oneDNN Graph API: New Features and Update

Tao Lv, Intel
2025/6/26

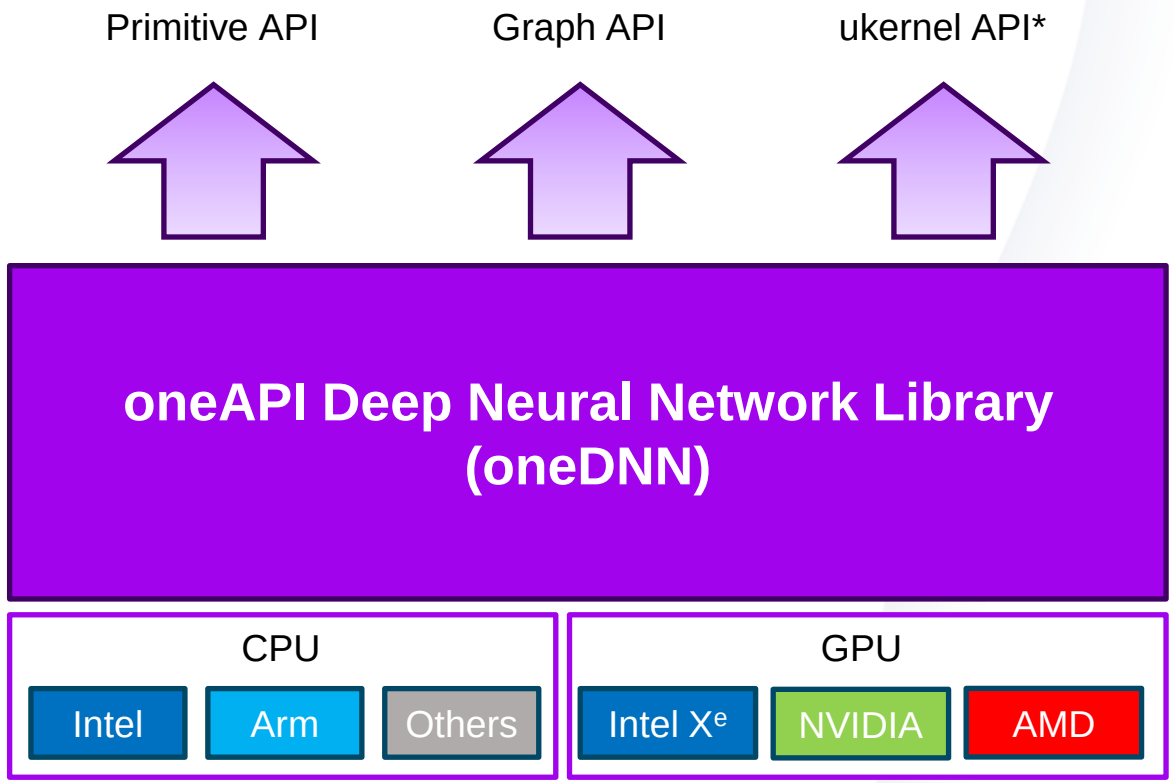
UXL
Unified Acceleration Foundation

Agenda

- oneDNN APIs
- oneDNN Graph API: concepts and programming model
- New features in oneDNN v3.7/v3.8
- PyTorch integration and performance
- Future directions

oneDNN APIs

An API for each usage



Why Graph API?

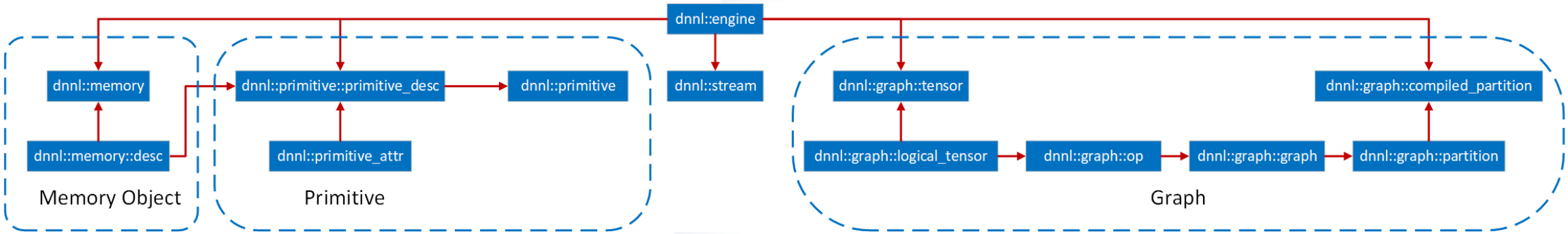
- Stable API surface to express complex operations and fusions (SDPA, GQA, MQA, MLP etc.)
- Simplify compute epilogue fusions (conv/matmul + bias + activation)
- Counterpart to cuDNN Graph API

* ukernel API is experimental and CPU only.

oneDNN Graph API

API Concepts

- Share the common engine and stream abstractions between primitive API and graph API.
- Logical tensor defines the meta-data (shape, dtype, layout, etc) of tensor and represents the edges in a graph.
- Op defines the operations with attributes and represents the nodes in a graph.
- Graph contains a collection of operations and their input and output logical tensors.
- Partition: a subgraph for target specific optimizations.
- Compiled partition: compiled executable object from a partition with given engine and shape information.
- Tensor: data storage + logical tensor + engine.



https://uxlfoundation.github.io/oneDNN/dev_guide_graph_basic_concepts.html

Programming – graph, op, and partition

```
// score = query x key.T
auto query = logical_tensor(id++, dt, qv_sz, layout_type::strided);
auto key = logical_tensor(id++, dt, k_sz, layout_type::strided);
auto score = logical_tensor(id++, dt, score_sz, layout_type::strided);
auto bmm1 = op(id++, op::kind::MatMul, "bmm1");
bmm1.set_attr<bool>(op::attr::transpose_b, true);
bmm1.add_inputs({query, key});
bmm1.add_outputs({score});
```

```
// scaled_score = score / scale
auto scale = logical_tensor(id++, dt, scale_sz, layout_type::strided);
auto scaled_score
        = logical_tensor(id++, dt, score_sz, layout_type::strided);
auto scale_div = op(id++, op::kind::Divide, "scale_div");
scale_div.add_inputs({score, scale});
scale_div.add_outputs({scaled_score});
```

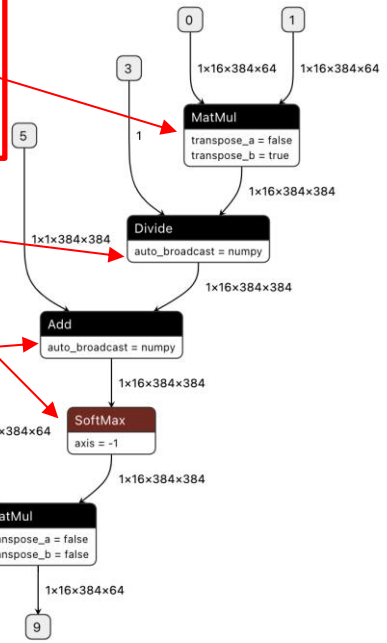
```
// masked_score = scaled_score + mask
auto mask = logical_tensor(id++, dt, mask_sz, layout_type::strided);
auto masked_score
        = logical_tensor(id++, dt, score_sz, layout_type::strided);
auto mask_add = op(id++, op::kind::Add, "mask_add");
mask_add.add_inputs({scaled_score, mask});
mask_add.add_outputs({masked_score});
```

```
// attention_probs = softmax(masked_score)
auto probs = logical_tensor(id++, dt, score_sz, layout_type::strided);
auto softmax = op(id++, op::kind::SoftMax, "softmax");
softmax.set_attr<int64_t>(op::attr::axis, -1);
softmax.add_inputs({masked_score});
softmax.add_outputs({probs});
```

```
// attention_output = attention_probs x value
auto value = logical_tensor(id++, dt, k_sz, layout_type::strided);
auto output = logical_tensor(id++, dt, qv_sz, layout_type::stride
auto bmm2 = op(id++, op::kind::MatMul, "bmm2");
bmm2.add_inputs({probs, value});
bmm2.add_outputs({output});
```

```
// Construct a sdpa graph with engine kind and operations.
dnnl::graph::graph sdpa(ekind);
sdpa.add_op(bmm1);
sdpa.add_op(scale_div);
sdpa.add_op(mask_add);
sdpa.add_op(softmax);
sdpa.add_op(bmm2);
sdpa.finalize();

// Get partitions from the sdpa graph.
std::vector<partition> partitions = sdpa.get_partitions();
```



https://github.com/uxlfoundation/oneDNN/blob/main/examples/graph/sdpa.cpp

Programming – compile and execute

```
// Create engine and stream
engine eng = dnnl::engine(kind::cpu, 0);
stream strm = dnnl::stream(eng);

// Compile the partition with inputs, outputs, and an engine.
compiled_partition cp = partitions[0].compile(
        {query, key, scale, mask, value}, {output}, eng);

// Create tensor objects
auto ts_query = tensor(query, eng, q_ptr);
auto ts_key = tensor(key, eng, k_ptr);
auto ts_scale = tensor(scale, eng, s_ptr);
auto ts_mask = tensor(mask, eng, m_ptr);
auto ts_value = tensor(value, eng, v_ptr);
auto ts_output = tensor(output, eng, o_ptr);

// Execute the compiled partition of sdpa.
cp.execute(
        strm, {ts_query, ts_key, ts_scale, ts_mask, ts_value}, {ts_output});

// Wait for the computation to finish.
strm.wait();
```

Native CPU execution

```
// Create engine and stream from sycl device, context, and queue.
engine eng = sycl_interop::make_engine(dev, ctx);
stream strm = sycl_interop::make_stream(eng, queue);

// Compile the partition with inputs, outputs, and an engine.
compiled_partition cp = partitions[0].compile(
        {query, key, scale, mask, value}, {output}, eng);

// Create tensor objects
auto ts_query = tensor(query, eng, q_ptr);
auto ts_key = tensor(key, eng, k_ptr);
auto ts_scale = tensor(scale, eng, s_ptr);
auto ts_mask = tensor(mask, eng, m_ptr);
auto ts_value = tensor(value, eng, v_ptr);
auto ts_output = tensor(output, eng, o_ptr);

// Execute the compiled partition of sdpa.
cp.execute(
        strm, {ts_query, ts_key, ts_scale, ts_mask, ts_value}, {ts_output});

// Wait for the computation to finish.
strm.wait();
```

SYCL execution

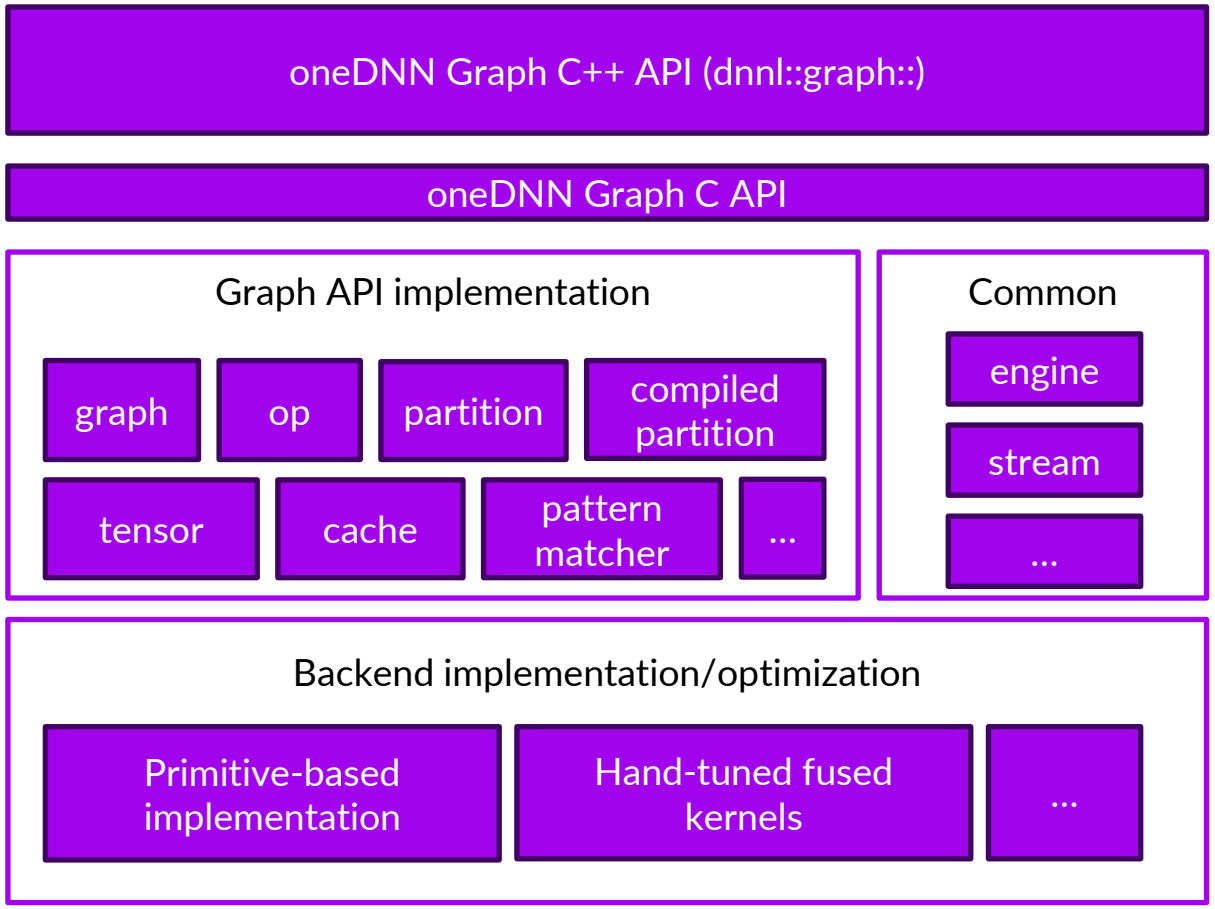
The only difference is how engine and stream are created.

Operation Coverage

- 86 basic AI operations covering both inference and training: https://uxlfoundation.github.io/oneDNN/graph_supported_operations.html
- Based on the operations, oneDNN Grpah API supports:
  - Convolution/MatMul based epilogue fusions.
  - SDPA/GQA/MQA fusions used in LLM.
  - MLP, Gated-MLP, Convolution residual blocks.
  - … with different optimization levels for f32/bf16/f16/fp8/int4 on Intel CPU and GPU.
- The main purpose is not to implement the operations within oneDNN, but to construct and optimize fusion patterns with them.

Category	Operation
Unary elementwise	Abs, Square, Sqrt, Exp, Erf, Tanh, Log, Pow, Round, etc.
Binary elementwise	Add, Subtract, Multiply, Divide, Maximum, Minimum, etc.
Activation	ReLU, LogSoftmax, Softmax, Sigmoid, SoftPlus, Clamp, Elu, GELU, HardSwish, LeakyReLU, Mish, PReLU, etc.
Neural network	Convolution, ConvTranspose, MatMul, BatchNorm, LayerNorm, GroupNorm, MaxPool, AvgPool, etc.
Reduction	ReduceL1, ReduceL2, ReduceMax, ReduceMean, ReduceMin, ReduceProd, ReduceSum
Data manipulation	Reorder, Reshape, Transpose, Concat, Interpolate, GenIndex, etc.
Low Precision	Quantize, Dequantize, TypeCast

Software Architecture



What's new in oneDNN v3.7/v3.8

https://github.com/uxlfoundation/oneDNN/releases/tag/v3.7 (Feb. 2025)

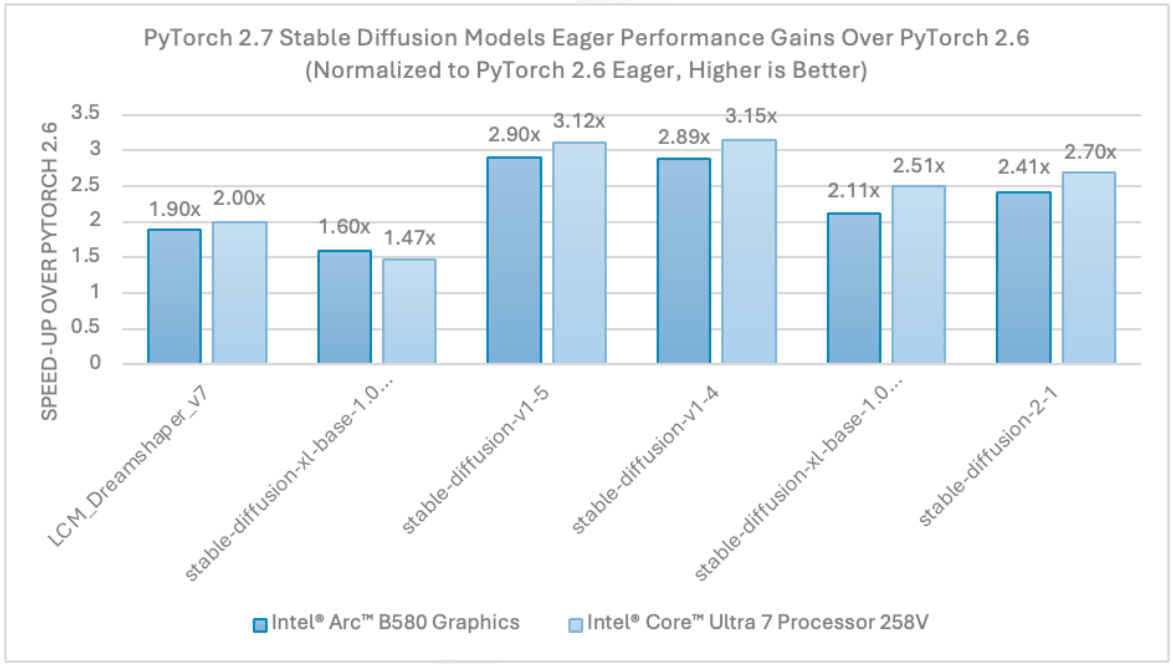
https://github.com/uxlfoundation/oneDNN/releases/tag/v3.8 (May 2025)

- Continued performance optimizations for SDPA (incl. GQA) on Intel CPU and GPU platforms, covering f32, bf16, f16, and int8 data types.
- SDPA variants with explicit or implicit attention masks.
- SDPA variants with int4/int8 compressed Key and Value.
- "Safe" softmax to handle all –infinity attention masks, aligning the numerical semantics with PyTorch.
- Initial support for Gated-MLP on Intel platforms.
- Initial support for Graph API on NVIDIA GPU platforms (through cuDNN and generic SYCL kernels).
- Better validation and benchmarking support through benchdnn.

Accelerate PyTorch 2.7 on Intel GPU

https://pytorch.org/blog/pytorch-2-7-intel-gpus/

- PyTorch 2.7 release has oneDNN Graph API integration to accelerate SDPA inference and attention-based models on Intel GPU platforms.
  - *"…, Stable Diffusion float16 inference achieved up to 3x gain over PyTorch 2.6 release on …"*



UXL
Unified Acceleration Foundation

What's expected in PyTorch 2.8

SDPA optimization and enhancements through Graph API

- Better performance from oneDNN v3.8.1 for Intel GPU. (#152091)
- Bf16/f16 SDPA with f32 intermediate data type. (#152091)
- Optimization for head size <= 576.
- Optimization for f32 SDPA with implicit causal mask.
- Optimization for GQA, d_qk != d_v. (#150992)
- Enable "safe" softmax in PyTorch SDPA integration. (#151976)
- …

Future directions

- More implicit attention mask modes: top-left -> bottom-right, etc. (oneDNN v3.9, #2885)
- SDPA training forward and backward. (oneDNN v3.9, #3233)
- Host-side attention scale for GPU execution to save host overhead. (oneDNN v3.9)
- GQA variants with soft-capping used by Gemma models. (oneDNN v3.9)
- Asynchronous threadpool runtime (#3305).
- More data types (fp8, fp4, etc.) and the combinations.
- Advanced attention trends: Paged Attention, fused MLA, Chunked Prefill, etc.
- Optimized kernels for MLP variants.
- Reduce compilation and kernel generation time.
- …

Report requests and questions on oneDNN GitHub: https://github.com/uxlfoundation/oneDNN/issues

Thanks