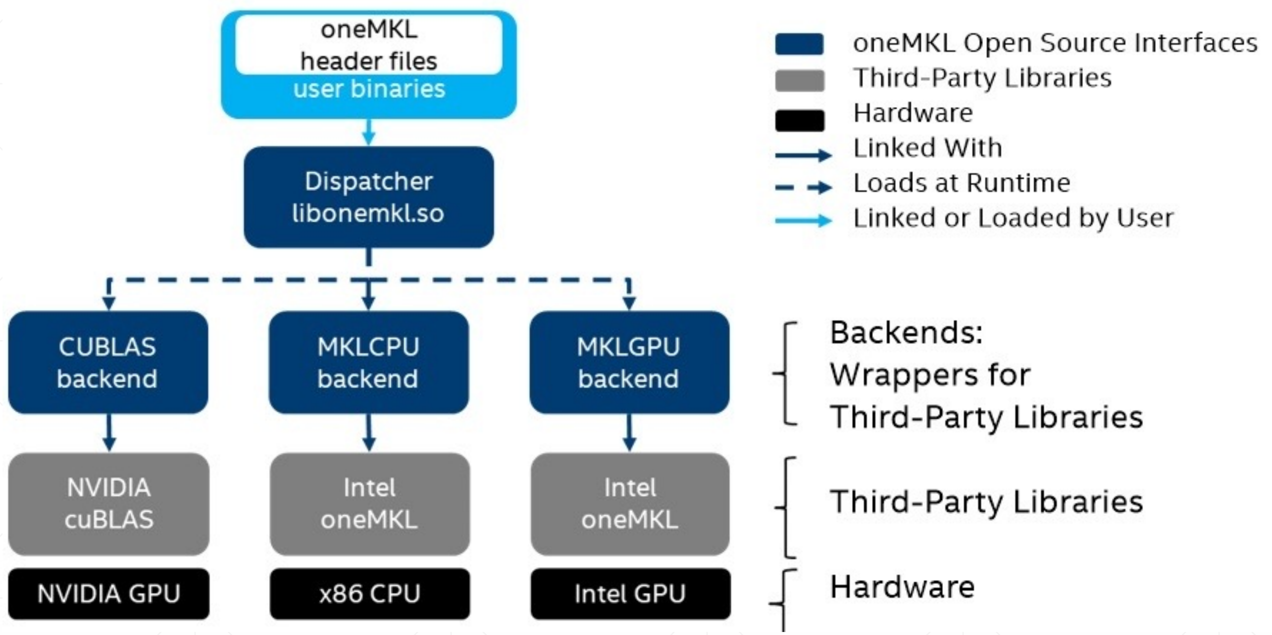
	NVIDIA A100 for NVLink		NVIDIA A100 for PCIe
Peak FP64	9.7 TF		9.7 TF
Peak FP64 Tensor Core	19.5 TF		19.5 TF
Peak FP32	19.5 TF		19.5 TF
Tensor Float 32 (TF32)	156 TF | 312 TF*		156 TF | 312 TF*
Peak BFLOAT16 Tensor Core	312 TF | 624 TF*		312 TF | 624 TF*
Peak FP16 Tensor Core	312 TF | 624 TF*		312 TF | 624 TF*
Peak INT8 Tensor Core	624 TOPS | 1,248 TOPS*		624 TOPS | 1,248 TOPS*
Peak INT4 Tensor Core	1,248 TOPS | 2,496 TOPS*		1,248 TOPS | 2,496 TOPS*
GPU Memory	40GB	80GB	40GB
GPU Memory Bandwidth	1,555 GB/s	2,039 GB/s	1,555 GB/s
Interconnect	NVIDIA NVLink 600 GB/s** PCIe Gen4 64 GB/s		NVIDIA NVLink 600 GB/s** PCIe Gen4 64 GB/s
Multi-Instance GPU	Various instance sizes with up to 7 MIGs @ 10 GB		Various instance sizes with up to 7 MIGs @ 5 GB
Form Factor	4/8 SXM on NVIDIA HGX™ A100		PCIe

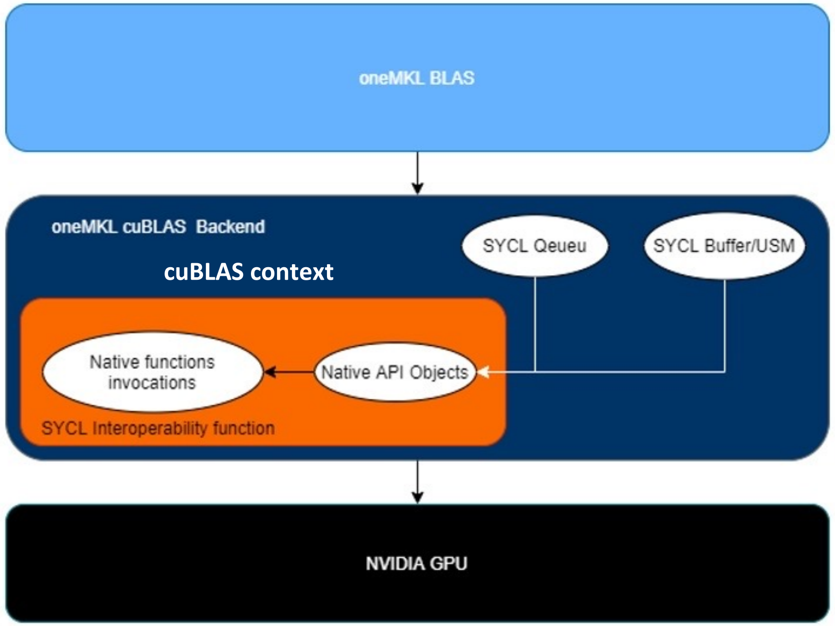
Performance Comparision Between Native CUDA and SYCL



- Migrated SYCL codes (Oak Ridge National Lab, USA)
  - https://github.com/zjin-lcf/HeCBench
- Tested systems:
  - AMD EPYC 7543 32-Core Processor
  - NVIDIA A100-SMX4 80GB
  - Intel oneAPI 2023.2.0 with NVIDIA plugin

oneAPI Math Kernel Library (oneMKL) Interfaces

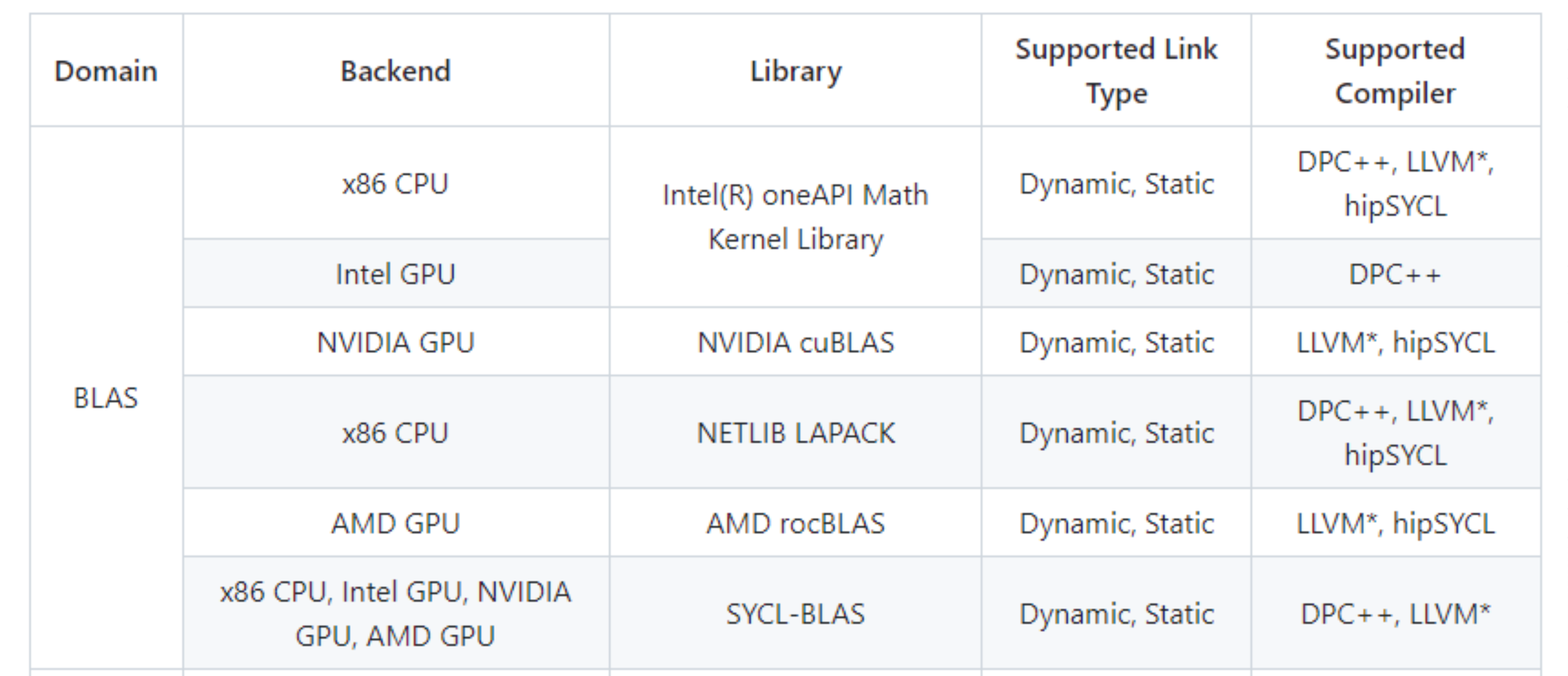




https://techdecoded.intel.io/resources/a-vendor-neutral-path-to-math-acceleration/

- oneMKL interfaces allows mapping oneMKL calls to 3rd party backends:
  - Automatic selection of backends at runtime based on SYCL device selection, i.e cpu_selector_v/gpu_selector_v
    - iGPU device queue -> Intel oneMKL backend selected
    - NVIDIA device queue -> NVIDIA cuBLAS backend selected
    - Generic CPU device queue -> Netlib referece BLAS backend selected
- 3rd party backends: BLAS, LAPACK, RNG, DFT
  - NVIDIA (CUDA SDK) / AMD (ROCm)

oneMKL Interfaces: Support Domains

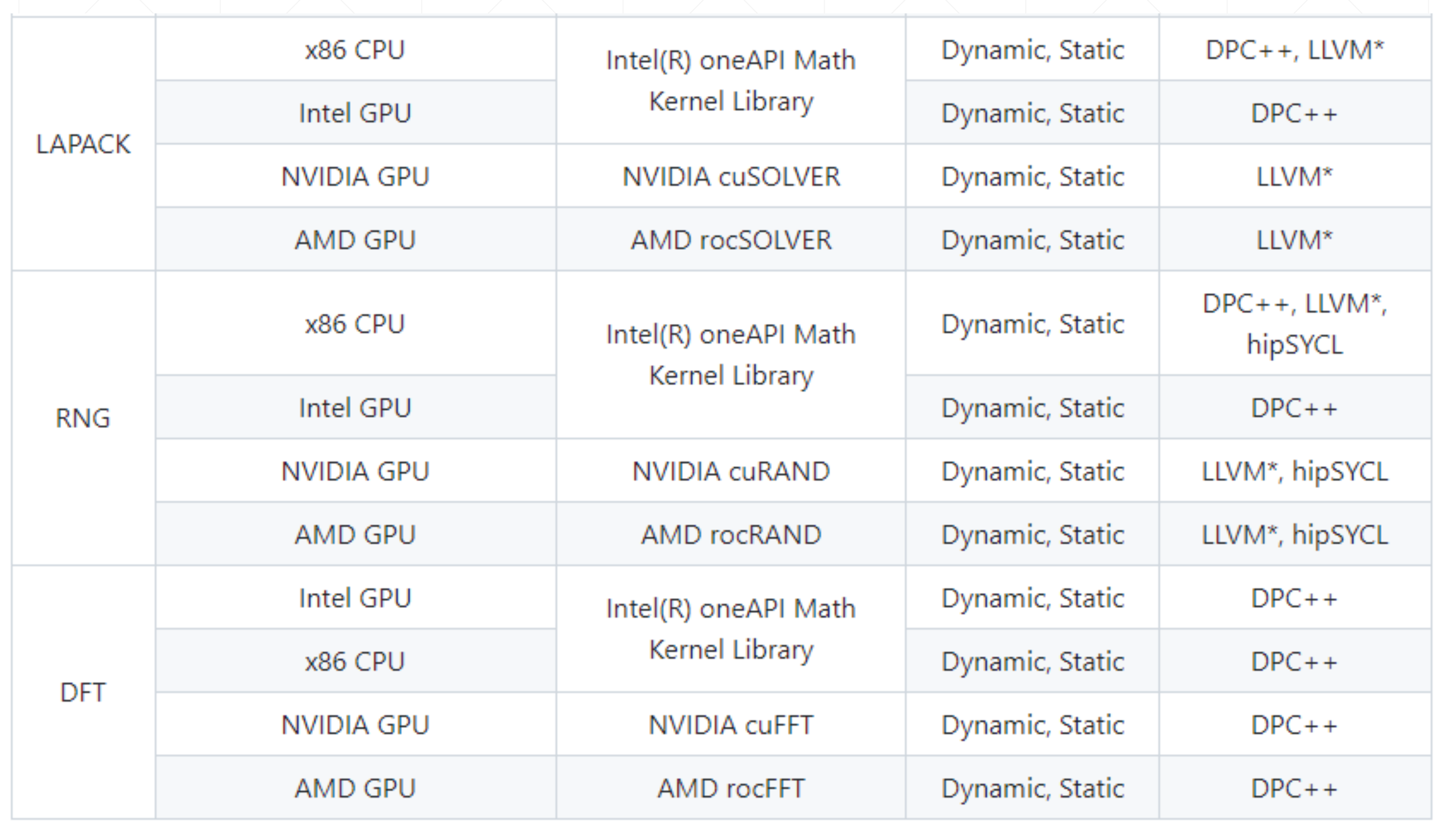


Domain	Backend	Library	Supported Link Type	Supported Compiler
BLAS	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*, hipSYCL
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuBLAS	Dynamic, Static	LLVM*, hipSYCL
	x86 CPU	NETLIB LAPACK	Dynamic, Static	DPC++, LLVM*, hipSYCL
	AMD GPU	AMD rocBLAS	Dynamic, Static	LLVM*, hipSYCL
	x86 CPU, Intel GPU, NVIDIA GPU, AMD GPU	SYCL-BLAS	Dynamic, Static	DPC++, LLVM*

LLVM*: oneAPI compilers + NVIDIA/AMD plugins

https://github.com/oneapi-src/oneMKL

oneMKL Interfaces: Support Domains



LAPACK	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuSOLVER	Dynamic, Static	LLVM*
	AMD GPU	AMD rocSOLVER	Dynamic, Static	LLVM*
RNG	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*, hipSYCL
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuRAND	Dynamic, Static	LLVM*, hipSYCL
	AMD GPU	AMD rocRAND	Dynamic, Static	LLVM*, hipSYCL
DFT	Intel GPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++
	x86 CPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuFFT	Dynamic, Static	DPC++
	AMD GPU	AMD rocFFT	Dynamic, Static	DPC++

LLVM*: oneAPI compilers + NVIDIA/AMD plugins

oneMKL: SGEMM with Unified Shared Memory (USM)

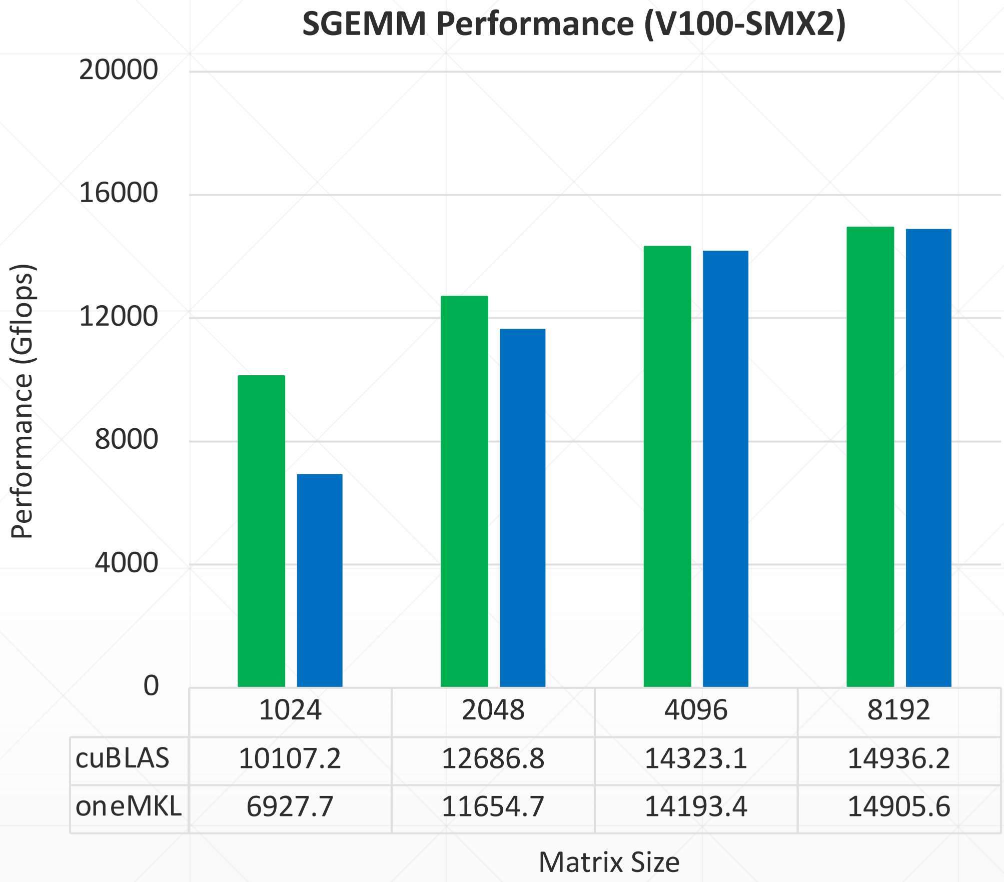
```
// GPU is selected implicitly


// host data
float* A = (float *) aligned_alloc(32, (m * k) * sizeof(float));
float* B = (float *) aligned_alloc(32, (k * n) * sizeof(float));
float* C = (float *) aligned_alloc(32, (m * n) * sizeof(float));

// device data
float *dA, *dB, *dC;
cudaMalloc((void**) &dA, (m * k) * sizeof(float));
cudaMalloc((void**) &dB, (k * n) * sizeof(float));
cudaMalloc((void**) &dC, (m * n) * sizeof(float));

// copy matrix to gpu
cublasSetMatrix(m, k, sizeof(float), A, ldA, dA, ldA);
cublasSetMatrix(k, n, sizeof(float), B, ldB, dB, ldB);
cublasSetMatrix(m, n, sizeof(float), C, ldC, dC, ldC);

// cublas context
cublasStatus_t status;
cublasHandle_t handle;
cublasCreate(&handle);

// cuda events
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);
for (int i=0; i < LOOP; i++) {
    status = cublasSgemm(
        handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k
        &alpha, dA, ldA, dB, ldB, &beta, dC, ldC
    );
}
cudaEventRecord(stop);
// copy data back to host
cublasGetMatrix(m, n, sizeof(float), dC, ldC, C, ldC);
cublasDestroy(handle);
```

NVIDIA CUDA

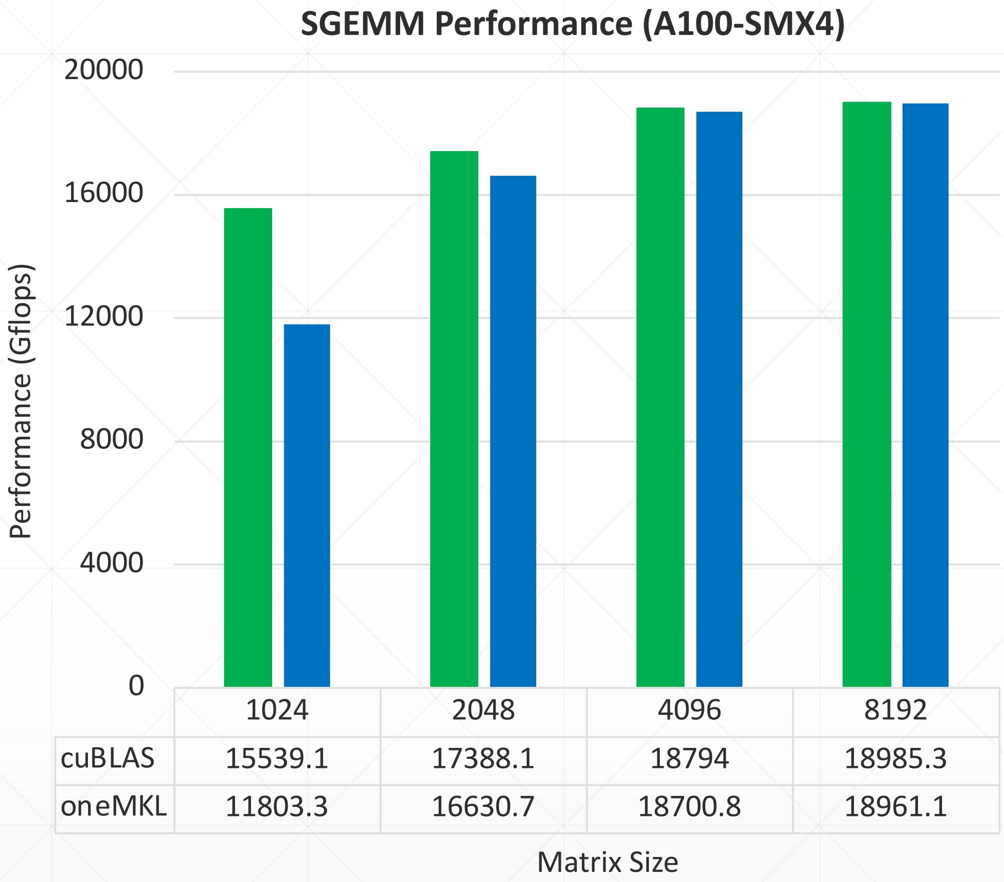
```
// device is selected explicitly
sycl::queue dev_queue(sycl::gpu_selector_v);

// USM is fully supported with BLAS{1,2,3}
float *A_USM = sycl::malloc_shared<float>(m * k, dev_queue);
float *B_USM = sycl::malloc_shared<float>(k * n, dev_queue);
float *C_USM = sycl::malloc_shared<float>(m * n, dev_queue);


// no explicit device pointer allocation




// no explicit data transfer from host to device




// cuBLAS context is wrapped under oneapi::mkl::blas call









auto start = std::chrono::high_resolution_clock::now();
for (int i=0; i < LOOP; i++) {
   oneapi::mkl::blas::column_major::gemm(
          dev_queue, transA, transB, m, n, k,
          alpha, A_USM, ldA, B_USM, ldB, beta, C_USM, ldC
  ).wait();
}
auto end = std::chrono::high_resolution_clock::now();
// no explicit data transfer from device back to host
```

oneAPI

oneMKL: SGEMM Performance on NVIDIA Devices


SGEMM Performance (V100-SMX2)

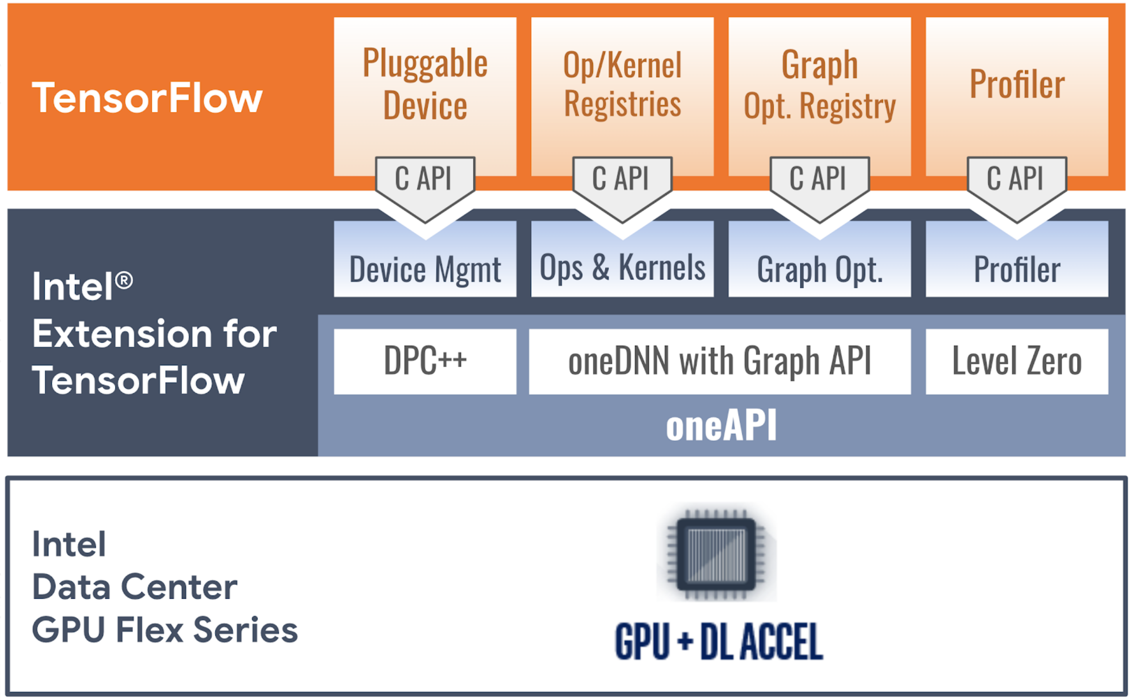
	1024	2048	4096	8192
cuBLAS	10107.2	12686.8	14323.1	14936.2
oneMKL	6927.7	11654.7	14193.4	14905.6


SGEMM Performance (A100-SMX4)

	1024	2048	4096	8192
cuBLAS	15539.1	17388.1	18794	18985.3
oneMKL	11803.3	16630.7	18700.8	18961.1

- oneMKL archives parity performance with native cuBLAS
  - V100-SMX2: R_{max} = 14.9 TFlops, R_{peak} = 15.7 TFlops
  - A100-SMX4: R_{max} = 18.9 TFLops, R_{peak} = 19.5 TFlops

oneAPI Deep Neural Network Library (oneDNN)



TensorFlow

Pluggable Device

Op/Kernel Registries

Graph Opt. Registry

Profiler

C API

C API

C API

C API

Intel® Extension for TensorFlow

Device Mgmt

Ops & Kernels

Graph Opt.

Profiler

DPC++

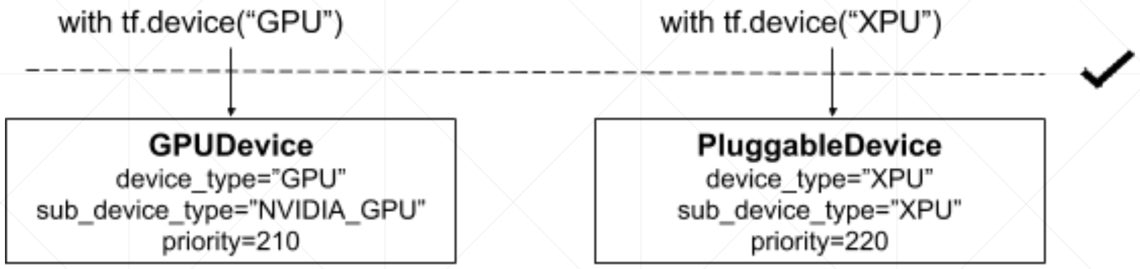
oneDNN with Graph API

Level Zero

oneAPI

Intel Data Center GPU Flex Series

GPU + DL ACCEL



with tf.device("GPU")

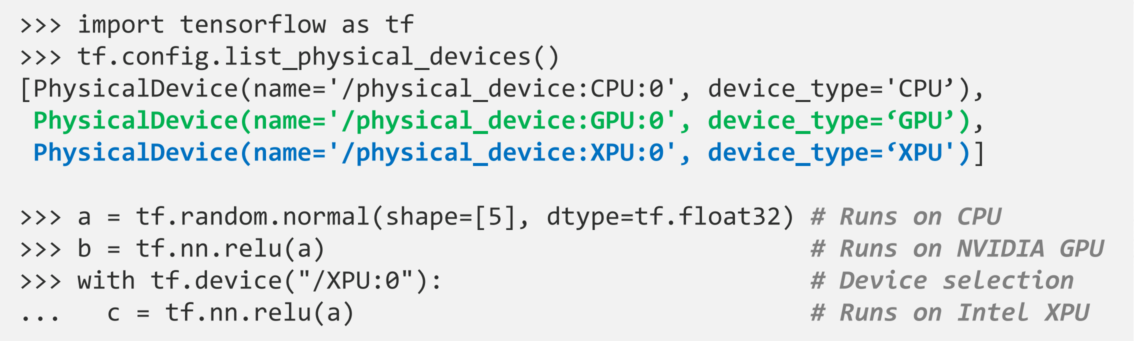
with tf.device("XPU")

✓

GPUDevice
device_type="GPU"
sub_device_type="NVIDIA_GPU"
priority=210

PluggableDevice
device_type="XPU"
sub_device_type="XPU"
priority=220

Scenario 2: Single PluggableDevice registered as a new device type, e.g., "XPU"

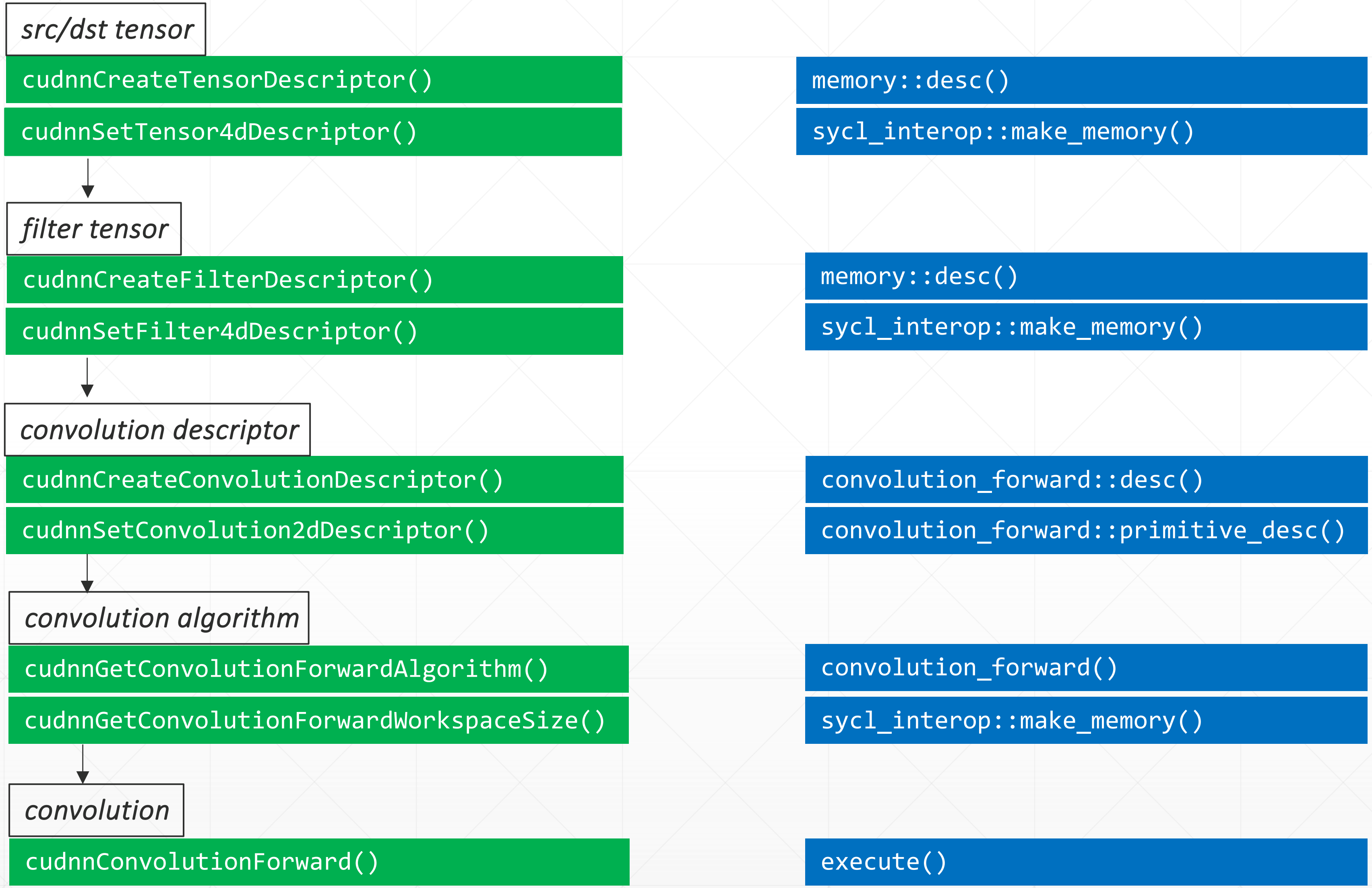


```
>>> import tensorflow as tf
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU’),
 PhysicalDevice(name='/physical_device:GPU:0', device_type=‘GPU’),
 PhysicalDevice(name='/physical_device:XPU:0', device_type=‘XPU')]

>>> a = tf.random.normal(shape=[5], dtype=tf.float32) # Runs on CPU
>>> b = tf.nn.relu(a)                                 # Runs on NVIDIA GPU
>>> with tf.device("/XPU:0"):                         # Device selection
...   c = tf.nn.relu(a)                               # Runs on Intel XPU
```

- Open-source cross-platform performance library of basic building blocks for deep learning
  - Optimized for Intel® XPUs
  - Experimental support for NVIDIA (cuBLAS/cuDNN) and AMD GPU (rocBLAS)
- Intel® Extension for TensorFlow
  - Developers can train and infer TensorFlow models on Intel AI hardware with zero code change

```
$ pip install --upgrade intel-extension-for-tensorflow[gpu]
```

cuDNN vs oneDNN: Edge Detection Comparison



src/dst tensor	
cudnnCreateTensorDescriptor()	memory::desc()
cudnnSetTensor4dDescriptor()	sycl_interop::make_memory()
filter tensor	
cudnnCreateFilterDescriptor()	memory::desc()
cudnnSetFilter4dDescriptor()	sycl_interop::make_memory()
convolution descriptor	
cudnnCreateConvolutionDescriptor()	convolution_forward::desc()
cudnnSetConvolution2dDescriptor()	convolution_forward::primitive_desc()
convolution algorithm	
cudnnGetConvolutionForwardAlgorithm()	convolution_forward()
cudnnGetConvolutionForwardWorkspaceSize()	sycl_interop::make_memory()
convolution	
cudnnConvolutionForward()	execute()

oneDNN: Primitive Operation Benchmark

- oneDNN provides internal benchmark tool
  - Measurement of performance of primitive operations on different hardwares

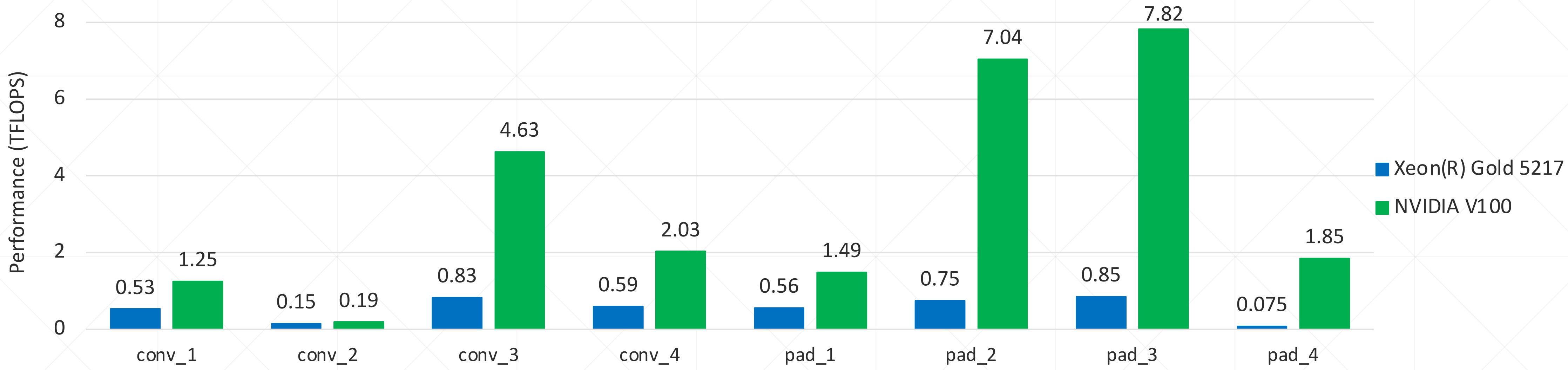
- Benchmark of convolution primitives

```
$ benchdnn                          \
    --conv                          \ # Convolution primitive
    --engine=gpu:0                  \ # Use first GPU device
    --dir=FWB_B                     \ # Forward training with bias
    --cfg=f32                       \ # Single precision (FP32)
    --mode=p                        \ # Performance measurement mode
    --alg=direct                    \ # Direct algorithm
    --batch=inputs/conv/shapes_3d \ # input file
    --perf-template=csv             \ # output in csv format
    > conv_perf.dat
```

FWD_B	dnnl_forward_training w/ bias
FWD_D	dnnl_forward_training w/o bias
FWD_I	dnnl_forward_inference
BWD_D	dnnl_backward_data
BWD_WB	dnnl_backward_weights w/ bias
BWD_W	dnnl_backward_weights w/o bias
BWD_DW	dnnl_backward

- Return status of benchdnn:
  - PASSED: test passed the validation
  - SKIPPED: test was not run and a reason is reported.
  - FAILED: test did not pass the validation with respect to reference result
  - LISTED: test was initialized but primitive descriptor was not created in this case, i.e dry run
  - UNIMPLEMENTED: test corresponds to unimplemented feature and treated as FAIL

oneDNN: Convolution Benchmark

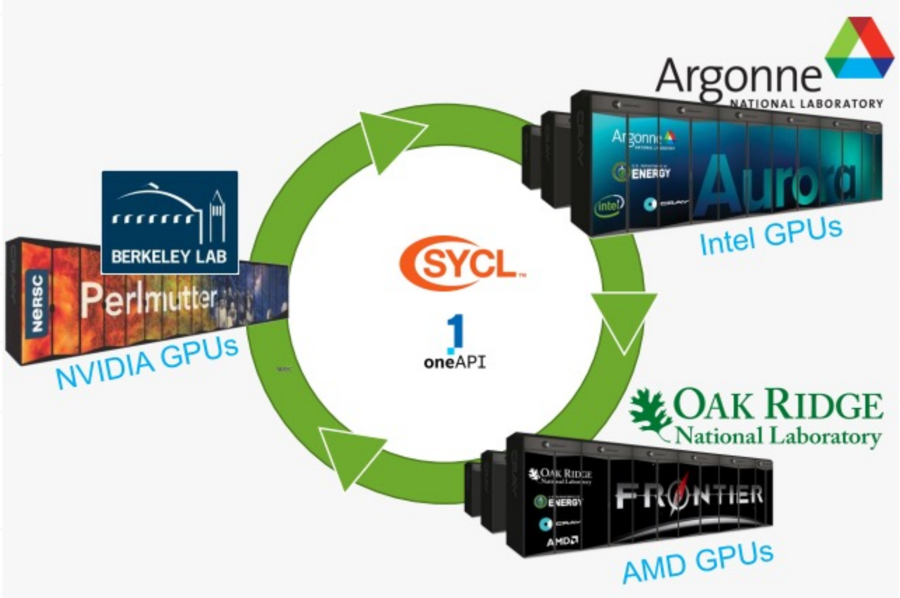


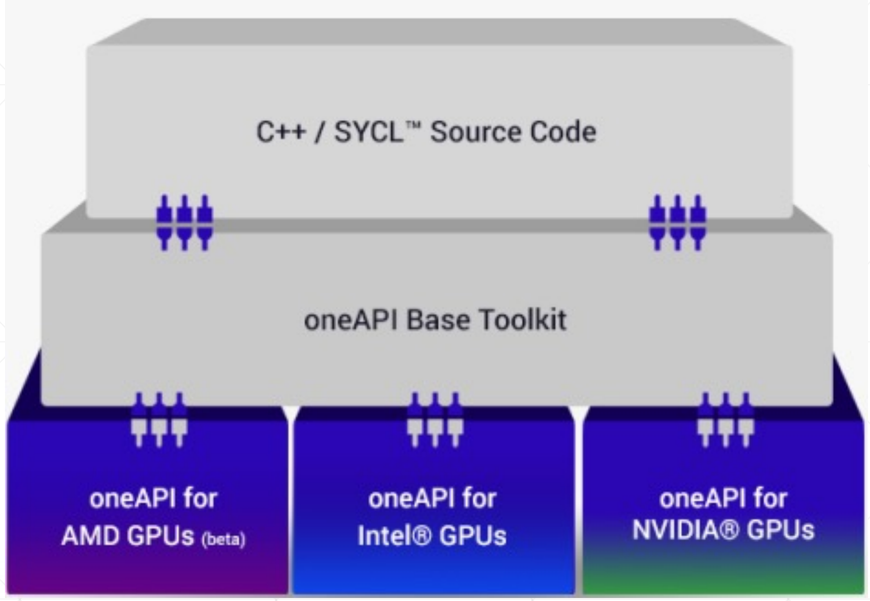
- Problem descriptors:
  - ic, io: input and output channel
  - id, ih, iw: input depth, height and width
  - kd, kh, kw: kernel depth, height and width
  - pd, ph, pw: front, top and left padding
  - n: descriptor name
  - _: optional delimiter between entries for readability.
- Limitations of NVIDIA backend:
  - bf16 is *not* yet implemented
  - https://github.com/oneapi-src/oneDNN/blob/master/src/gpu/nvidia/README.md

```
shapes_3d
```

```
ic16oc16_ih13kh3ph1_iw50kw3pw1_id10kd3pd1_n"3d_conv_pad:1"
ic64oc64_ih10kh3ph1_iw20kw3pw1_id15kd3pd1_n"3d_conv_pad:2"
ic256oc256_ih7kh3ph1_iw9kw3pw1_id11kd3pd1_n"3d_conv_pad:3"
ic256oc256_ih7kh1ph1_iw9kw1pw1_id11kd1pd1_n"3d_conv_pad:4"
ic16oc16_ih13kh3ph0_iw50kw3pw0_id10kd3pd0_n"3d_conv:1"
ic16oc16_ih13kh1ph0_iw50kw1pw0_id10kd1pd0_n"3d_conv:2"
ic256oc256_ih7kh3ph0_iw9kw3pw0_id11kd3pd0_n"3d_conv:3"
ic256oc256_ih7kh1ph0_iw9kw1pw0_id11kd1pd0_n"3d_conv:4“
```

Conclusion





CodePlay Software, *Targeting Multi-Vendor Architectures with oneAPI and SYCL*

- oneAPI offers:
  - Free, and open standard approach to unleash performance on heterogenous platform
  - Performance portability: write once, run everywhere

- NVIDIA’ plugin provides performance parity with native CUDA
  - HPC applications can interop with CUDA performance libraries via oneMKL interfaces
  - Support for NVIDIA devices via oneDNN is under development