oneDNN project overview and developments

Mourad Gouicem
03/14/2024

UXL FOUNDATION
Unified Acceleration

oneDNN overview

oneAPI Deep Neural Network Library (oneDNN)

CPU			GPU		
ARM	Intel	Others	AMD	Intel HD Graphics / Xᵉ	NVidia

oneDNN is a performance library for deep learning

- Helps developers create high performance deep learning frameworks
- Abstracts out instruction set and other complexities of performance optimizations
- Enabled by default on most popular frameworks (Tensorflow, Pytorch, PaddlePaddle)
- Open source for community contributions
- Native packages available for major Linux distributions

Github repository: https://github.com/oneapi-src/oneDNN

Specification: https://spec.oneapi.io/versions/latest/elements/oneDNN/source/index.html

oneDNN delivers performance in all major deep learning frameworks

oneDNN open-source library

Features:

- Optimizations targeting Multi-layer Perceptron (MLPs), Convolutional Neural Networks (CNNs) (1D, 2D and 3D), Recurrent Neural Networks (RNNs), Transformers, LLMs
- Inference: float32, tf32, bfloat16, float16, and int8(1) with support of dynamic/static quantization, and operator fusion
- Training: float32, tf32, bfloat16, float16(1)
- API: C and C++ with SYCL, OpenCL interoperability
- oneDNN API abstracts runtimes and devices: same program runs on multiple platforms
- Runtime dispatching for easy binary deployment and best performance on all platforms
- Just-in-time kernel compilation for small disk footprint, best performance, and portability

Support matrix:

- Compilers: Intel, GCC, CLANG, MSVC, DPC++
- OS: Linux, Windows, macOS(2)
- Runtimes: SYCL, OpenCL, OpenMP, TBB, Threadpool

Category	Functions
Graph API	• ~80 DL operations + wildcard • Scalable fusion: Graph, Partitions
Compute intensive operations	• (De-)Convolution • Matmul • Inner Product • RNN (LSTM, LSTMP, vanilla, GRU, AUGRU)
Memory bandwidth limited operations	• Normalization (batch, group, layer, local response) • Binary • Concat • Elementwise • LogSoftmax/Softmax • Pooling • PReLU • Resampling • Shuffle • Sum • Reduction
Data manipulation	• Reorder

(1) Low precision data types are supported only for platforms where hardware acceleration is available

(2) macOS does not have GPU support

oneDNN project development

Open design

- All API decisions are discussed publicly through RFCs on github (~30 RFCs the past 12 months)
- Open specification, migration under UXL foundation governance in progress
- Regular meetings with industrial contributors

Open-source development on github

- 90+ external PRs in the past 12 months
- Regular industrial contributors: Intel, ARM, Fujitsu, Codeplay.
- Native Linux packages maintained by Debian Science Team: Arch Linux, Fedora, FreeBSD, RedHat, Ubuntu

Cross platform

CPU	GPU
Intel/AMD: SSE4.1, AVX, AVX2, AVX512, AVX512/VNNI, AVX2/VNNI, AMX	Intel: GEN11, Xe LP, Xe HP, Xe HPG, Xe HPC
Arm: Aarch64(2), SVE 128/256/512	NVIDIA(2): all supported by cuBLAS/cuDNN
Others(1): openPOWER, IBMz, RISC-V (RV64, RVV)	AMD(2): all supported by rocBLAS/MIOpen

(1) Limited testing in validation
(2) Best performance obtained by linking to external closed-source libraries

UXL FOUNDATION
Unified Acceleration

oneDNN commitment to open development

Moving to shared code-ownership model

- Each backend will have its own code-owners, to speed up review/merging of Pull Requests
- Each backend-owner to own validation for their backend
- Intel team moving to public github repository as main development repository
- Expansion of public CI is under discussion

Simplifying new devices support with generic SYCL configuration

- Allows to run a functional oneDNN baseline build on any device with SYCL support.
- Covering all functionalities with pure SYCL reference kernels.
- Removing spurious dependencies to SYCL runtime backend objects.

Thank you!

UXL FOUNDATION
Unified Acceleration

oneDNN primitive API concepts

Key abstractions

Primitive

- primitive descriptor (lightweight implementation aware descriptor)
- primitive (jitted code)

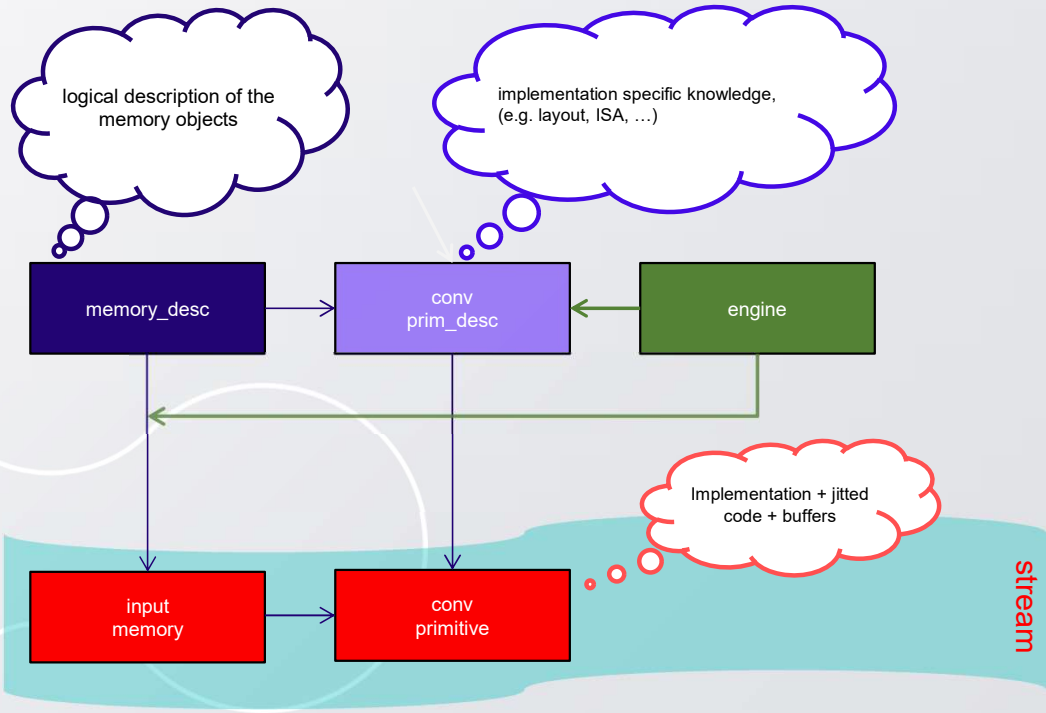
Memory

- memory desc (dims, data type, layout)
- memory (desc + engine, pointer to data)

Engine – execution device

Stream – execution context

Simplified programming model



UXL FOUNDATION
Unified Acceleration

oneDNN graph API concepts

Key abstractions

Graph

- Ops compose the graph and describe operations
- logical_tensor describe dependencies between ops

Partition

- Represents subgraph that can be run in a single compiled unit
- Must be queried from a graph

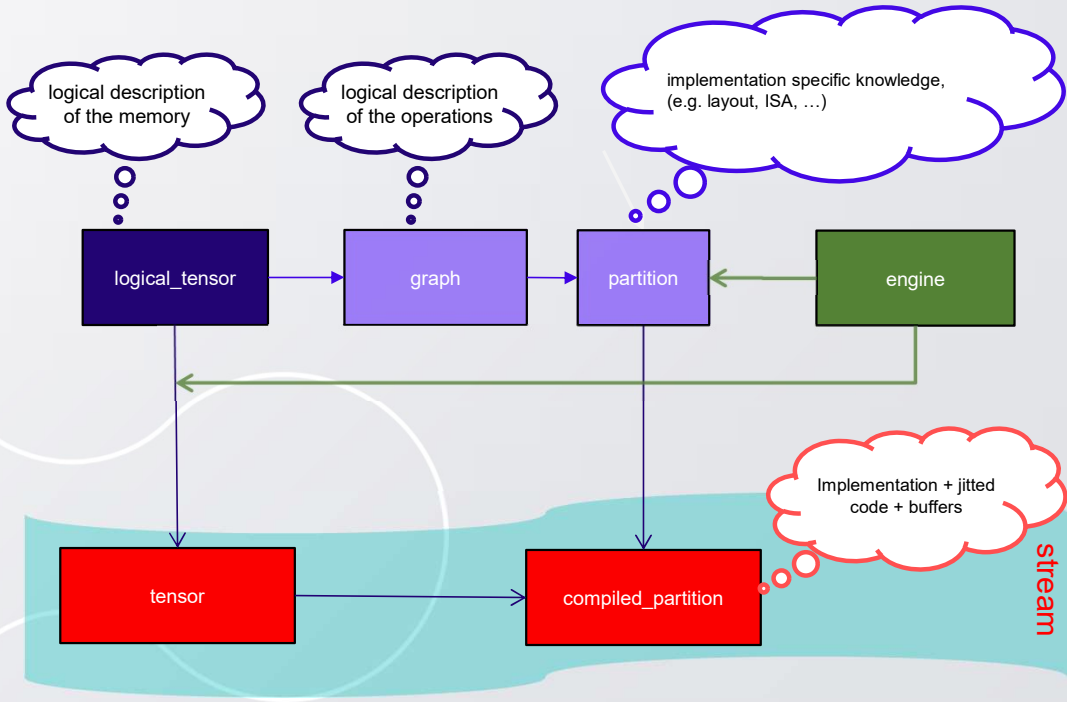
Tensor

- logical_tensor (id, dims, datatype, layout)
- tensor (logical_tensor + engine + pointer to data)

Engine – execution device

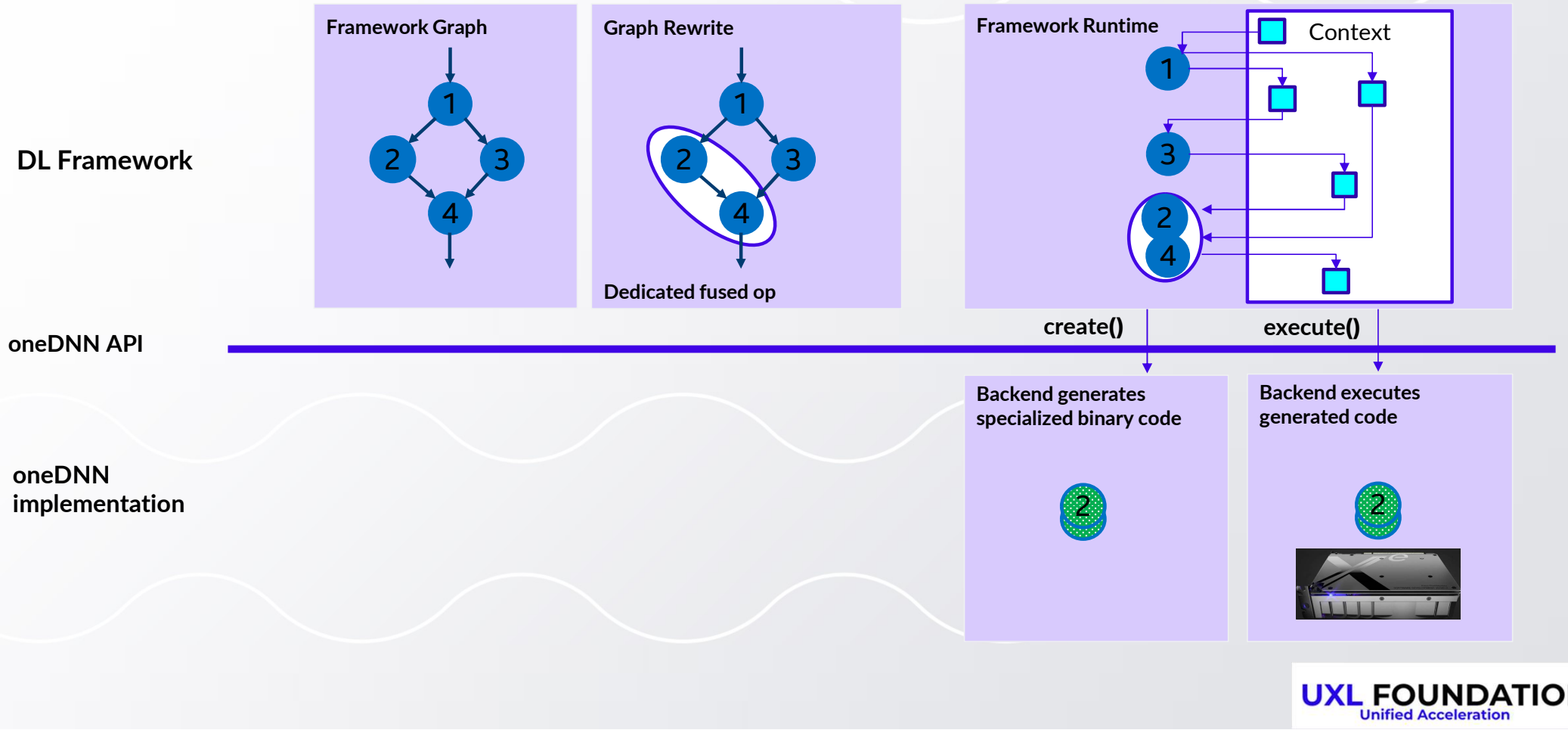
Stream – execution context

Simplified programming model



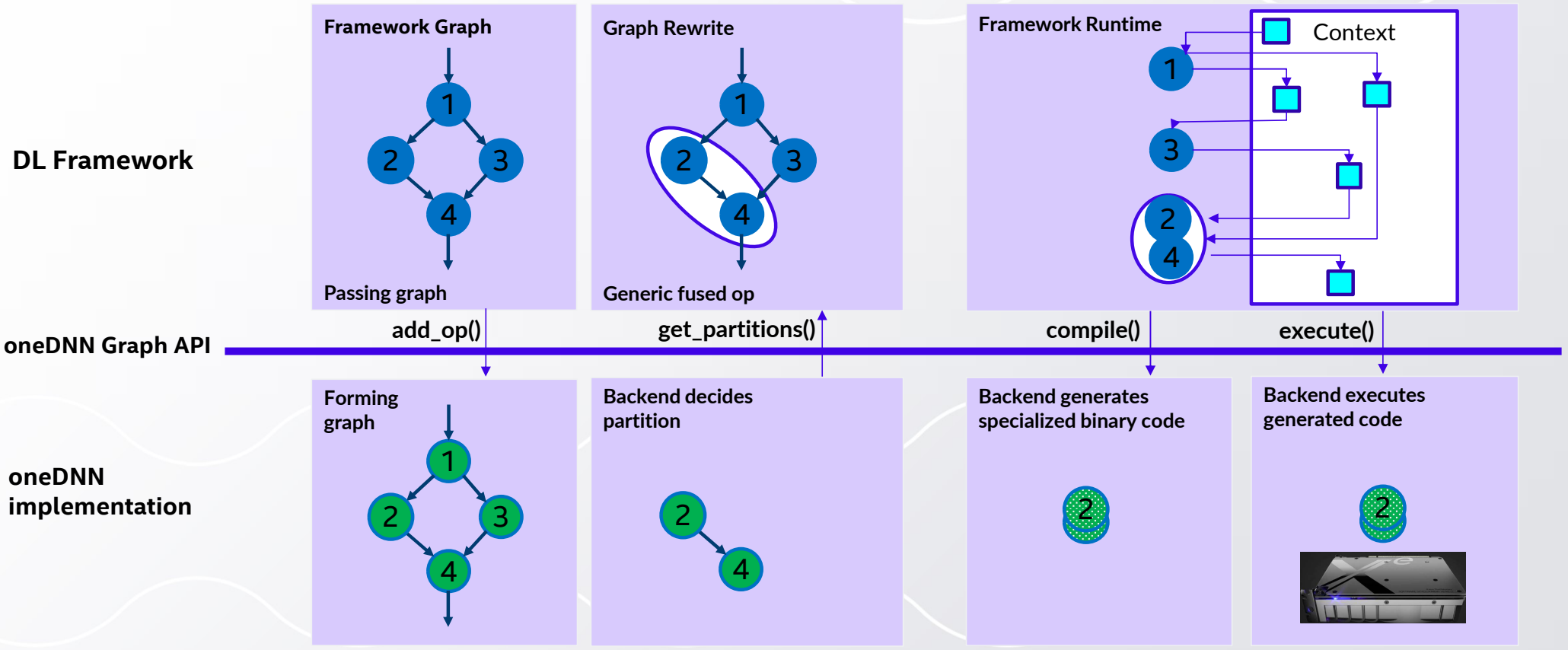
UXL FOUNDATION
Unified Acceleration

Framework integration: primitive API



UXL FOUNDATION
Unified Acceleration

Framework integration: graph API



UXL FOUNDATION
Unified Acceleration

oneDNN Programming Model

```
// Create GPU engine for the first (index is 0) GPU device
engine eng(engine::kind::gpu, 0);

// Create memory object: 5D tensor, f32, NCHW format
memory::dims tz = { 2, 3, 4, 5 };

// Memory descriptor: tensor sizes, data types, format tag
memory::desc mem_d(
    tz, memory::data_type::f32, memory::format_tag::nchw);

// Memory object, memory is allocated
memory mem(mem_d, eng);

// Primitive descriptor for Element-wise primitive (ReLU)
auto eltwise_pd =
    eltwise_forward::desc(eng, prop_kind::forward,
        algorithm::eltwise_relu, mem_d, mem_d, 0.0f);

// Element-wise primitive
auto eltwise = eltwise_forward(eltwise_pd);

// Create stream and execute primitive
auto s = stream(eng);
eltwise.execute(s, { { DNNL_ARG_SRC, mem },
                     { DNNL_ARG_DST, mem } });
```

- **Engine** is an abstraction of a computational device: a CPU, a specific GPU card in the system

oneDNN Programming Model

```
// Create GPU engine for the first (index is 0) GPU device
engine eng(engine::kind::gpu, 0);

// Create memory object: 5D tensor, f32, NCHW format
memory::dims tz = { 2, 3, 4, 5 };

// Memory descriptor: tensor sizes, data types, format tag
memory::desc mem_d(
    tz, memory::data_type::f32, memory::format_tag::nchw);

// Memory object, memory is allocated
memory mem(mem_d, eng);

// Primitive descriptor for Element-wise primitive (ReLU)
auto eltwise_pd =
    eltwise_forward::desc(eng, prop_kind::forward,
        algorithm::eltwise_relu, mem_d, mem_d, 0.0f);

// Element-wise primitive
auto eltwise = eltwise_forward(eltwise_pd);

// Create stream and execute primitive
auto s = stream(eng);
eltwise.execute(s, { { DNNL_ARG_SRC, mem },
                     { DNNL_ARG_DST, mem } });
```

- **Engine** is an abstraction of a computational device: a CPU, a specific GPU card in the system
- **Memory objects** encapsulate handles to memory allocated on a specific engine, tensor dimensions, data type, and memory format

oneDNN Programming Model

```
// Create GPU engine for the first (index is 0) GPU device
engine eng(engine::kind::gpu, 0);

// Create memory object: 5D tensor, f32, NCHW format
memory::dims tz = { 2, 3, 4, 5 };

// Memory descriptor: tensor sizes, data types, format tag
memory::desc mem_d(
    tz, memory::data_type::f32, memory::format_tag::nchw);

// Memory object, memory is allocated
memory mem(mem_d, eng);

// Primitive descriptor for Element-wise primitive (ReLU)
auto eltwise_pd =
    eltwise_forward::desc(eng, prop_kind::forward,
        algorithm::eltwise_relu, mem_d, mem_d, 0.0f);

// Element-wise primitive
auto eltwise = eltwise_forward(eltwise_pd);

// Create stream and execute primitive
auto s = stream(eng);
eltwise.execute(s, { { DNNL_ARG_SRC, mem },
                     { DNNL_ARG_DST, mem } });
```

- **Engine** is an abstraction of a computational device: a CPU, a specific GPU card in the system
- **Memory objects** encapsulate handles to memory allocated on a specific engine, tensor dimensions, data type, and memory format
- **Primitive** is a functor object that encapsulates a particular computation such as forward convolution

oneDNN primitive programming model

```
// Create GPU engine for the first (index is 0) GPU device
engine eng(engine::kind::gpu, 0);

// Create memory object: 5D tensor, f32, NCHW format
memory::dims tz = { 2, 3, 4, 5 };

// Memory descriptor: tensor sizes, data types, format tag
memory::desc mem_d(
    tz, memory::data_type::f32, memory::format_tag::nchw);

// Memory object, memory is allocated
memory mem(mem_d, eng);

// Primitive descriptor for Element-wise primitive (ReLU)
auto eltwise_pd =
    eltwise_forward::desc(eng, prop_kind::forward,
        algorithm::eltwise_relu, mem_d, mem_d, 0.0f);

// Element-wise primitive
auto eltwise = eltwise_forward(eltwise_pd);

// Create stream and execute primitive
auto s = stream(eng);
eltwise.execute(s, { { DNNL_ARG_SRC, mem },
                     { DNNL_ARG_DST, mem } });
```

- **Engine** is an abstraction of a computational device: a CPU, a specific GPU card in the system
- **Memory objects** encapsulate handles to memory allocated on a specific engine, tensor dimensions, data type, and memory format
- **Primitive** is a functor object that encapsulates a particular computation such as forward convolution
- **Streams** encapsulate execution context tied to a particular engine