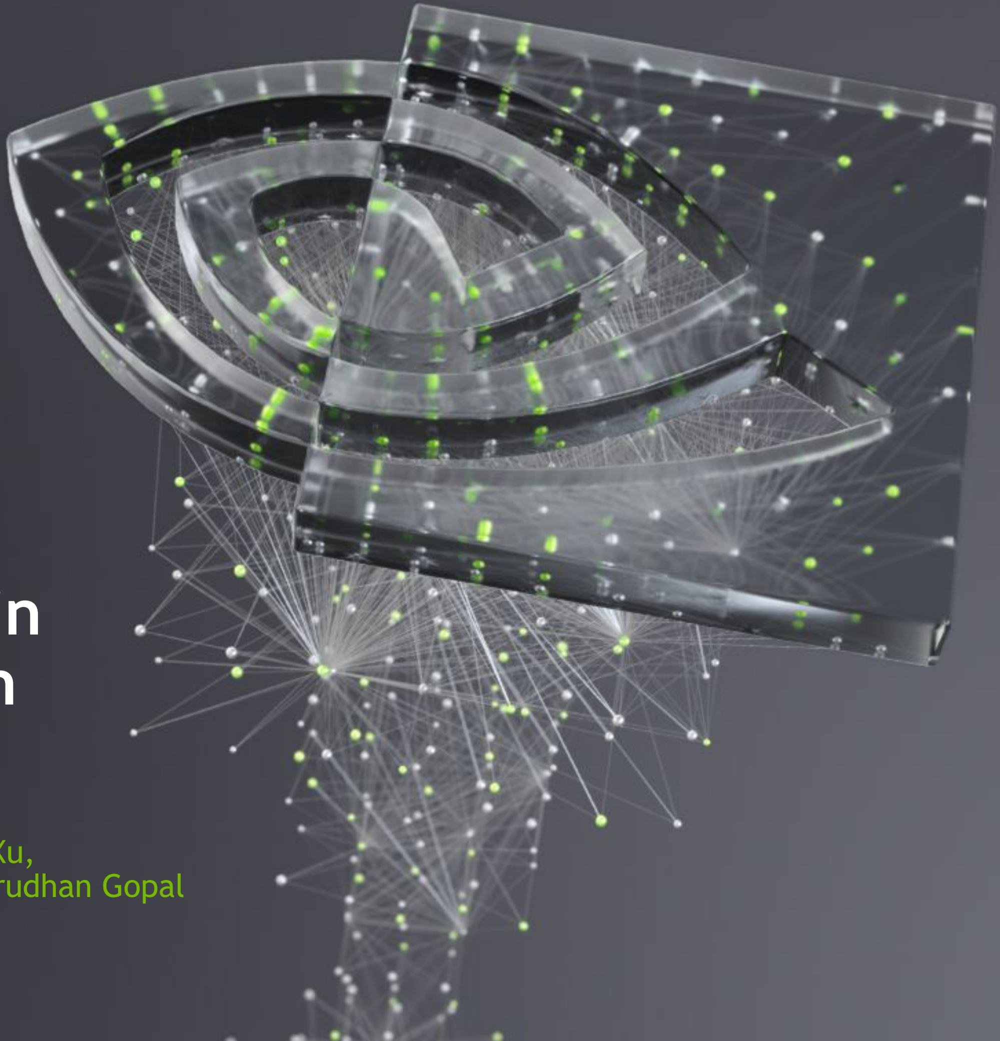


CUDNN V8: New Advances in Deep Learning Acceleration

Mostafa Hagog, Kevin Vincent, Yang Xu,
Mathieu Zhang, Seth Walters, Scott Yokim, Anerudhan Gopal
March, 2020

CUDNN: GPU ACCELERATED DEEP LEARNING

Co-designed with each architecture generation (e.g. Volta Tensor Cores)

Simple API to integrate into any machine learning framework or toolkit

Over two million downloads, available in every cloud, data center, embedded and graphics

New releases approximately monthly

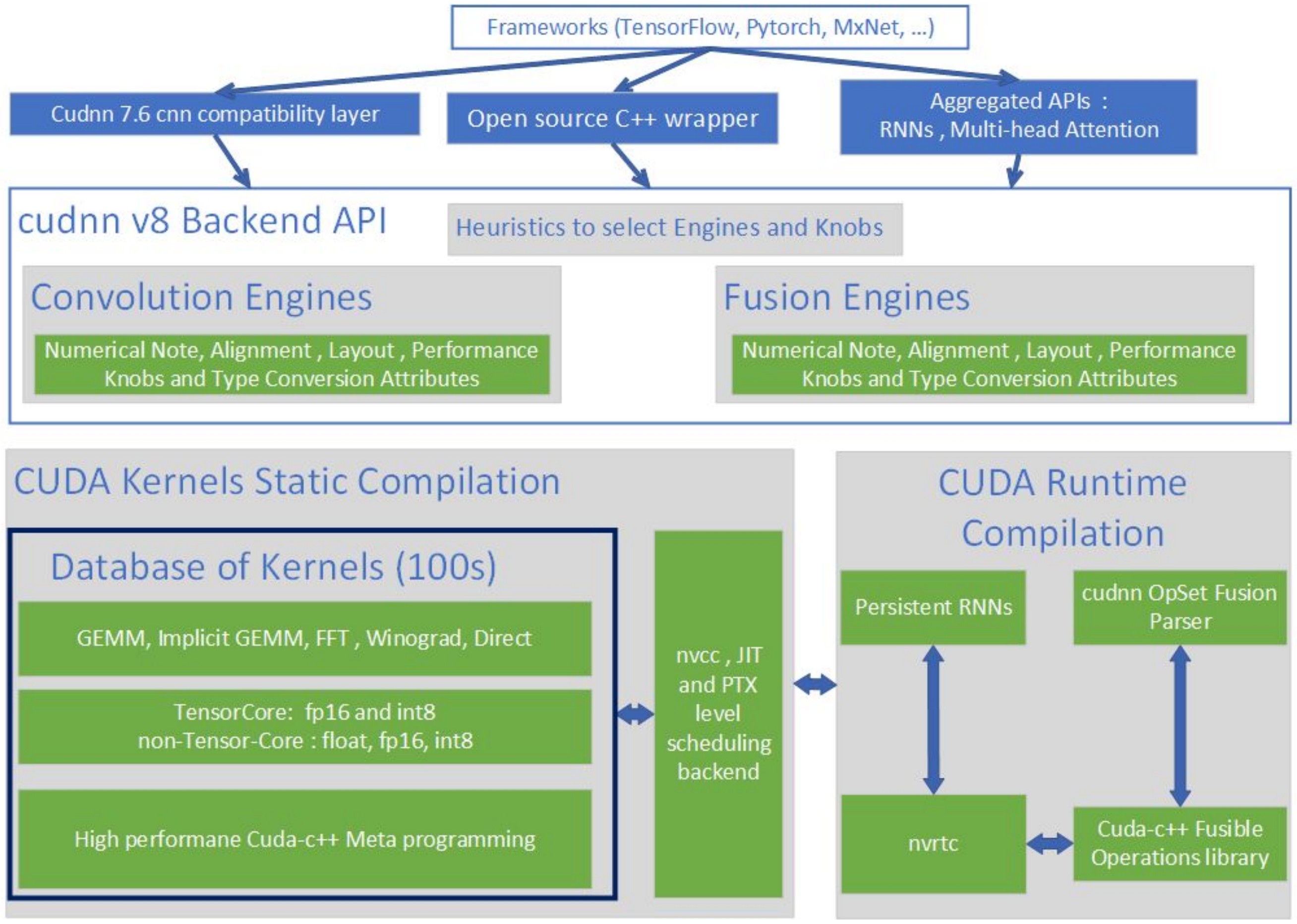
RETROSPECTIVE ON CUDNN V7: LIMITATIONS

- Rigid Functional APIs
    - In the past, each new DL operation required a brand new API
- Rigid performance selection
    - The algorithm concept does not address new advances in software and hardware
- Numerical properties, alignment, layouts and type-conversion are not easily communicated
- Large library size a burden on distribution and memory usage

INTRODUCING CUDNN V8

- New low-level backend API
  - Provides flexibility in performance tuning and performance selection
- New high level C++ API for ease of use, released as open source
- Backward compatibility layer -- no code changes wrt to cuDNN 7.6 to use cuDNN v8
- Performance improvements on Volta and Turing with new kernels and pre-trained heuristics
- Bug fixes and API cleanup, support cuda-graph capture and a new deprecation policy
- Address Library size by splitting library based on usage
- New runtime fusion capabilities to adapt quickly to new functionalities

CUDNN V8 SOFTWARE ARCHITECTURE BLOCK DIAGRAM


Frameworks (TensorFlow, Pytorch, MxNet, ...)
Cudnn 7.6 cnn compatibility layer
Open source C++ wrapper
Aggregated APIs : RNNs , Multi-head Attention
cudnn v8 Backend API
Heuristics to select Engines and Knobs
Convolution Engines
Numerical Note, Alignment , Layout , Performance Knobs and Type Conversion Attributes
Fusion Engines
Numerical Note, Alignment , Layout , Performance Knobs and Type Conversion Attributes
CUDA Kernels Static Compilation
Database of Kernels (100s)
GEMM, Implicit GEMM, FFT , Winograd, Direct
TensorCore: fp16 and int8
non-Tensor-Core : float, fp16, int8
High performane Cuda-c++ Meta programming
nvcc , JIT and PTX level scheduling backend
CUDA Runtime Compilation
Persistent RNNs
cudnn OpSet Fusion Parser
nvrtc
Cuda-c++ Fusible Operations library

NEW AND IMPROVED CONCEPTS IN CUDNN V8

- **Tensor** -- Support graphs, more shapes , and 64 bit addressing
- **Operation** -- Describes a computation node
- **Graphs** -- a DAG of operations connected by tensors to express computations
- **Engines and Knobs** -- Abstractions of kernels and performance options
- **Attributes** -- Queryable properties of engines (e.g. numerical and alignment properties)
- **Non-eager execution** -- Separate constructions and validation of computation from the execution

IMPROVED TENSOR DESCRIPTOR

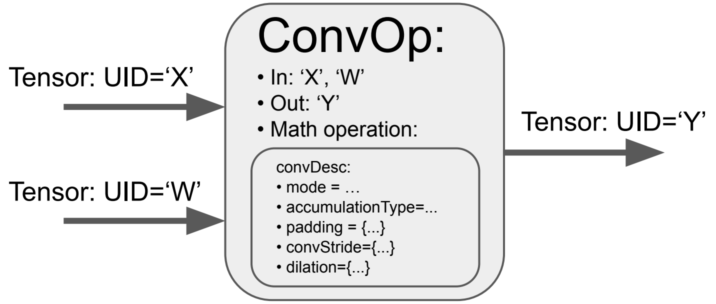
- New in v8
  - Support 64 bit addressing
  - Convolution groups dimension
- Support Fusion Graphs
  - Encode symbolic mathematical relationship between tensors in each operation node
  - “UID” use to uniquely identify a tensor
  - “isVirtual” use to indicate whether the tensor exists in memory or needs to be written out to memory

TensorDesc:

- UID = ‘X’
- isVirtual = false
- dataType = CUDNN_DATA_HALF
- alignment = 16B
- dim = {32, 64, 56, 56}
- stride = {200704, 1, 3584, 64}

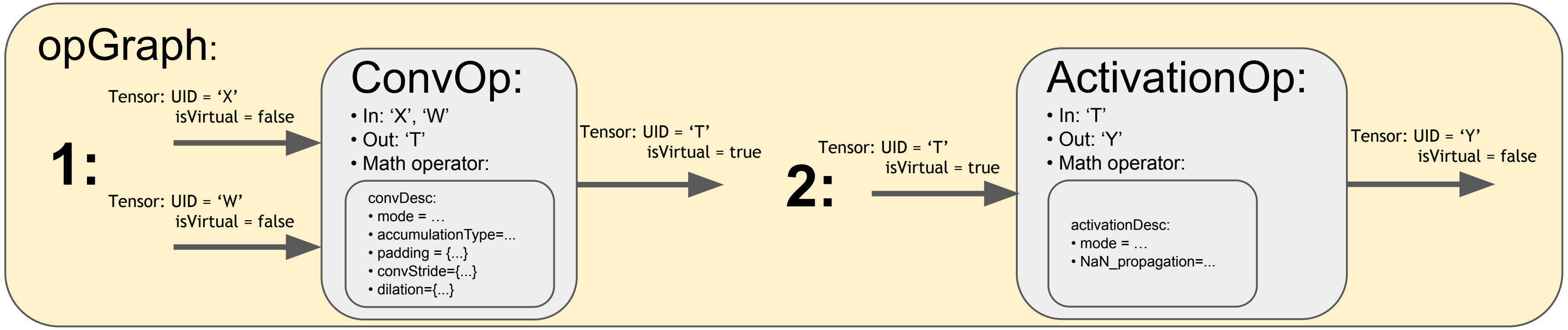
NEW CONCEPT: OPERATION DESCRIPTOR

- Operation
  - Elementary unit to express a computation
  - Each node in the computation graph is an operation
- Each operation node contains information about:
  - I/O: input/output tensors' size, datatype, layout, UID
  - Parameters and settings that determine the mathematical behavior
- Example operations:
  - convolution, bias , activation
- Support 12 operations today
  - More to come in future releases

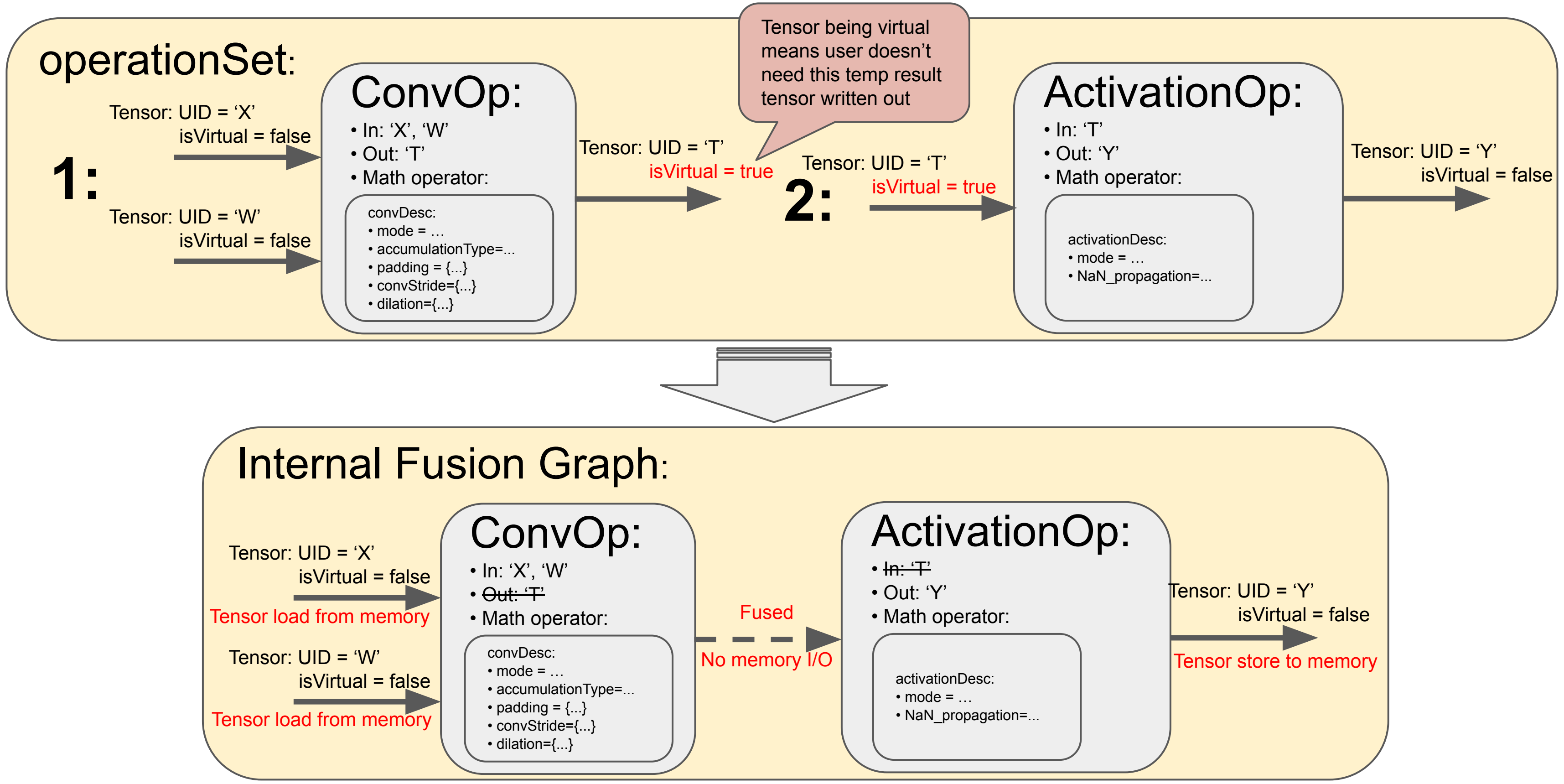


NEW CONCEPT: OPERATION GRAPHS

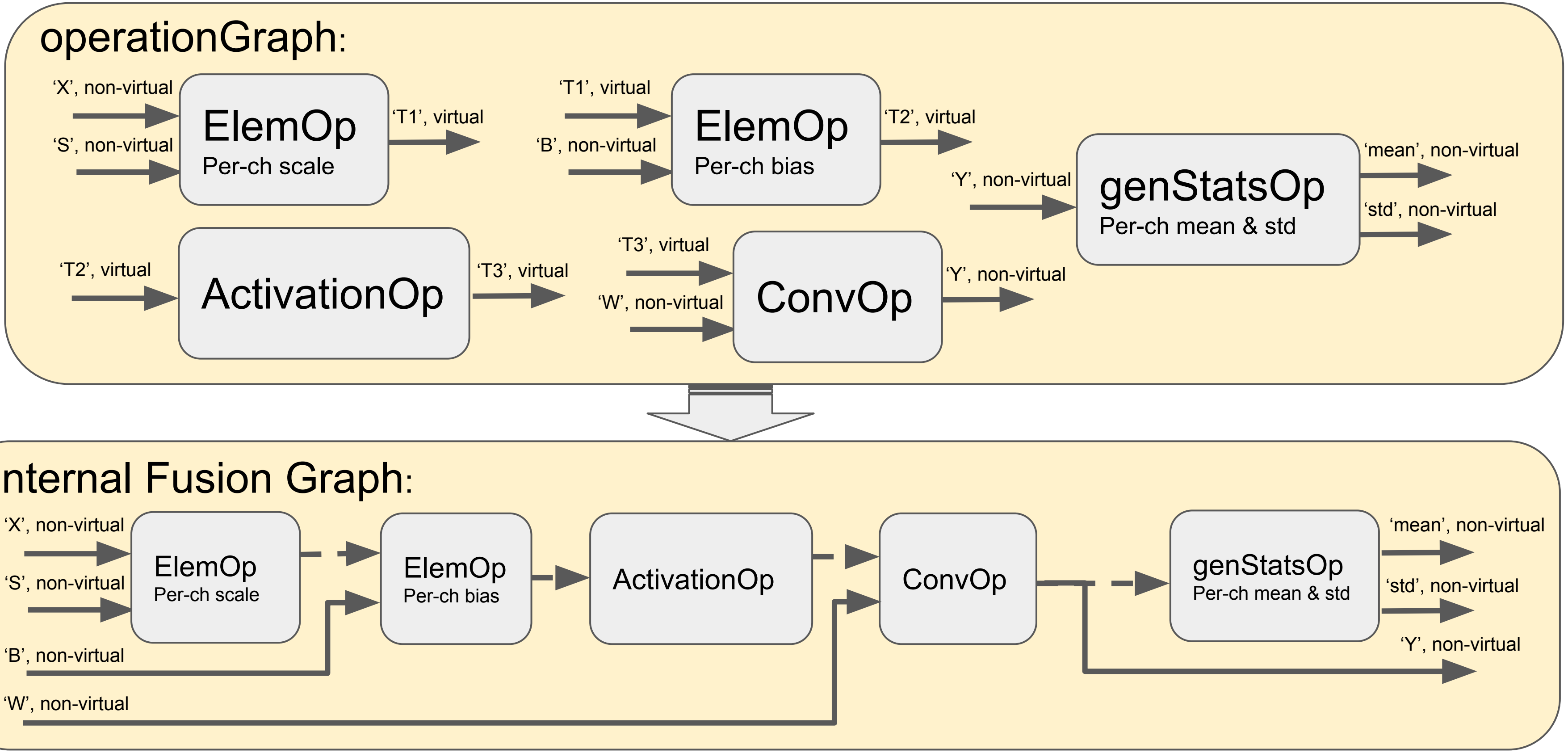
- An array of operation nodes representing a computation graph
- Uses Tensor UIDs to deduce the data-flow of the computation graph
- Once user expresses the computation as a graph, cuDNN can optimize it under the hood
- cuDNN can continue to optimize in future releases without changes to user code
- Easier to extend optimizations (E.g. conv+relu vs. conv+gelu)



OPERATION GRAPH FUSION EXAMPLE: CONV+RELU FUSION

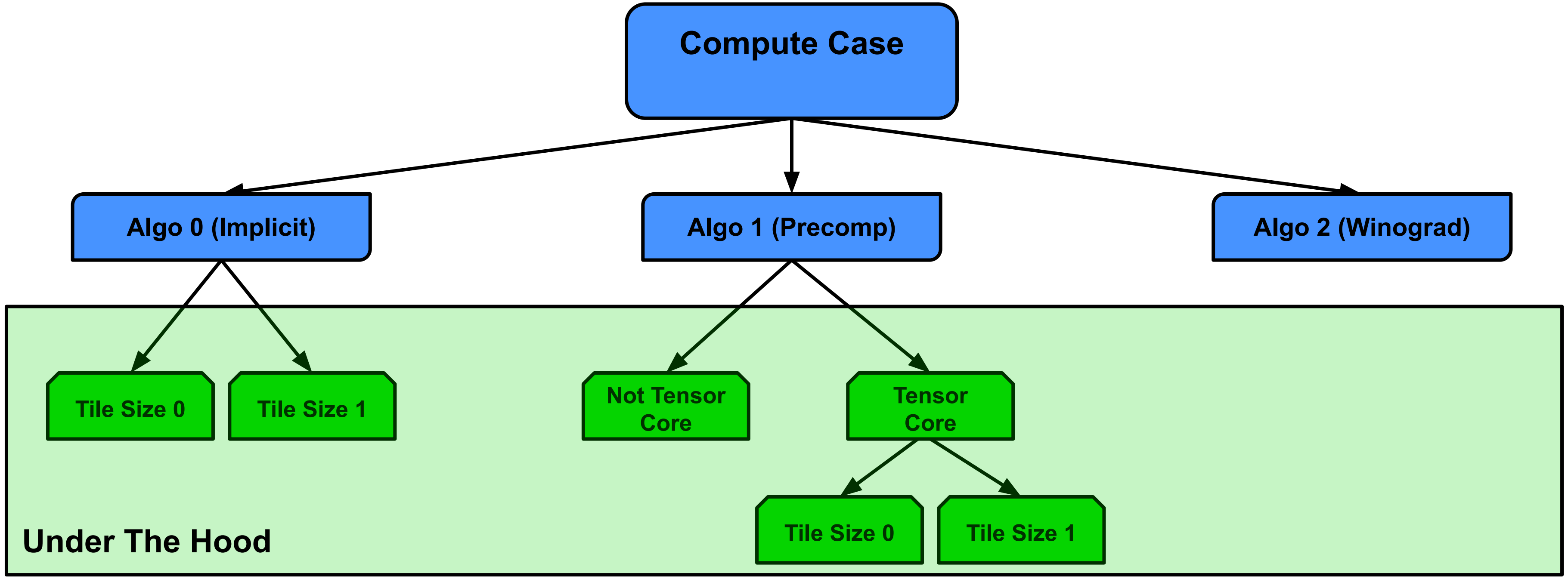


ADVANCED USAGE: CONVOLUTION + BATCH NORMALIZATION



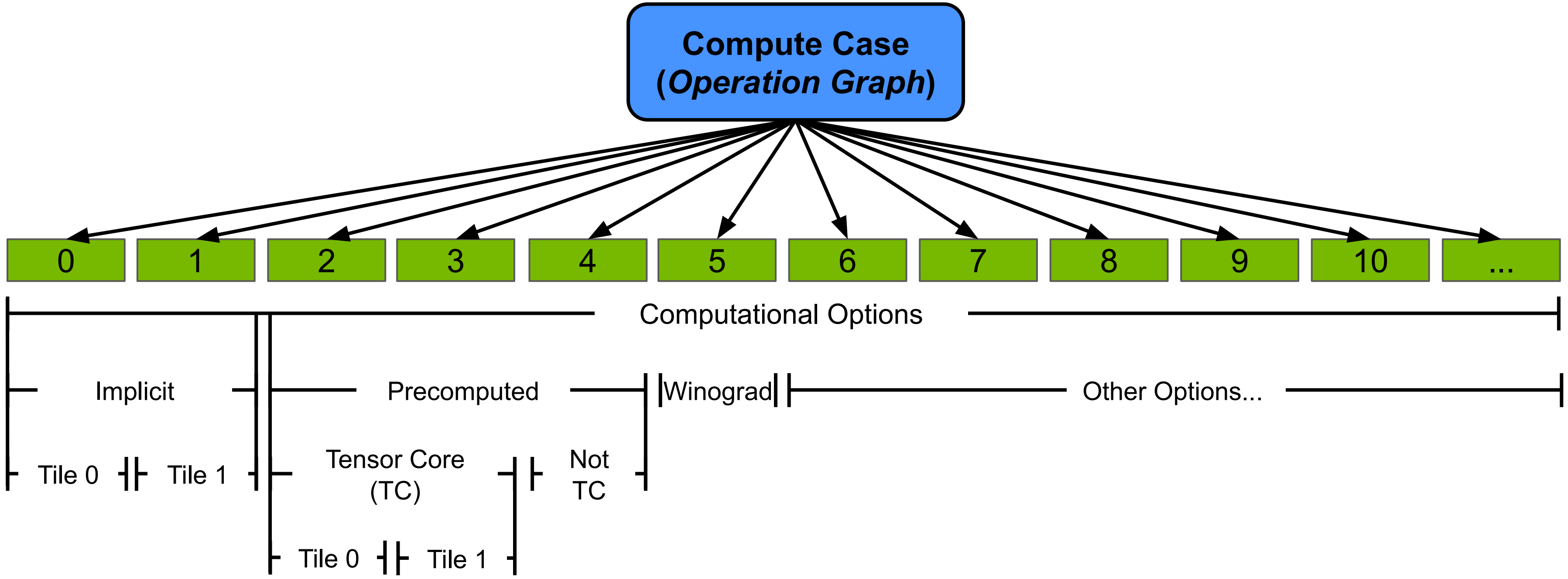
cuDNN v7.6 ALGORITHMS

Blue are visible to user in cuDNN v7.6 API
Green are chosen under the hood in cuDNN v7.6 API



DATABASE OF KERNELS

Compute Case: A mathematical problem to solve; e.g. Convolution.
Computational Option: A method able to compute a given compute case; e.g. a kernel.



GROUPING COMPUTATIONAL OPTIONS

Problem:

- Flat array of options is unmanageable
- “Algorithm” buckets aren’t scale-able

Solution:

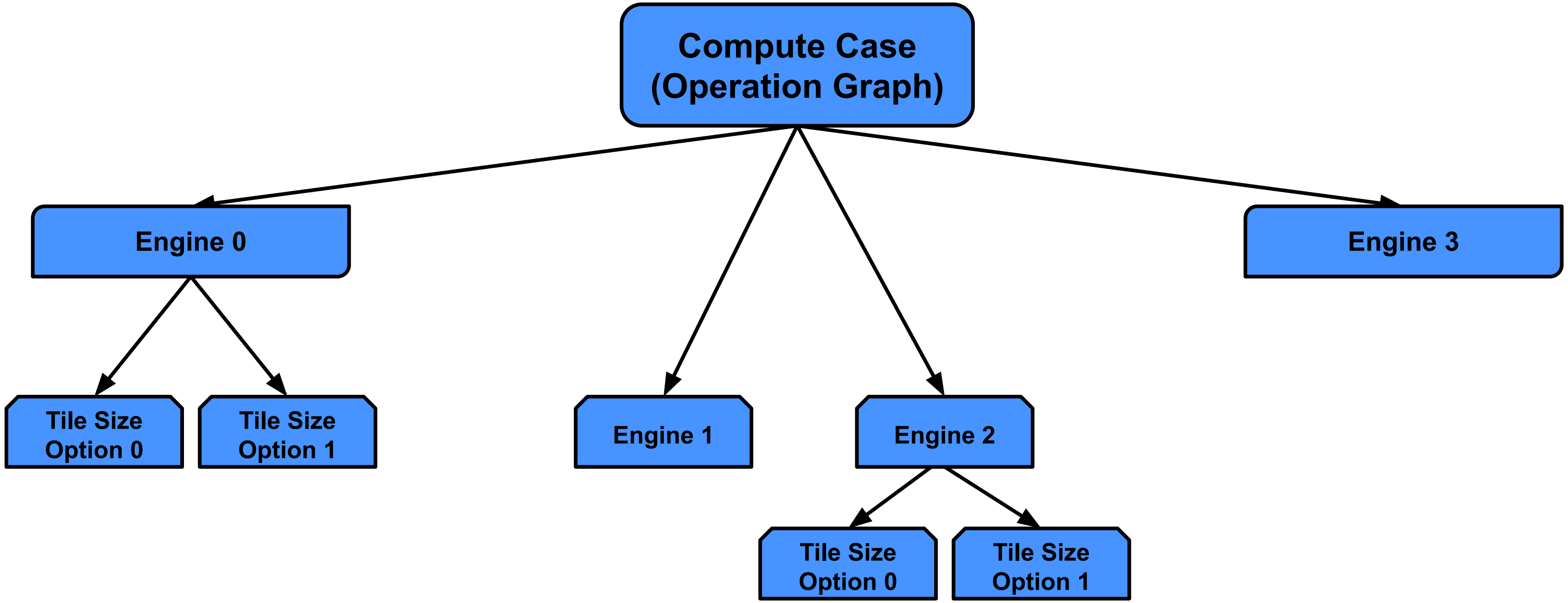
- Define Attributes for each compute option:
  - Numerics (Determinism, Winograd, …)
  - Data Alignment Requirement
- Group options with common attributes together
  - ex: “Implicit” group
- Provide selector for options available within group
  - ex: “Select this Tile Size Option”

NEW CONCEPT: ENGINES

- **Group of Computational Options: Engine**
  - Combines several computational options with identical attributes
- **Option Selector Within Engine: Knob**
  - Allows users to select a specific option available in a given Engine
  - ex: “Select Tile Size Option=0”
- **Benefits**
  - Computational options are visible and selectable by the user
  - Finer control of performance selection
  - Programmatic visibility and control of numerics and alignment

CUDNN V8 ENGINES

Blue are visible to user in cuDNN v7.6 API
Green are chosen under the hood in cuDNN v7.6 API



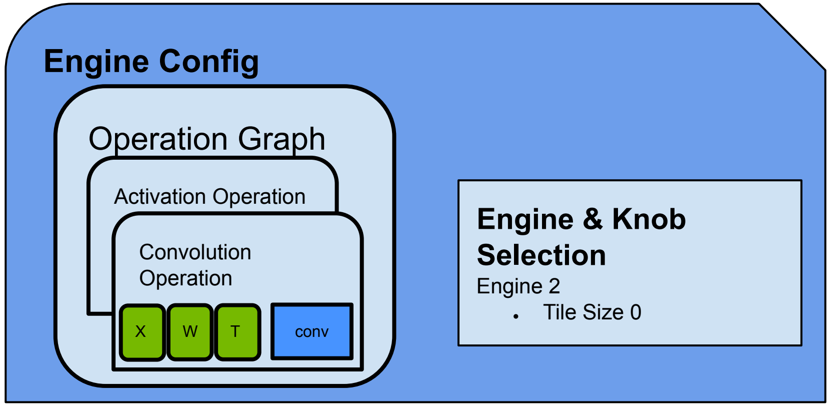
EXAMPLE ENGINE ATTRIBUTES

Numerical Note: Notable numerical difference in computed results; such as non-determinism & Winograd computations. These can increase performance at the cost of non-ideal numerical results.

Engine Index	**Numerical Notes**	**Knob Choices**	**Alignment**
0		Tile Size Options (0 & 1)	sizeof(elem)
1			sizeof(elem)
2	Tensor Core	Tile Size Options (0 & 1)	16B
3	Winograd		sizeof(elem)

NEW CONCEPT: ENGINE CONFIG

Engine Config: Descriptor containing an Operation Graph, Engine Index and Knob Choices. Fully describes a computation in cuDNN v8.

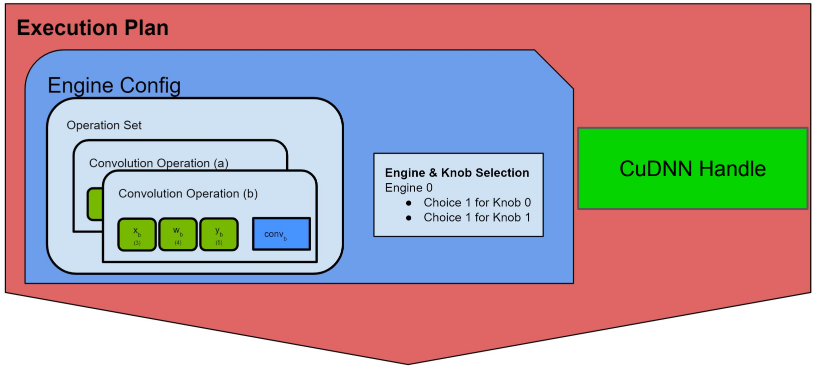


NEW HEURISTIC API

- **Motivation:**
    - For a given compute case, there are many compute options to choose from
    - Some users need determinism, some can’t spare any workspace memory
- **New Heuristic API:**
    - Returns list of EngineConfig sorted by expected performance
    - Guaranteed to contain a deterministic non-workspace EngineConfig
- **Benefits:**
    - Users can query Numerics, Knob Choices, Alignment from the EngineConfig
    - Users can override chosen Engine & Knobs
    - Users can auto-tune across all returned EngineConfigs

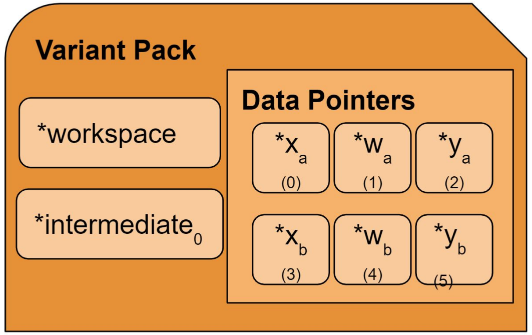
NEW CONCEPT: NON-EAGER EXECUTION

- Execution Plan
  - A container for executing a computation graph on the GPU
  - Includes graph and engine configs
  - Separate APIs for build and execute stages
- Build (`cudnnBackendFinalize`)
  - Search for engine that supports the computation
  - Query Heuristics for best performance
  - Calculate work-space
- Execute
  - Executes the computation graph on the GPU
- **Benefits:**
  - Amortise CPU overhead -- runtime compilation, querying heuristics, etc.
  - Reduce CPU time to launch kernels to couple of microseconds
  - Create once use multiple times



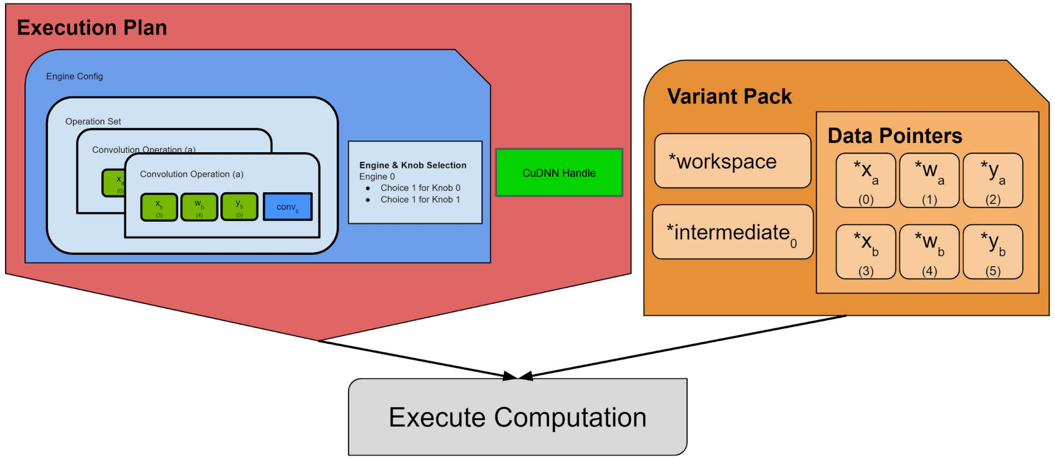
VARIANT PACK

- Lightweight container for data and workspace pointers
- Can change with each iteration of an Execution Plan
- Binds to the execution plan at execution time
- The pointer bindings are done through tensor descriptors' UID
- **Benefits:**
  - Allows re-use of execution plan without the need to copy or double buffer inter training iteration
  - Allows users to economize on execution plans by re-using same execution plan for different layers

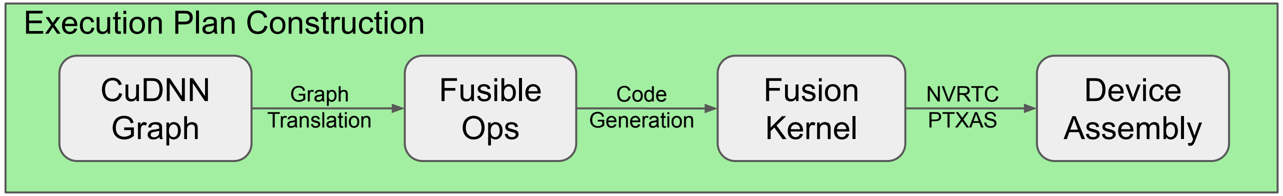


EXECUTE

- Execute the plan
  - Using the engine and performance options from the execution plan
  - Using pointers from the variant pack
  - On the GPU and stream specified by the handle



RUN TIME OP FUSION



- CuDNN op fusion: enabled by an internal Fusible Ops framework built on cutlass
- CuDNN Graph is translated into internal Fusible Operations, which then generate a kernel
- The fused kernel is compiled (via NVRTC), and run, on-the-fly

	Kernel run time (relative)		GPU memory I/O (relative)	
Use case (half in/out, float accumulation)	Legacy API	Graph API	Legacy API	Graph API
conv(FWD) + bias + relu	1	0.31	1	0.33
conv(FWD) + bias + tanh	1.02	0.32	1	0.33

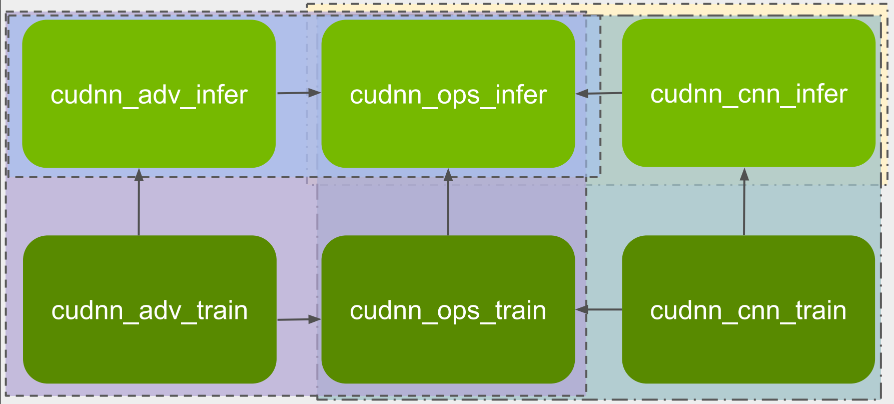
measured on RTX 2080 Ti

LIBRARY SPLIT

- Split the library into multiple sub-libraries based use-cases
- A cross product of training vs. inference and cnn (convolutional neural network) vs. advanced use-cases
- File size for distribution and host/gpu memory utilization in mind
- Compile time linking or lazy loading use-cases are supported
- **Benefits:**
    - For embedded applications one can use only the sub-components that the application requires and reduce file size and memory consumption
    - For frameworks lazy loading would allow applications using the framework to reduce memory footprint based on what the application actually uses.

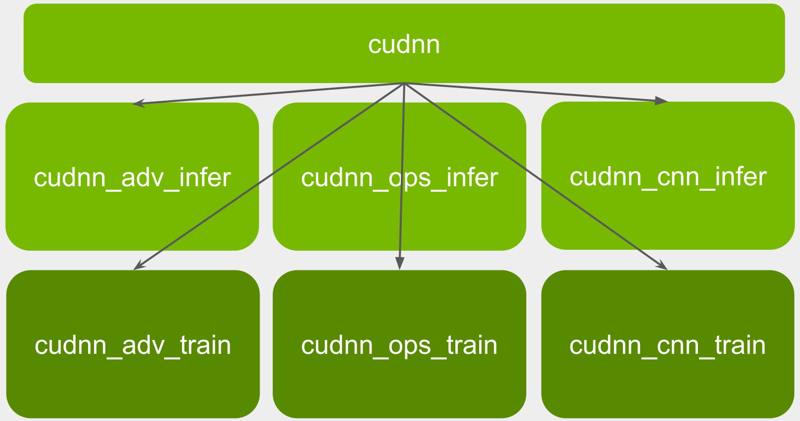
LIBRARY SPLIT -- use case illustration

Static Loading with Dynamic Linkage



- CNN Inference use case
- CNN Training use case
- Advanced Inference use case (e.g. RNNs)
- Advanced Training use case

Dynamic Lazy Loading



- No change in compilation command line
- First-time library load might be slower

Improvements of cudnn v8 on Volta and Turing

- Optimized grouped convolutions and 3D convolutions on Turing and Volta with added new kernels
- Better heuristic selection improves performance across all 2D, 3D and grouped convolutions
- Addressed several long standing bug fixes
- Some notable performance improvements :
    - 3D convs
    - Grouped convs

Call to action

- API Compatibility with 7.6 allows you to drop v8 in your favorite framework and benefit from the performance gains without changing your code
- We urge you to provide feedback as soon as you can
- Using the new API will allow you to tune for performance in your application , we would like to hear your experience with this new approach
- The graph API and runtime fusion capabilities are new features, we want to work with you on improvements on these fronts
- Library split is an acknowledgement that the library size is a concerns , it's not the end of the road , provide us with feedback for future improvements.

Backup

IMPROVED TENSOR DESCRIPTOR

- Goal: Encode symbolic mathematical relationship between tensors in each operation node
- “UID”: use to uniquely identify a tensor
- “isVirtual”: use to indicate whether the tensor exists in memory or needs to be written out to memory

TensorDesc:

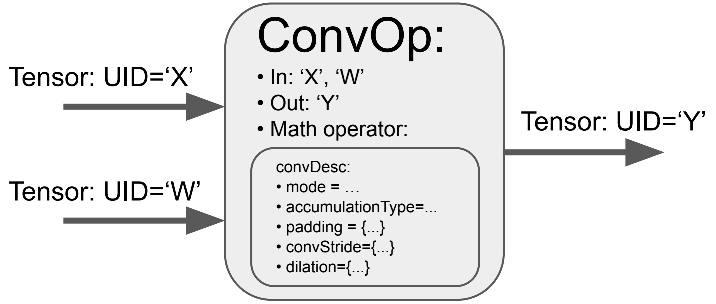
- UID = ‘X’
- isVirtual = false
- dataType = CUDNN_DATA_HALF
- alignment = 16B
- dim = {32, 64, 56, 56}
- stride = {200704, 1, 3584, 64}

Code Sample:

```
cudnnBackendCreateDescriptor(CUDNN_BACKEND_TENSOR_DESCRIPTOR, &Bdesc);

// set type and alignment
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_DATA_TYPE,      CUDNN_TYPE_DATA_TYPE, 1, &dataType);
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_BYTE_ALIGNMENT, CUDNN_TYPE_INT64,     1, &alignment);

// set dims and strides
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_DIMENSIONS, CUDNN_TYPE_INT64, nbDims, dimA);
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_STRIDES,    CUDNN_TYPE_INT64, nbDims, strA);

// set UID and whether it’s virtual
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_IS_VIRTUAL, CUDNN_TYPE_BOOLEAN, 1, &isVirtual);
cudnnBackendSetAttribute(Bdesc, CUDNN_ATTR_TENSOR_UNIQUE_ID,  CUDNN_TYPE_INT64,   1, &uid);

cudnnBackendFinalize(Bdesc);
```

NEW CONCEPT: OPERATION DESCRIPTOR

- Operation descriptor is the elementary unit that is used to express a computation in cuDNN v8
- The basic element for the DAG of a fusion graph
- Each operation node contains information about:
  - I/O: input/output tensors' size, datatype, layout, UID
  - Math: parameters and settings that determine the math equations
- Example operations: convolution, bias , activation.
  - Support for N operators today, more to come in future releases.
- Example attributes: blah, blah

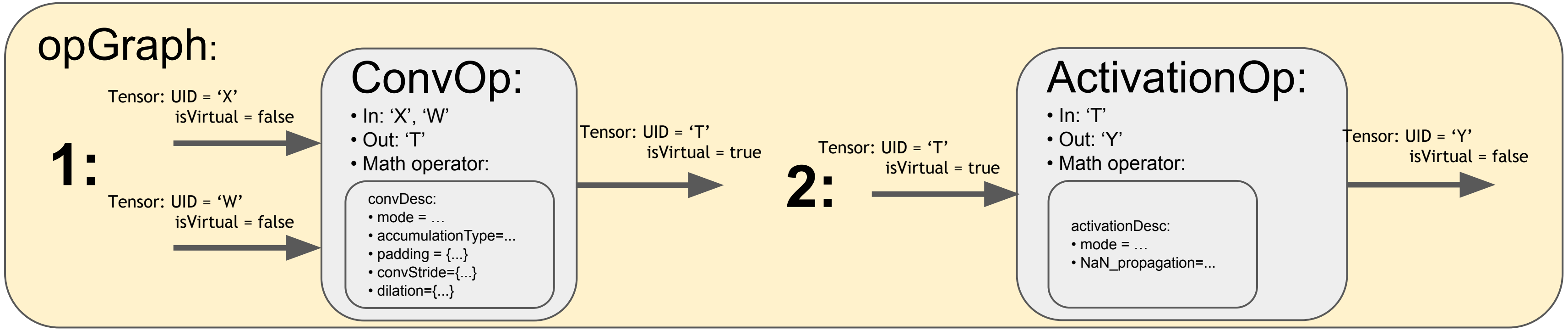


Code Sample:

```
cudnnBackendCreateDescriptor(CUDNN_BACKEND_OPERATION_CONVOLUTION_FORWARD_DESCRIPTOR, &opDesc);

// operator math config
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_CONV_DESC, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, convDesc);

// I/O descriptors
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_X, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, XTensorDesc));
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_W, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, WTensorDesc);
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_Y, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, YTensorDesc);

// blending coefficients
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_ALPHA, CUDNN_TYPE_DOUBLE, 1, &alpha);
cudnnBackendSetAttribute(opDesc, CUDNN_ATTR_OPERATION_CONVOLUTION_FORWARD_BETA,  CUDNN_TYPE_DOUBLE, 1, &beta );

cudnnBackendFinalize(opDesc);
```

NEW CONCEPT: OPERATION GRAPHS

- An array of operation nodes representing a computation sub-graph
- Uses Tensor UIDs to deduce the data-flow of the computation sub-graph



Code Sample:

```
cudnnBackendCreateDescriptor(CUDNN_BACKEND_OPERATION_SET_DESCRIPTOR, &opSetDesc);

cudnnBackendSetAttribute(opSetDesc, CUDNN_ATTR_OPERATION_SET_HANDLE, CUDNN_TYPE_HANDLE, 1, &handle);
cudnnBackendSetAttribute(opSetDesc, CUDNN_ATTR_OPERATION_SET_OPS, CUDNN_TYPE_OPERATION, numOps, ops);

cudnnBackendFinalize(opSetDesc);
```

$\Leftarrow$ Validation and graph construction

QUERYABLE ENGINE ATTRIBUTES

cuDNN v7.6 allows users to query determinism for a given algorithm.
cuDNN v8.0 allows users to query all numerical notes, knobs and other attributes for a given EngineConfig.

Benefit: Users are able to query potentially important information which was previously hidden.

Code Sample

```
// Query For Numerical Notes From Engine
cudnnBackendNumericalNote_t notes[10];
cudnnBackendGetAttribute(engine, CUDNN_ATTR_ENGINE_NUMERICAL_NOTE, CUDNN_TYPE_NUMERICAL_NOTE, 10, &noteCount, &numNotes);

// Check For Non-Deteriminism Note
for(int noteIdx = 0; noteIdx < noteCount; noteIdx++) {
    if(numNotes[noteIdx] == CUDNN_NUMERICAL_NOTE_NONDETERMINISTIC) {
        // Engine will be non-deterministic
    }
}
```

NEW HEURISTIC API

cuDNN v7.6 heuristics map a given Computation Case to an ideal algorithm.
cuDNN v8.0 heuristics map a given Computation Case to an ideal EngineConfig.

Benefit: Users can query Numerics, Knob Choices, Alignment from the EngineConfig.

Many kernels to choose from. Ranks engines according to performance. Can override.

Code Sample

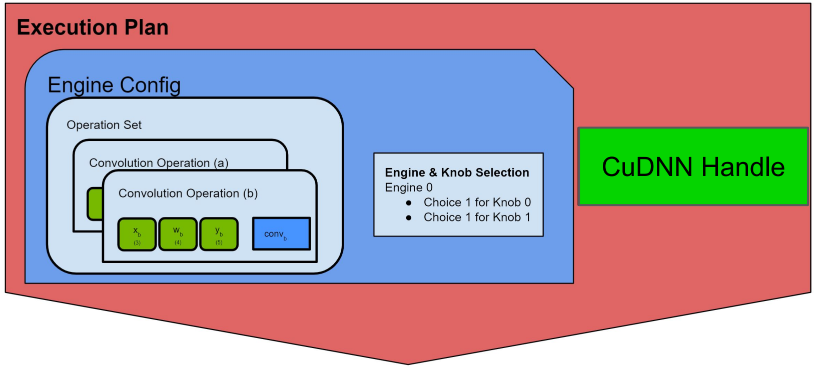
```
// Create & Initialize EngineHeuristics Descriptor
cudnnBackendCreateDescriptor(CUDNN_BACKEND_ENGINEHEUR_DESCRIPTOR, &engHeur);
cudnnBackendSetAttribute(engHeur, CUDNN_ATTR_ENGINEHEUR_OPERATION_SET, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, opSetDesc);
cudnnBackendSetAttribute(engHeur, CUDNN_ATTR_ENGINEHEUR_MODE, CUDNN_TYPE_HEUR_MODE, 1, &heurMode);
cudnnBackendFinalize(engHeur);

// Create EngineConfig Descriptor
cudnnBackendCreateDescriptor(CUDNN_BACKEND_ENGINE_CONFIG_DESCRIPTOR, &engConfig));

// Retrieve Optimal EngineConfig Descriptor From Heuristic
//  *Users can change “1” to any number of results they would like to retrieve
cudnnBackendGetAttribute(engHeur, CUDNN_ATTR_ENGINEHEUR_RESULTS, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, &returnedCount,
engConfig);
```

NEW CONCEPT: NON-EAGER EXECUTION

Separate APIs for build and execute stages

Amortise CPU overhead -- runtime compilation, querying heuristics, searching kernel data-base, etc.

Create once use multiple times

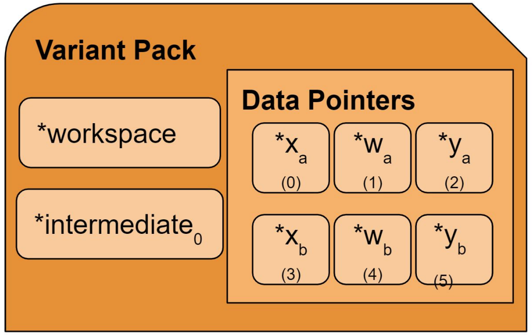


Code Sample:

```
// create execution plan
cudnnBackendCreateDescriptor(CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR, &execPlan);

// set handle and engine config
cudnnBackendSetAttribute(execPlan, CUDNN_ATTR_EXECUTION_PLAN_HANDLE, CUDNN_TYPE_HANDLE, 1, &handle);
cudnnBackendSetAttribute(execPlan, CUDNN_ATTR_EXECUTION_PLAN_ENGINE_CONFIG, CUDNN_TYPE_BACKEND_DESCRIPTOR, 1, engConfig);

// execution plan validation
cudnnBackendFinalize(execPlan);
```

MAKE VARIANT PACK

- Lightweight container for data and workspace pointers for an Execution Plan
- May change every iteration, binds to the execution plan at execution time
- The pointer bindings are done through tensor descriptors' UID

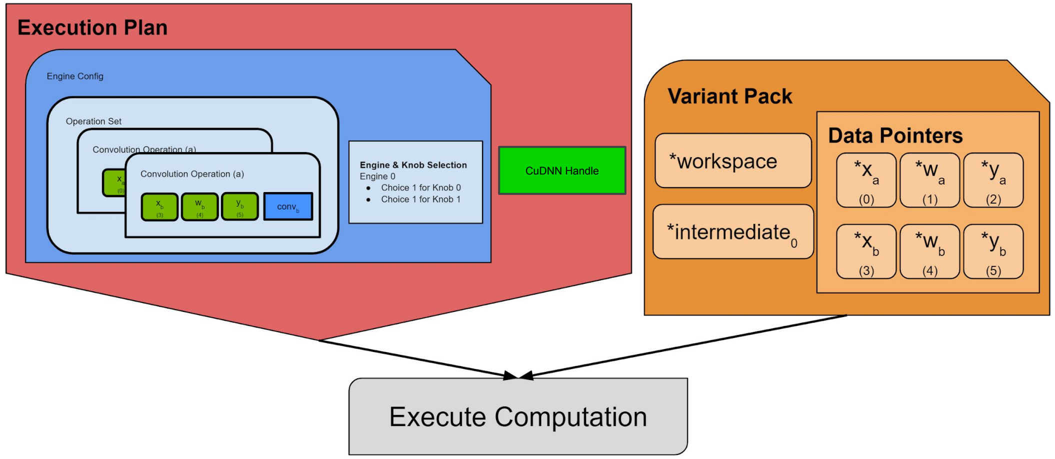


Code Sample:

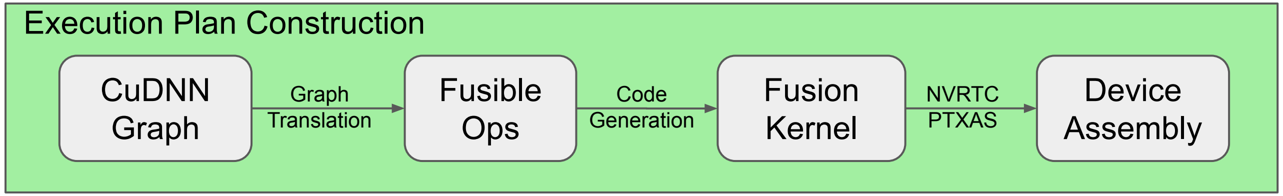
```
// create the variant pack descriptor
cudnnBackendCreateDescriptor(CUDNN_BACKEND_VARIANT_PACK_DESCRIPTOR, &varPack);

// gather device UIDs and pointers
int64_t uids[]  = {xTensor.getUid(),    wTensor.getUid(),    yTensor.getUid()};
void *devPtrs[] = {xTensor.getDevPtr(), wTensor.getDevPtr(), yTensor.getDevPtr()};

// fill up the variant pack
cudnnBackendSetAttribute(varPack, CUDNN_ATTR_VARIANT_PACK_UNIQUE_IDS,    CUDNN_TYPE_INT64,    COUNTOF(uids),    uids);
cudnnBackendSetAttribute(varPack, CUDNN_ATTR_VARIANT_PACK_DATA_POINTERS, CUDNN_TYPE_VOID_PTR, COUNTOF(devPtrs), devPtrs);
cudnnBackendSetAttribute(varPack, CUDNN_ATTR_VARIANT_PACK_WORKSPACE,     CUDNN_TYPE_VOID_PTR, 1,                workspace);

// variant pack validation
cudnnBackendFinalize(varPack);
```

EXECUTE

- Execute the plan
  - Using the engine and performance options from the execution plan
  - Using pointers from the variant pack
  - On the GPU and stream specified by the handle



RUN TIME OP FUSION



- CuDNN op fusion: enabled by an internal Fusible Ops API, built on cutlass
- CuDNN Graph is translated into Fusible Ops, which then generate Cutlass fusion kernel
- Today's demo shows various fusions; basically five variations of conv-bias-activation fusion (bias/no bias, activation/no activation, using either relu or tanh activation)
- The various fusions are generated, compiled (via NVRTC), and run, on-the-fly (note the generated cutlass code)
- Note that the runs pass functionally, and perf is on par with well optimized off-line compiled convolution kernels.
- Today's runs are all in fp32 compute with fp16 I/O, utilizing Tensor Cores on Turing architecture