nGraph + PlaidML

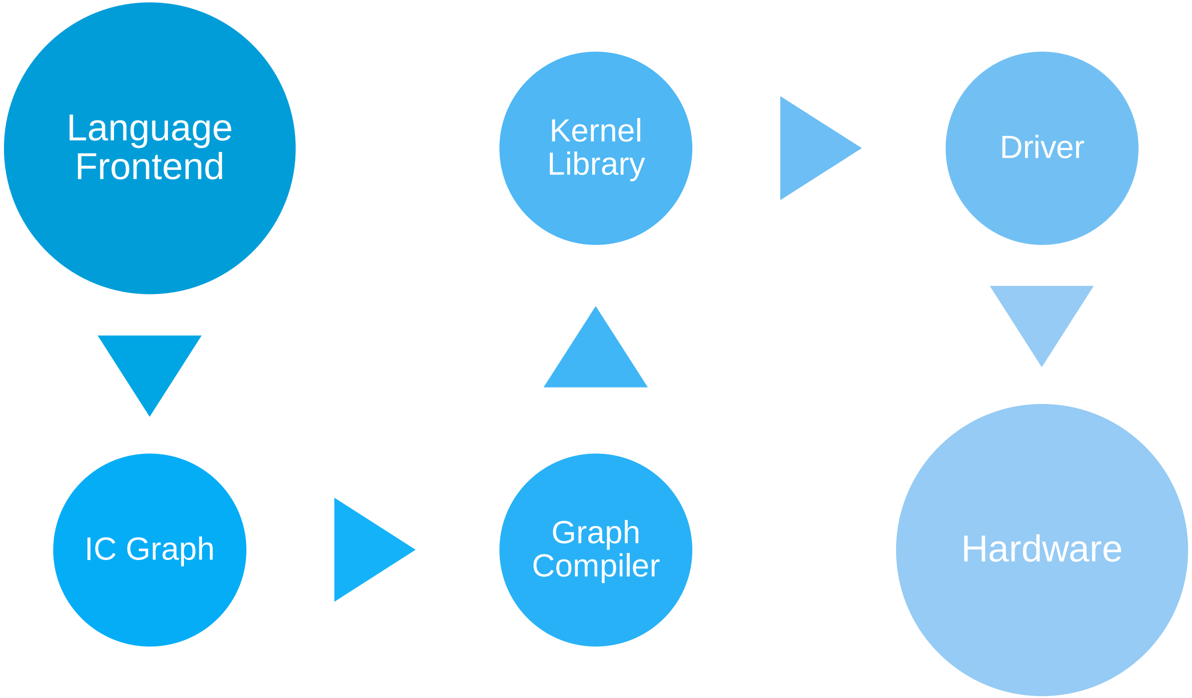
Unlocking Next-Generation Performance with Deep Learning Compilers

Jayaram Bobba and Tim Zerrell

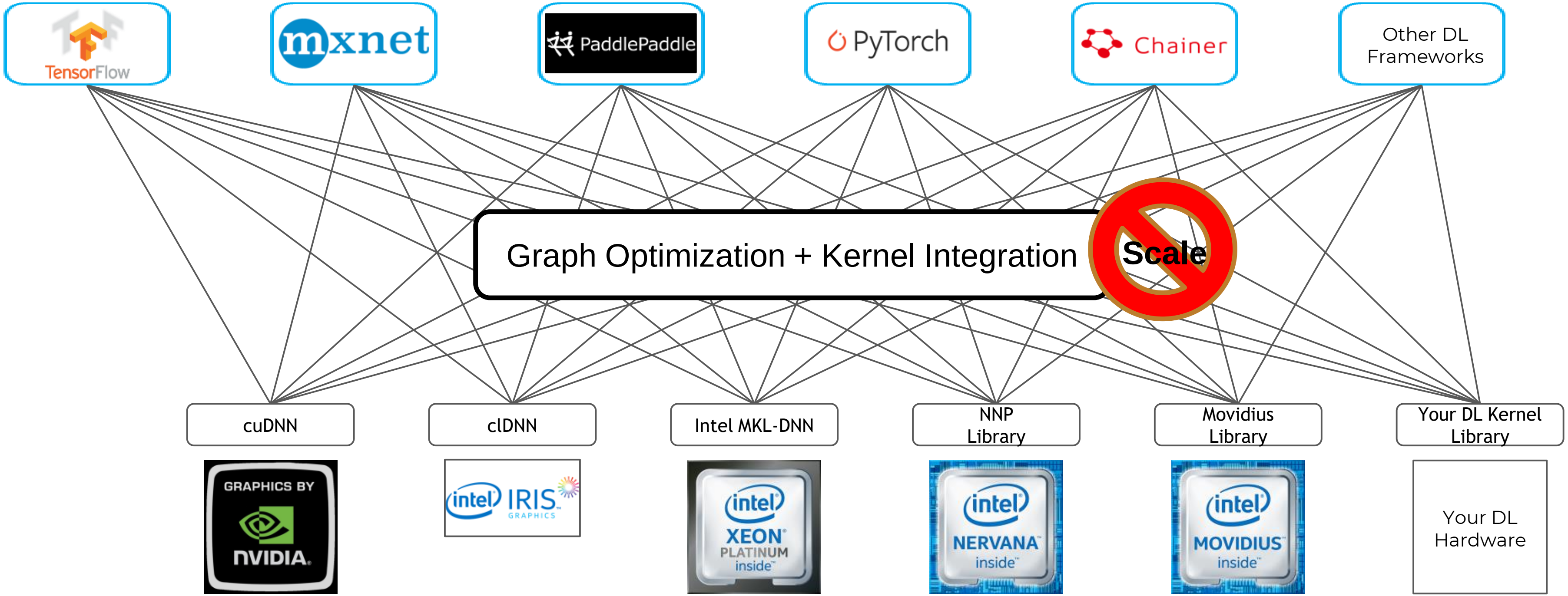
Legal Notices & Disclaimers

The Path to nGraph+PlaidML

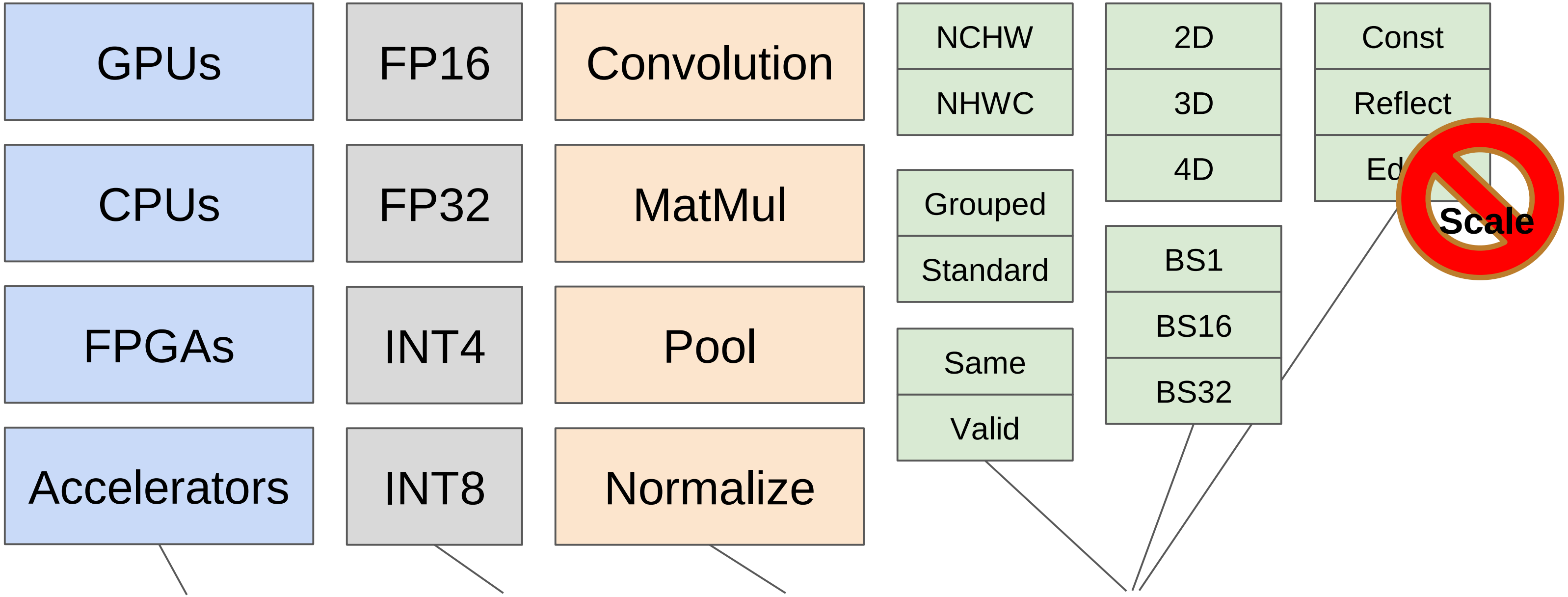
Simplified Deep Learning Stack



Current State of DL framework acceleration: Framework Optimization

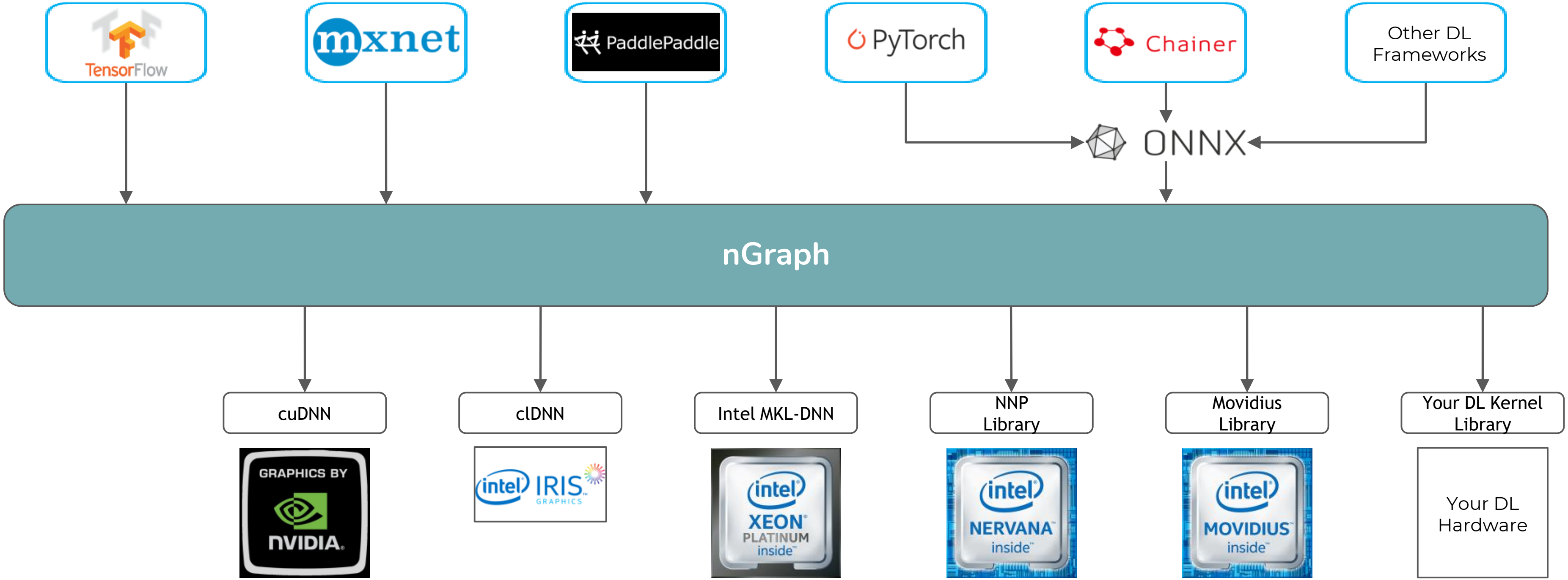


Current State of DL framework acceleration: Kernel Libraries



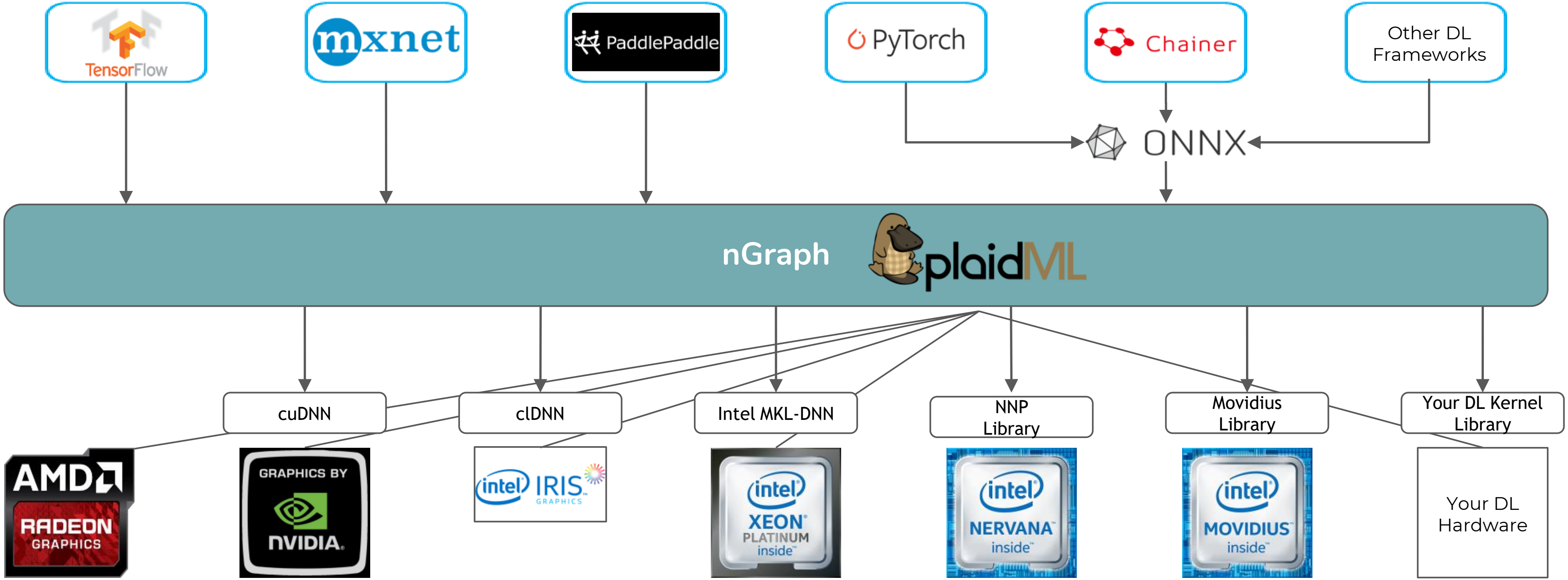
#ChipDesigns * #DTypes * #Ops * ∏(#Params) = #Kernels

Our Solution: **nGraph + PlaidML**



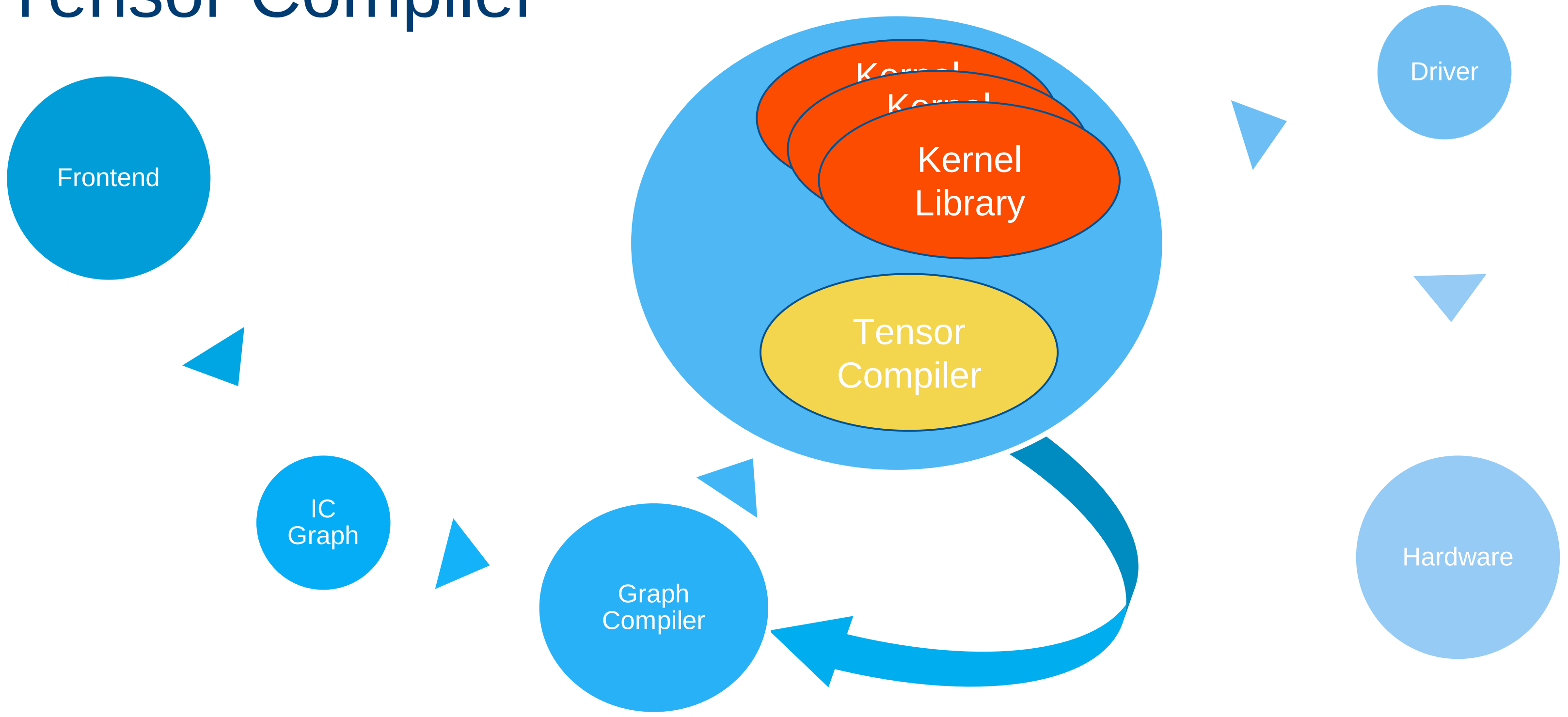
Graph level optimizations + Kernel Library Integration

Our Solution: **nGraph + PlaidML**



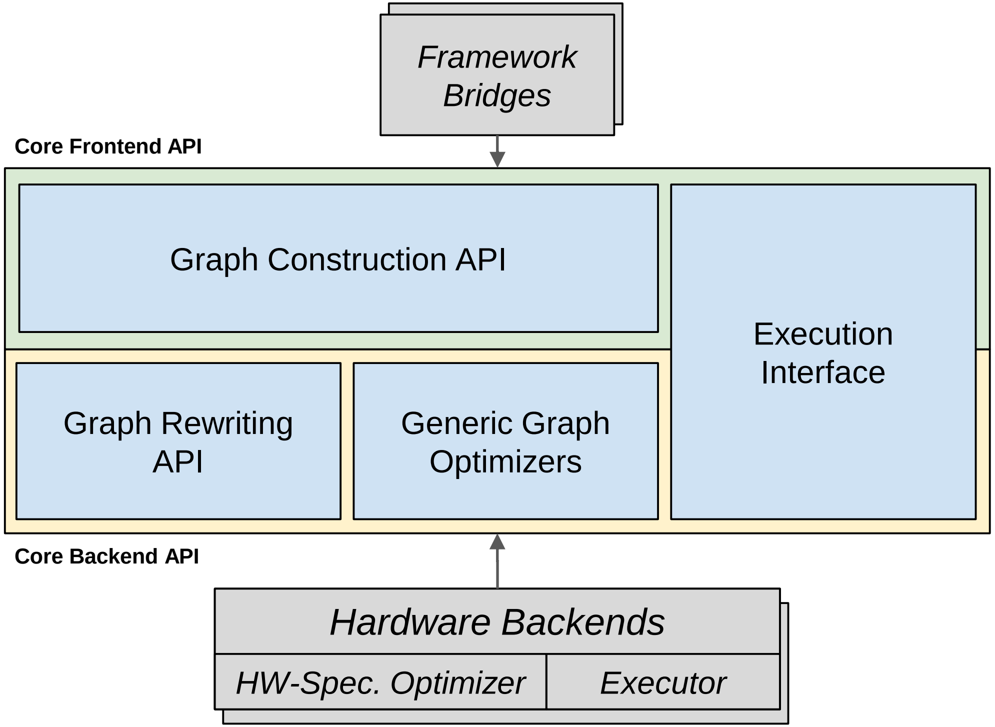
Graph level optimizations + Kernel Library Integration + Tensor Compiler

nGraph + PlaidML: A Multi-Platform Stack w/ Tensor Compiler



nGraph: A Deep Dive

The Whole Stack: **Bridges, Core**, Backends



Framework Bridges

TensorFlow Bridge

https://github.com/NervanaSystems/ngraph-tf

Option 1 (pre-built binaries)

1) pip install tensorflow
2) pip install ngraph-tensorflow-bridge
3) import ngraph_bridge

Option 2 (from source)

1) Download tensorflow v1.12.0
2) bazel build –config=opt –config=ngraph //tensorflow/tools/pip_package:build_pip_package

Mxnet Bridge

https://github.com/NervanaSystems/ngraph-mxnet

Option 1 (pre-built binaries)

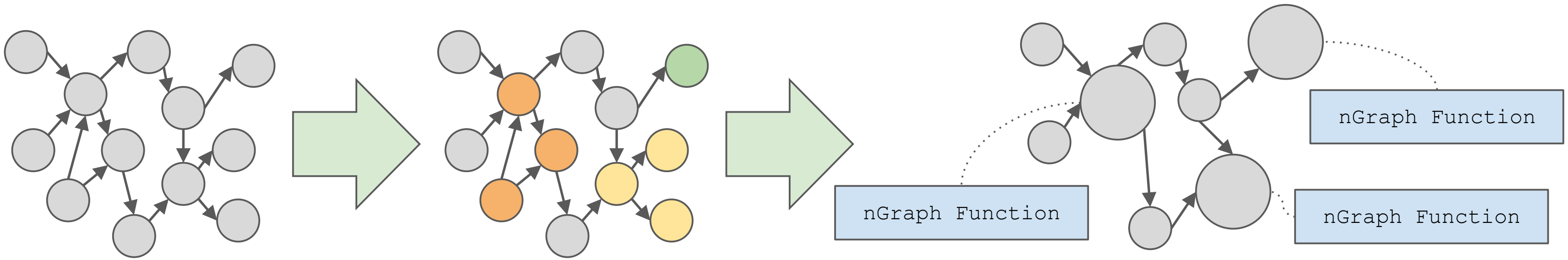
1) pip install ngraph-mxnet

Option 2 (from source)

1) Download ngraph-mxnet
2) Make USE_NGRAPH=1

●●●

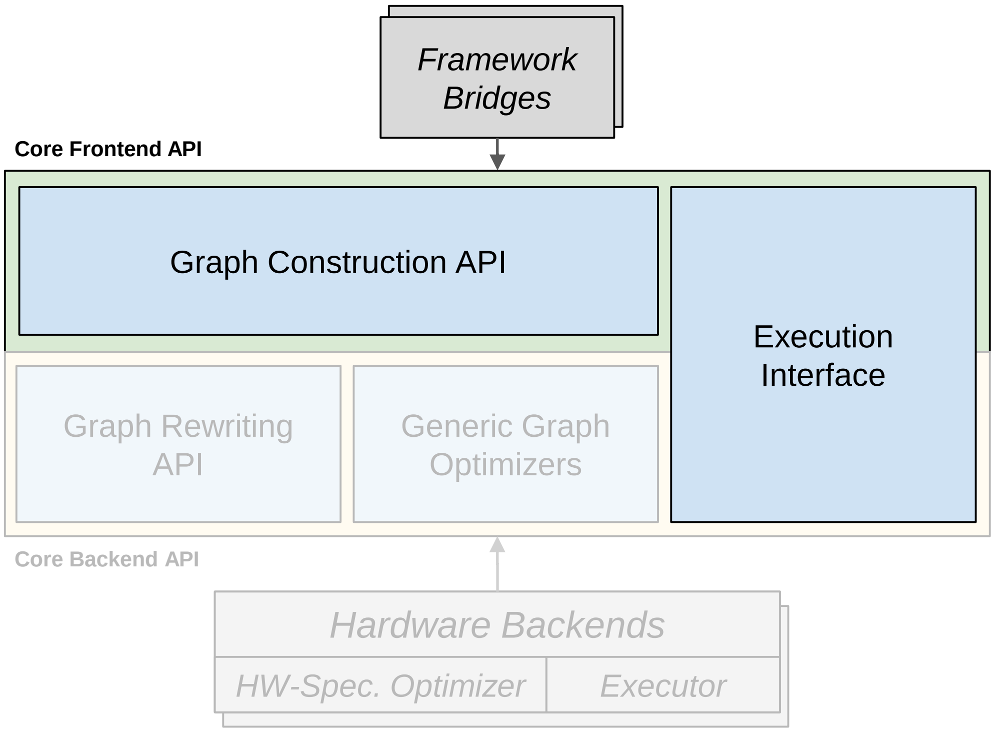
Framework Bridge: Translation Flow



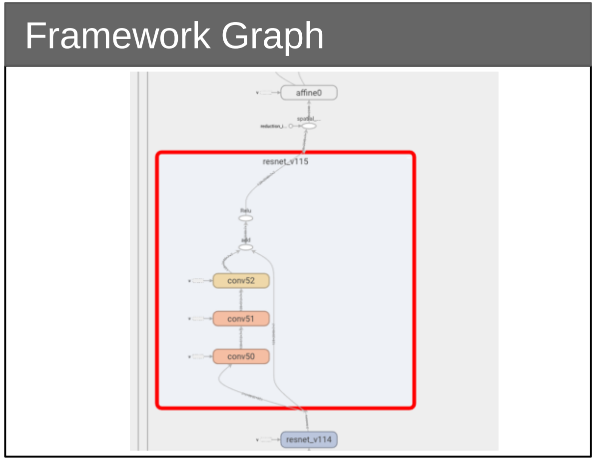
Original FW Graph | After **Clustering** | After **Translation**

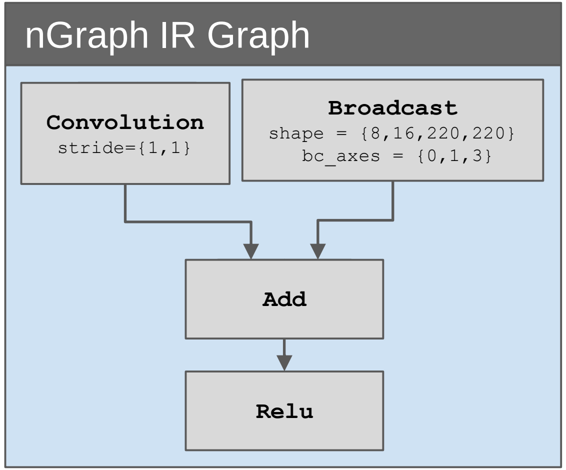
- Each cluster is a “unit of work” for nGraph.
- Anything not clustered stays on the framework native engine.
- Backend has **freedom to rewrite** nGraph `Function`s
  - …for optimization
  - …for easy integration with kernel libraries
  - etc.

Framework Bridges -> nGraph



Graph Construction API:
From Framework Graphs to nGraph IR





- **Rich op sets** (TF: ~1K ops[1])
- Usually **dynamically typed**
- **"Non-DL" ops**

- **Small set of simple ops**
- **Statically typed**
- **Focused on DL primitives**

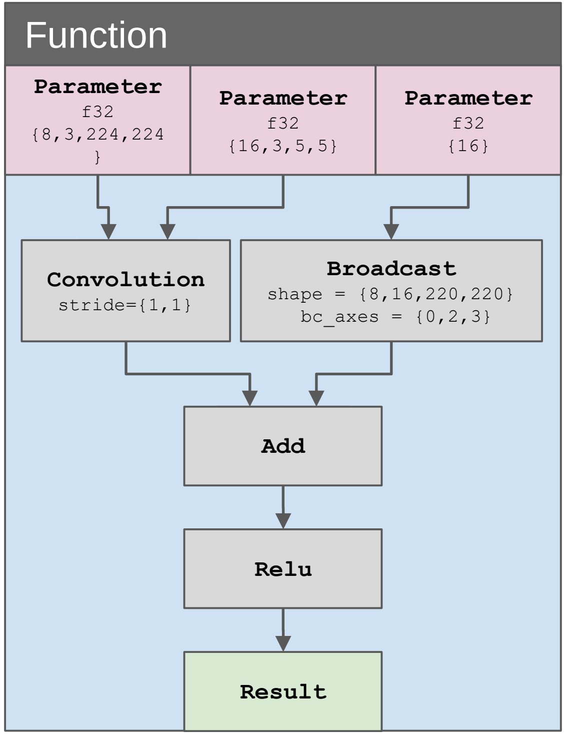
[1] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/ops/ops.pbtxt

Constructing Graphs

```
auto data_in = make_shared<op::Parameter>(element::f32, Shape{8,3,224,224});
auto w_in = make_shared<op::Parameter>(element::f32, Shape{16,3,5,5});
auto b_in = make_shared<op::Parameter>(element::f32, Shape{16});

auto conv = make_shared<op::Convolution>(data_in, w_in, Strides{1,1});
auto bias_bc = make_shared<op::Broadcast>(b_in, Shape{8,16,220,220},
                                          AxisSet{0,2,3});
auto conv_bias = make_shared<op::Add>(conv, bias_bc);
auto conv_bias_relu = make_shared<op::Relu>(conv_bias);

auto f = make_shared<Function>(conv_bias_relu,
                               ParameterVector{data_in, w_in, b_in});
```



nGraph Code

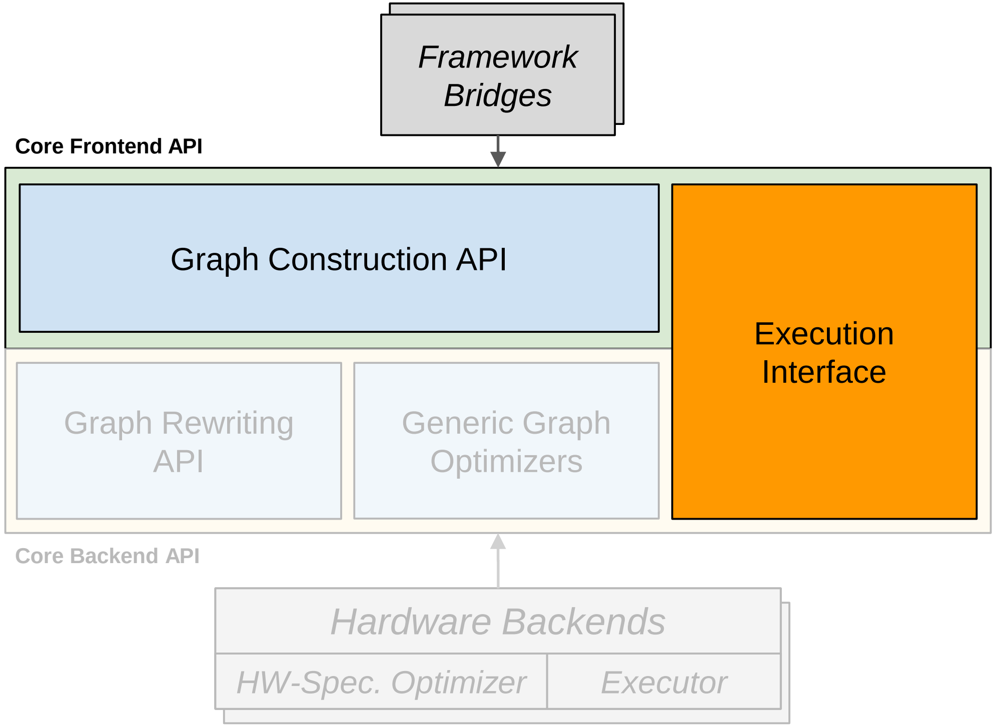
Branch: master ▾ ngraph / src / ngraph /

Create new file | Upload files | Find file | History

rkimballn1 and diyessi Backend API change pre-work (#2064) ... Latest commit e093355 4 hours ago

..			
autodiff	Update add_delta_to_slice to tolerate dynamic shapes (#1962)	a month ago	automatic graph differentiation
builder	QCBiasAdd and QCBiasSignedAdd for mkldnn (#2062)	20 hours ago	
codegen	remove version tagging from shared libraries (#2094)	16 days ago	
descriptor	Cyphers/bnorm back (#2129)	9 days ago	
frontend	Backend API change pre-work (#2064)	4 hours ago	Python, ONNX, ONNXIFI frontends
op	QCBiasAdd and QCBiasSignedAdd for mkldnn (#2062)	20 hours ago	nGraph Core Ops
pass	an env var to disable individual fusions (#2185)	a day ago	
pattern	Abort messages in Matcher to better understand cases where we fail to...	a day ago	
runtime	Backend API change pre-work (#2064)	4 hours ago	
state	Fix warnings in RNGState for clang on macosx (#1968)	a month ago	
type	Quantize(reorder) bias to int32 (#1933)	25 days ago	
CMakeLists.txt	On macos, (#2121)	10 days ago	
assertion.hpp	Cannot end a single line c++ commend with a backslash. (#1651)	3 months ago	

Execution Interface: Run graphs

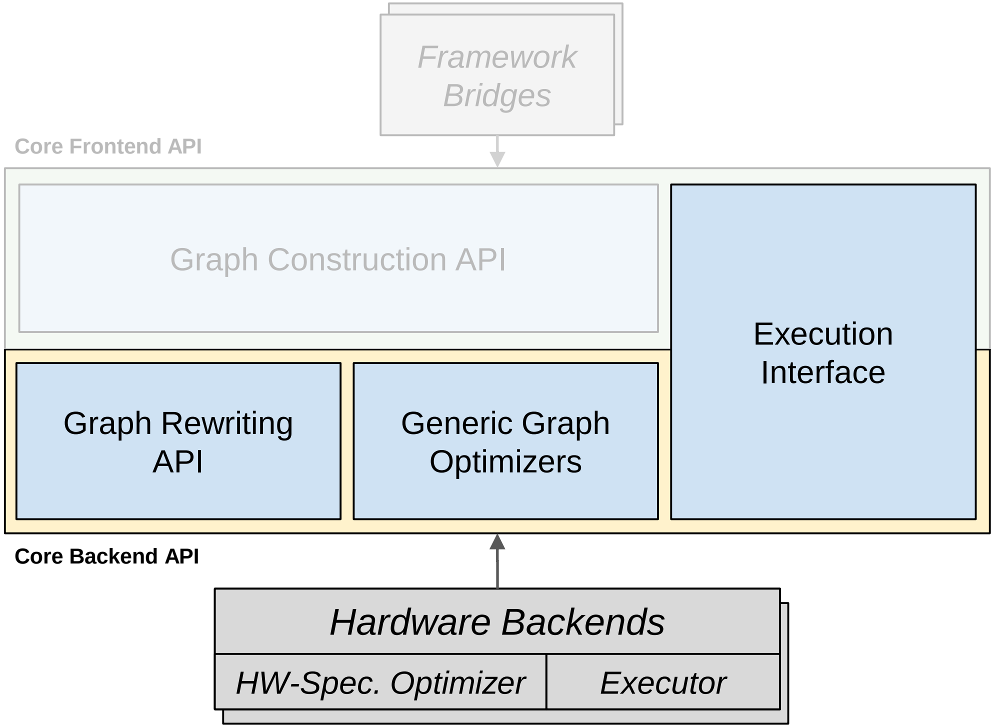


- **Execution API** is a simple four-method interface.
  - **create_tensor()**
  - **write()**
  - **read()**
  - **compile()**
  - **call()**

- These functions are **implemented by each backend**.

- NB: write(), read() can be avoided for host-resident tensors.

The Whole Stack: Bridges, **Core, Backends**



Hardware Backends

Branch: master ▾ ngraph / src / ngraph / runtime /

Create new file | Upload files | Find file | History

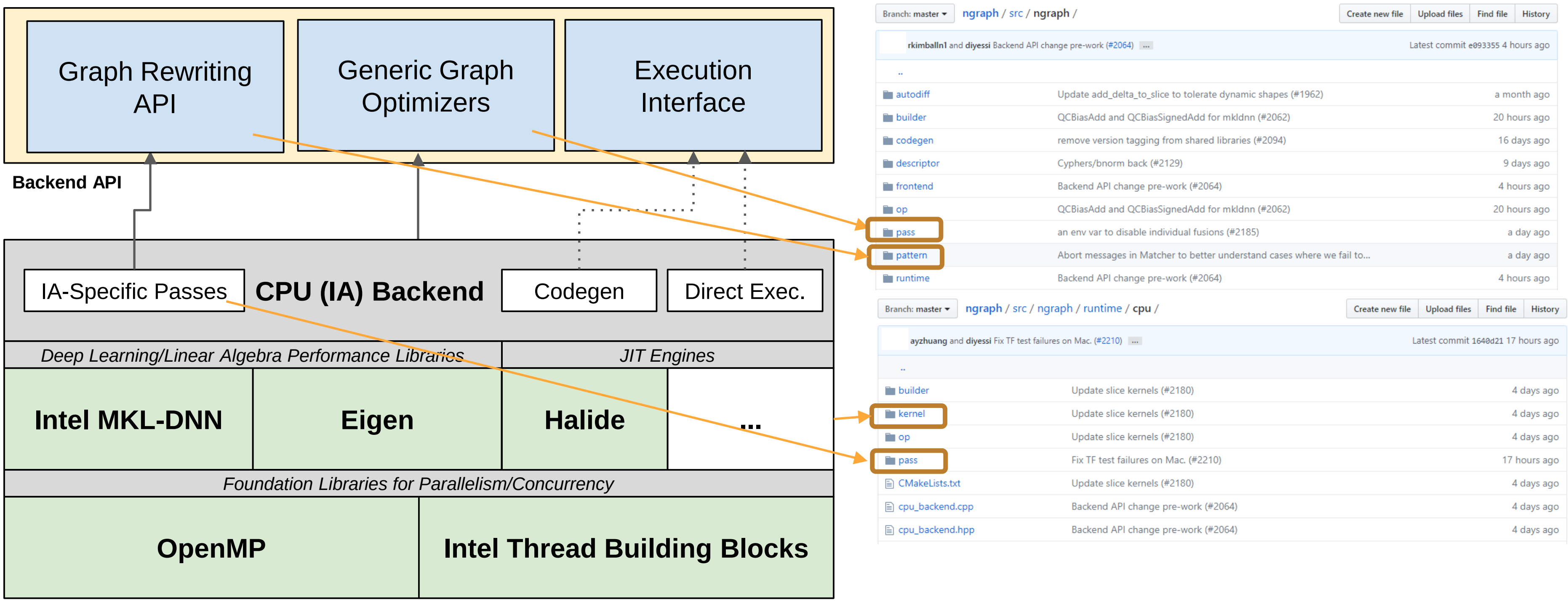
jbobba and rkimballn1 Update slice kernels (#2180) … Latest commit a16c496 13 minutes ago

..		
cpu	Update slice kernels (#2180)	13 minutes ago
gpu	Backend API change pre-work (#2064)	4 hours ago
hybrid	Backend API change pre-work (#2064)	4 hours ago
intelgpu	Backend API change pre-work (#2064)	4 hours ago
interpreter	Backend API change pre-work (#2064)	4 hours ago
nop	Backend API change pre-work (#2064)	4 hours ago
plaidml	Minor perf tweaks (#2095)	7 days ago
reference	fix to kahan summation in ref kernel (#2140)	4 days ago
CMakeLists.txt	NOP backend (#1979)	a month ago

More in external repos for new hardware and new usage models

* Other brands and names may be claimed as the property of others.

Example: Intel CPU Backend



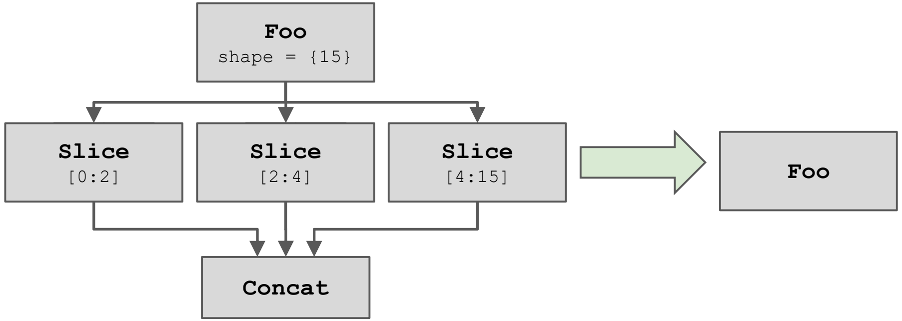
Generic Graph Optimizers: Optimization Passes

- Pass manager makes it easy to reuse and mix **generic optimization passes**, and your own **device-specific optimizations**.

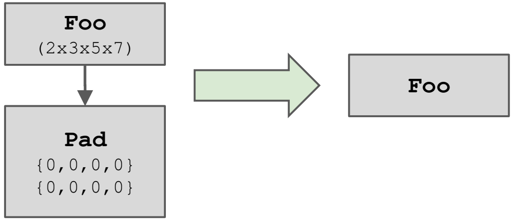
- Same, **unified interface and APIs** for both.

- nGraph Core includes a large library of **HW-agnostic passes**:
    - Algebraic Simplification
    - Common Subexpression Elimination
    - Constant Folding
    - Core Fusion
    - Reshape/Transpose Elimination
    - Reshape/Transpose Sinking
    - Zero-Element Tensor Elimination

Optimization Passes: Algebraic Simplification

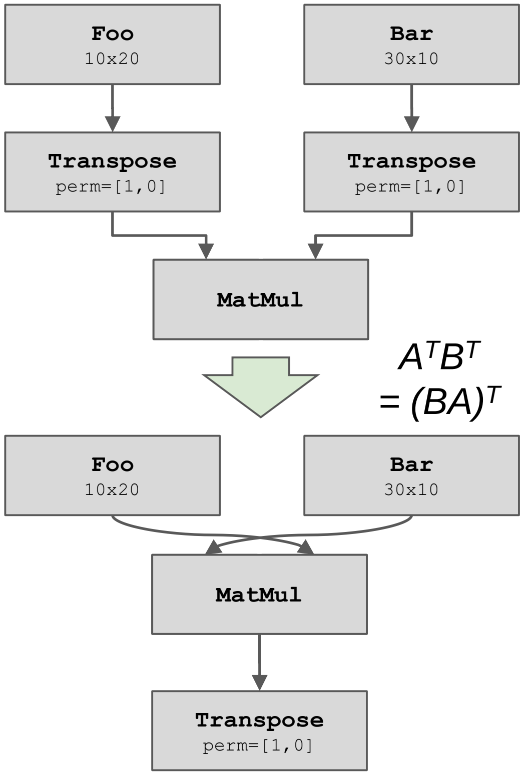


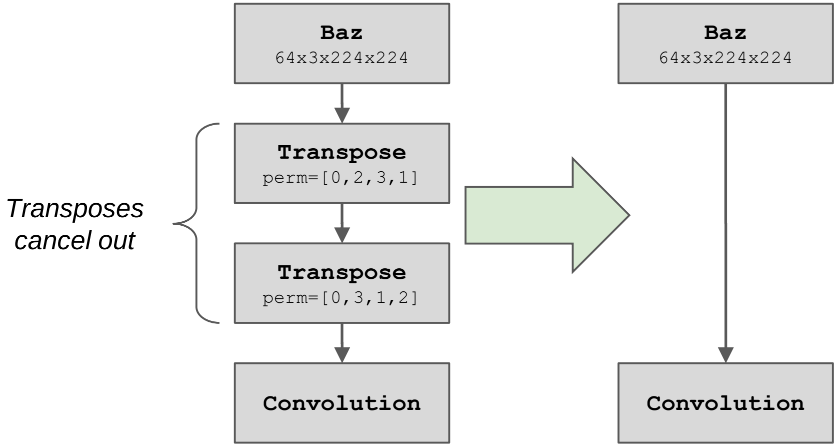
Tensor is being sliced up into pieces and immediately being reassembled



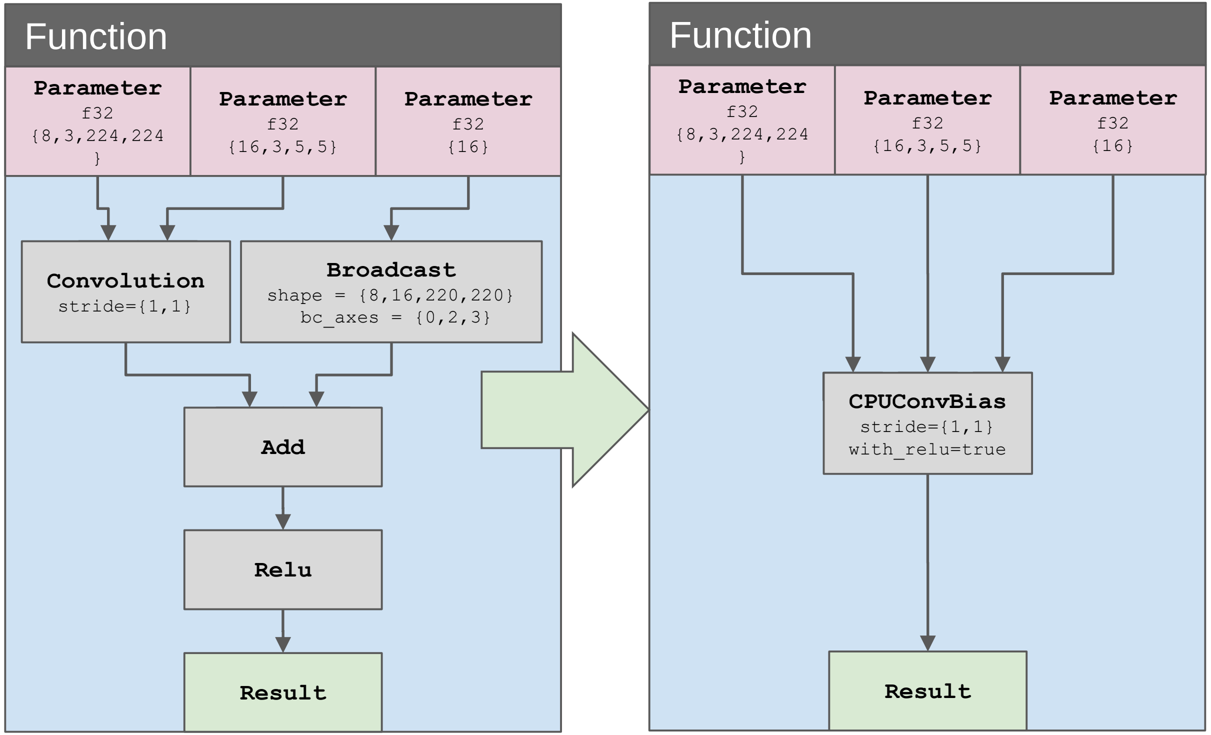
Tensor is being "padded" but the width of padding is zero all around

Optimization Passes: Reshape/Transpose Elimination





Pattern Matching & Graph Rewriting



Step 1: Describe pattern

```
auto data_batch = std::make_shared<pattern::op::Label>(element::f32, shape);
auto filters = std::make_shared<pattern::op::Label>(element::f32, shape);
auto pbias = std::make_shared<pattern::op::Label>(element::f32, Shape{});

auto pbroadcast = std::make_shared<op::Broadcast>(pbias, shape, AxisSet{0, 1, 2, 3});

auto pconv1 = std::make_shared<op::Convolution>(data_batch,
                                                filters,
                                                Strides{1, 1},
                                                Strides{1, 1},
                                                CoordinateDiff{0, 0},
                                                CoordinateDiff{0, 0},
                                                Strides{1, 1});
auto p_conv_bias = pbroadcast + pconv1;
```

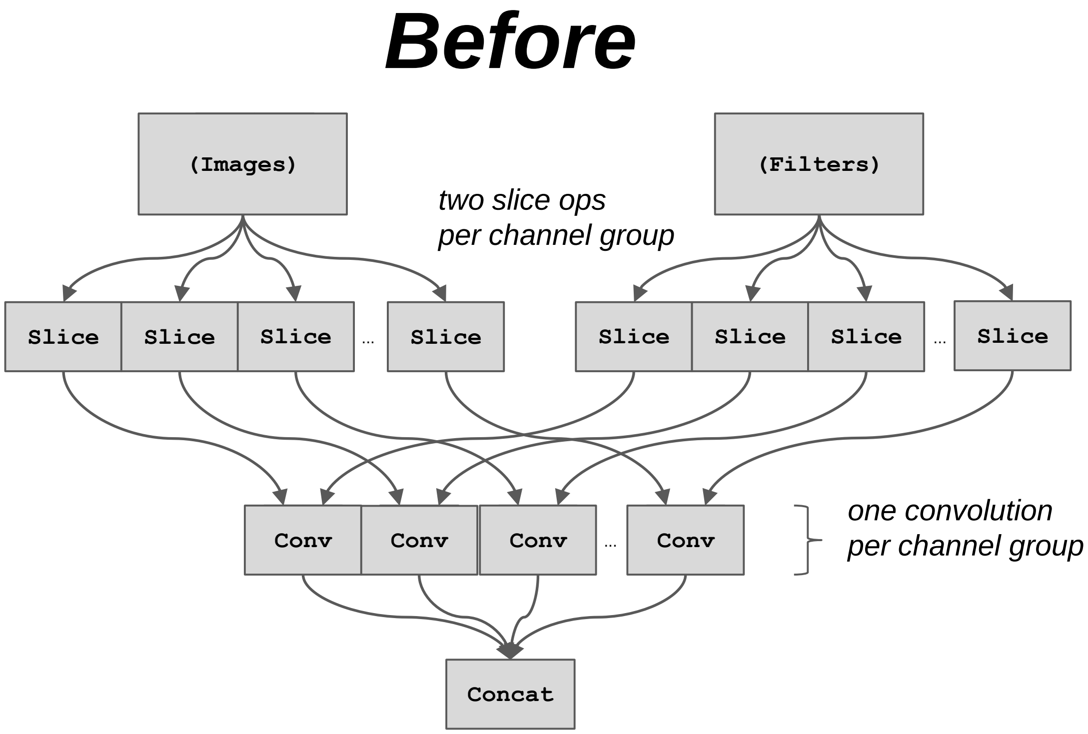
Step 2: Request pattern match

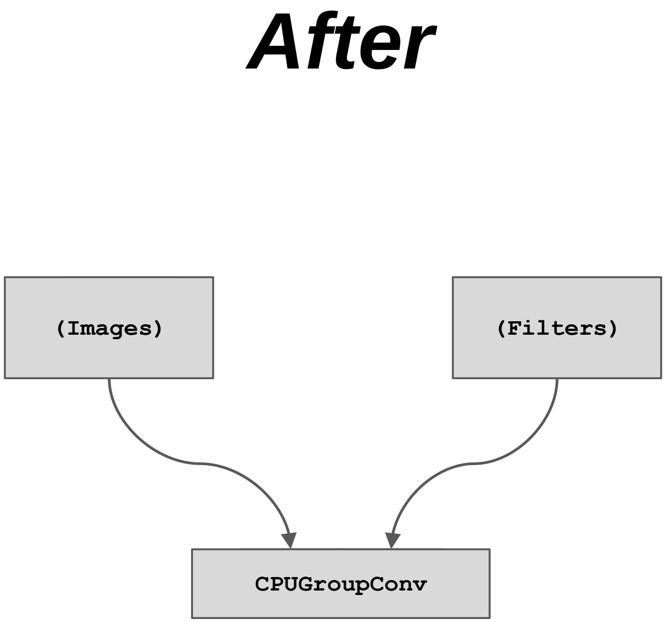
```
auto m =
    std::make_shared<ngraph::pattern::Matcher>(p_conv_bias, callback, "CPUFusion.ConvBias");
this->add_matcher(m);
```

Step 3: Rewrite match

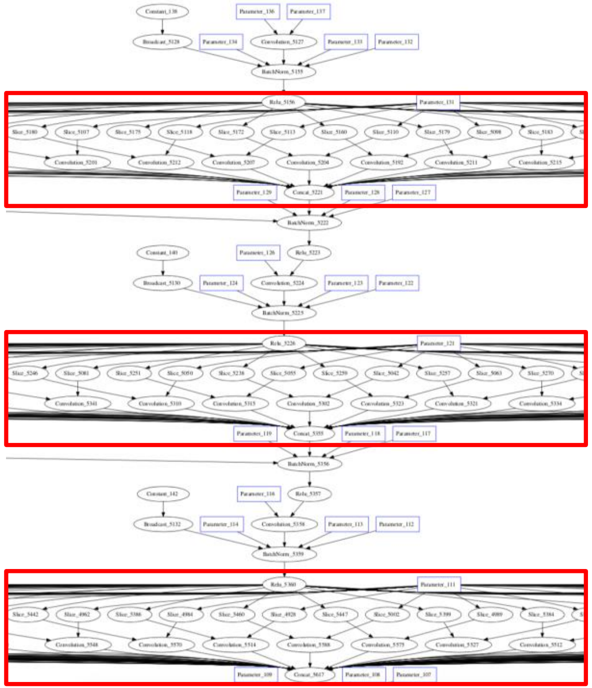
```
auto conv_bias = std::shared_ptr<Node>(new op::ConvolutionBias(conv, bias));
ngraph::replace_node(m.get_match_root(), conv_bias);
```

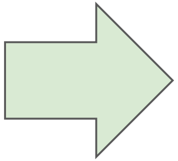
Backend Specific Opt: Group Convolution Fusion

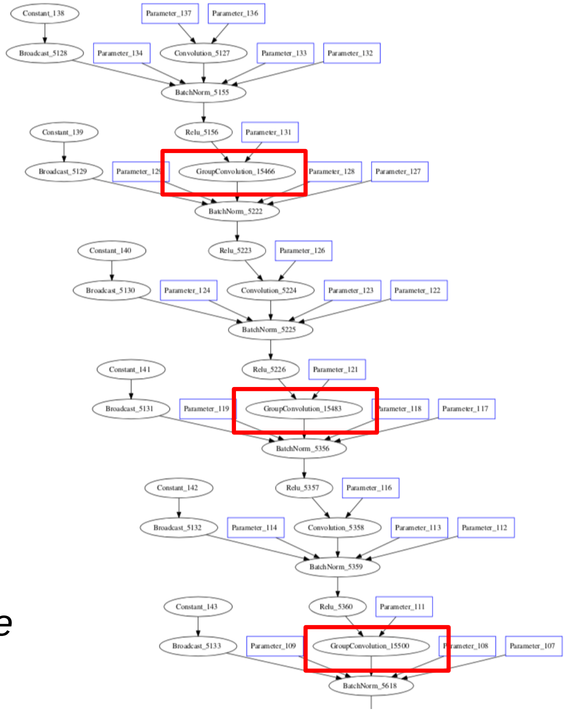




Example: MobileNet after Group Convolution Fusion

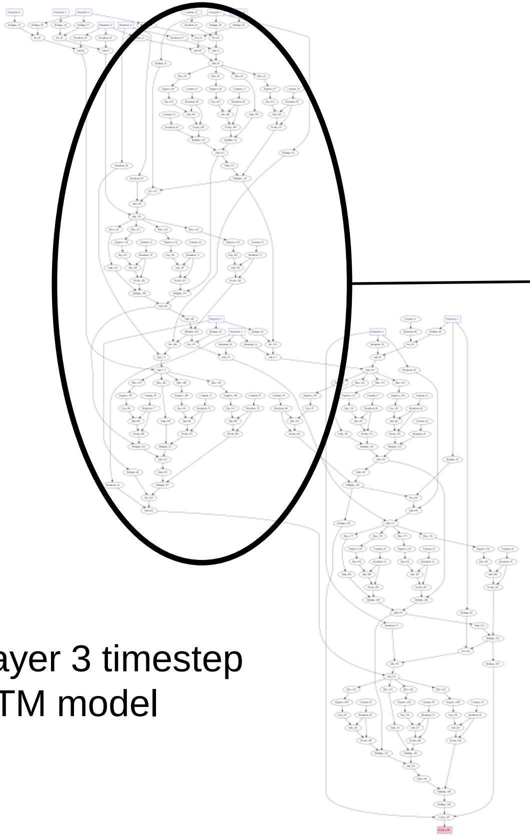




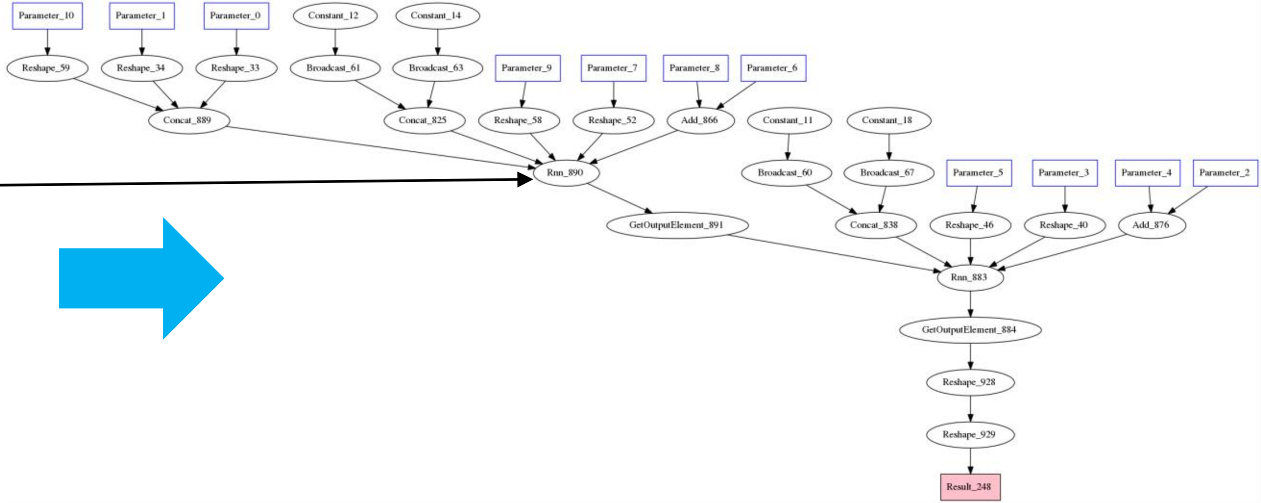


(Rectangles at left are actually way too wide to fit on the slide…)

Backend Specific Opt: RNN fusion



2-layer 3 timestep
LSTM model

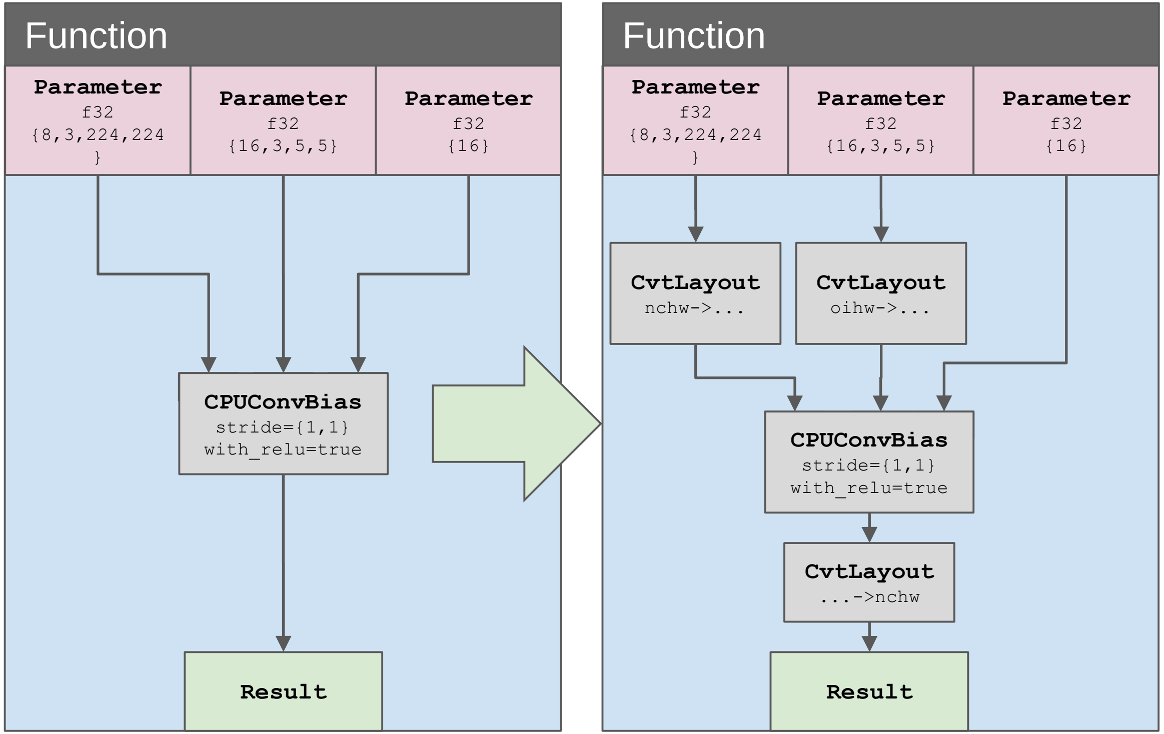


Recurrent matcher captures RNNs with arbitrary number of timesteps

Backend Specific Opt: Layout Assignment

- **Logically**, nGraph always uses "NCHW/OIHW" format.
- **Physically**, the backend has control of layout.

- CPU backend selects among layouts supported by Intel MKL-DNN.
    - `Oihw`
    - `OIhw4i16o4i_s8s8`
    - Many, many others
- Good choices here are critical to performance.



Registering and Running Optimization Passes

```
pass_manager.register_pass<pass::NopElimination>();
pass_manager.register_pass<pass::ZeroDimTensorElimination>();
pass_manager.register_pass<pass::AlgebraicSimplification>();
pass_manager.register_pass<cpu::pass::CPURnnMatFusion>();
pass_manager.register_pass<cpu::pass::CPUBatchFusion>();
pass_manager.register_pass<pass::CoreFusion>();
pass_manager.register_pass<cpu::pass::CPUFusion>();
pass_manager.register_pass<pass::ConstantFolding>();
pass_manager.register_pass<cpu::pass::CPULayout>(...);
pass_manager.register_pass<pass::CommonSubexpressionElimination>(...);
    ...
auto optimized_graph = pass_manager.run_passes(original_graph);
```

- Pass manager makes it easy to reuse and mix **generic optimization passes**, and your own **device-specific optimizations**.
- Example at left from Intel CPU backend. *(Abbreviated)*

nGraph Hands-on

Nasnet through TensorFlow and nGraph

Setup

Intel® Xeon® Scalable Processor, Ubuntu 16.04

- Install Tensorflow and ngraph-tensorflow-bridge

```
jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ virtualenv --system-site-packages .venv
Using base prefix '/usr'
New python executable in /localdisk/jbobba/nasnet/.venv/bin/python3
Also creating executable in /localdisk/jbobba/nasnet/.venv/bin/python
Installing setuptools, pip, wheel...done.
jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ . .venv/bin/activate
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ pip install tensorflow==v1.12
Collecting tensorflow==v1.12
```

```
Successfully installed absl-py-0.6.1 astor-0.7.1 gast-0.2.0 grpcio-1.17.0 h5py-2.8.0 keras-applications-1.0.6 keras-preprocessing-1.0.5 markdown-3.0.1 protobuf-3.6.1 tensorboard-1.12.0 tensorflow-1.12.0 termcolor-1.1.0 werkzeug-0.14.1
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ pip install ngraph-tensorflow-bridge
Collecting ngraph-tensorflow-bridge
  Using cached https://files.pythonhosted.org/packages/1c/c1/635f82fa03f9f5a0cc0543daff65c57dde1b94f121f20768adf325f89880/ngraph_tensorflow_bridge-0.8.0-py2.py3-none-manylinux1_x86_64.whl
Installing collected packages: ngraph-tensorflow-bridge
Successfully installed ngraph-tensorflow-bridge-0.8.0
```

- Clone tf_cnn_benchmarks

```
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ git clone -b cnn_tf_v1.12_compatible https://github.com/tensorflow/benchmarks
Cloning into 'benchmarks'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 3320 (delta 1), reused 0 (delta 0), pack-reused 3315
Receiving objects: 100% (3320/3320), 1.80 MiB | 9.19 MiB/s, done.
Resolving deltas: 100% (2322/2322), done.
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet$ cd benchmarks/scripts/tf_cnn_benchmarks/
```

Run Nasnet (stock TF)

```
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet/benchmarks/scripts/tf_cnn_benchmarks$ OMP_NUM_THREADS=28 KMP_AFFINITY=granularity=fine,compact,1,0 python tf_cnn_benchmarks.py --model=nasnet
 --forward_only --batch_size=1 --num_batches 200 --data_format NHWC --num_inter_threads=1 --num_intra_threads=28
2018-12-11 14:24:54.906743: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 AVX512F FMA
TensorFlow:  1.12
Model:       nasnet
Dataset:     imagenet (synthetic)
Mode:        BenchmarkMode.FORWARD_ONLY
SingleSess:  False
Batch size:  1 global
             1.0 per device
Num batches: 200
Num epochs:  0.00
Devices:     ['/gpu:0']
Data format: NHWC
Optimizer:   sgd
Variables:   parameter_server
==========

Step    Img/sec total_loss      top_1_accuracy  top_5_accuracy
1       images/sec: 10.4 +/- 0.0 (jitter = 0.0) 0.000   0.000   0.000
10      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
20      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
30      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
40      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
50      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
60      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
70      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
80      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
90      images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
100     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
110     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
120     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
130     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
140     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
150     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
160     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
170     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
180     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
190     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
200     images/sec: 10.4 +/- 0.0 (jitter = 0.1) 0.000   0.000   0.000
----------------------------------------------------------------
total images/sec: 10.42
----------------------------------------------------------------
```

Run Nasnet (nGraph TF)

- Import ngraph_bridge into the model

```
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet/benchmarks/scripts/tf_cnn_benchmarks$ git diff
diff --git a/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py b/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
index 6d6636c..ebe390a 100644
--- a/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
+++ b/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
@@ -23,6 +23,7 @@ from __future__ import print_function
 from absl import app
 from absl import flags as absl_flags
 import tensorflow as tf
+import ngraph_bridge

 import benchmark_cnn
 import cnn_util
```

- Run nasnet

```
(.venv) jbobba@nervana-skx21:/localdisk/jbobba/nasnet/benchmarks/scripts/tf_cnn_benchmarks$ OMP_NUM_THREADS=28 KMP_AFFINITY=granularity=fine,compact,1,0 python tf_cnn_benchmarks.py --model=nasnet
 --forward_only --batch_size=1 --num_batches 200 --data_format NCHW --num_inter_threads=1 --num_intra_threads=28
TensorFlow version installed: 1.12.0 (v1.12.0-0-ga6d8ffa)
nGraph bridge built with: 1.12.0 (v1.12.0-0-ga6d8ffa)
2018-12-11 13:56:50.405358: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 AVX512F FMA
TensorFlow:  1.12
Model:       nasnet
```

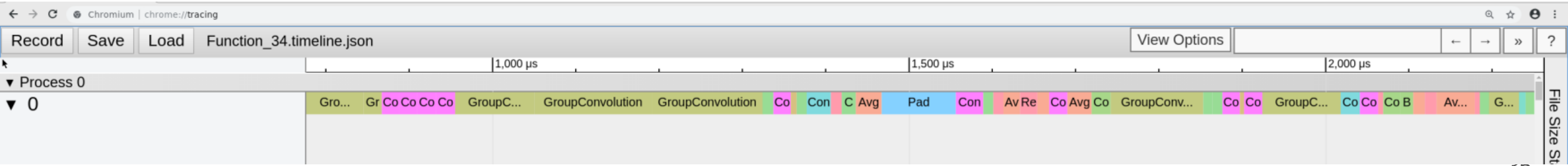
```
160     images/sec: 45.2 +/- 0.2 (jitter = 0.7) 0.000   0.000   0.000
170     images/sec: 45.3 +/- 0.2 (jitter = 0.7) 0.000   0.000   0.000
180     images/sec: 45.3 +/- 0.2 (jitter = 0.7) 0.000   0.000   0.000
190     images/sec: 45.3 +/- 0.2 (jitter = 0.6) 0.000   0.000   0.000
200     images/sec: 45.1 +/- 0.2 (jitter = 0.7) 0.000   0.000   0.000
----------------------------------------------------------------
total images/sec: 44.99
----------------------------------------------------------------
```

Performance Profiling

- Compile(): NGRAPH_PROFILE_PASS_ENABLE=1

```
 4ms ngraph::pass::LikeReplacement
 5ms ngraph::pass::NopElimination
10ms ngraph::pass::ZeroDimTensorElimination
 5ms ngraph::pass::AlgebraicSimplification
 9ms ngraph::runtime::cpu::pass::CPURnnMatFusion
 9ms ngraph::runtime::cpu::pass::CPUBatchFusion
12ms ngraph::pass::CoreFusion
34ms ngraph::runtime::cpu::pass::CPUFusion
 7ms ngraph::runtime::cpu::pass::CPUHorizontalFusion
 6ms ngraph::runtime::cpu::pass::CPUCollapseDims
 7ms ngraph::runtime::cpu::pass::CPUWorkspaceInsertion
 5ms ngraph::runtime::cpu::pass::CPUAssignment
12ms ngraph::runtime::cpu::pass::CPULayout
 6ms ngraph::pass::CommonSubexpressionElimination
10ms ngraph::runtime::cpu::pass::CPUPostLayoutOptimizations
 8ms ngraph::runtime::cpu::pass::CPUMemoryOptimization
 4ms ngraph::pass::GetOutputElementElimination
 9ms ngraph::pass::Liveness
 9ms ngraph::pass::PropagateCacheability
 7ms ngraph::pass::MemoryLayout
```

- Call(): NGRAPH_CPU_TRACING=1



Visualize graphs

- NGRAPH_ENABLE_SERIALIZE_TRACING=1
  - Serialized graphs that can be subsequently loaded into standalone nGraph tools like [nbench]

```
Function_34_000_N6ngraph4pass15LikeReplacementE.json
Function_34_001_N6ngraph4pass14NopEliminationE.json
Function_34_002_N6ngraph4pass24ZeroDimTensorEliminationE.json
Function_34_003_N6ngraph4pass23AlgebraicSimplificationE.json
Function_34_004_N6ngraph7runtime3cpu4pass15CPURnnMatFusionE.json
Function_34_005_N6ngraph7runtime3cpu4pass14CPUBatchFusionE.json
Function_34_006_N6ngraph4pass10CoreFusionE.json
Function_34_007_N6ngraph7runtime3cpu4pass9CPUFusionE.json
Function_34_008_N6ngraph7runtime3cpu4pass19CPUHorizontalFusionE.json
Function_34_009_N6ngraph7runtime3cpu4pass15CPUCollapseDimsE.json
Function_34_010_N6ngraph7runtime3cpu4pass21CPUWorkspaceInsertionE.json
Function_34_011_N6ngraph7runtime3cpu4pass13CPUAssignmentE.json
Function_34_012_N6ngraph7runtime3cpu4pass9CPULayoutE.json
Function_34_013_N6ngraph4pass30CommonSubexpressionEliminationE.json
Function_34_014_N6ngraph7runtime3cpu4pass26CPUPostLayoutOptimizationsE.json
Function_34_015_N6ngraph7runtime3cpu4pass21CPUMemoryOptimizationE.json
Function_34_016_N6ngraph4pass27GetOutputElementEliminationE.json
Function_34_017_N6ngraph4pass8LivenessE.json
Function_34_018_N6ngraph4pass21PropagateCacheabilityE.json
Function_34_019_N6ngraph4pass12MemoryLayoutE.json
```

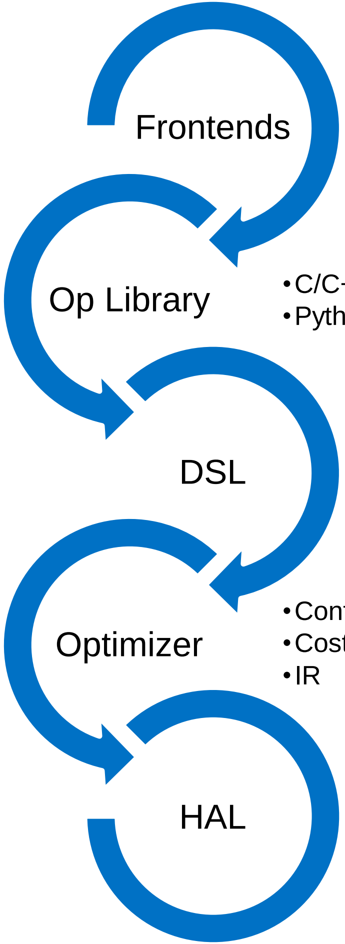
- NGRAPH_ENABLE_VISUALIZE_TRACING=1
  - Dumps graphs after each of the passes ([Ref])

PlaidML

plaidML

- https://github.com/plaidml/plaidml
- Explicitly models hardware
- Cost-based JIT schedule generation
- Differentiable DSL
- Data type & layout agnostic

PlaidML



- Keras
- ONNX
- nGraph

- C/C++
- Python

- Tile
- IR (FCs)

- Config
- Cost Model
- IR

- OpenCL
- LLVM
- CUDA

PlaidML Philosophy & High Level Architecture

‘Optimal kernels can be produced from hardware descriptions given sufficient constraints’

V0 (Current)

Tile DSL
- Differentiation
- Defractionalization
- Flattening

Flat Contractions
- Vectorization
- Cost-model Based Tiling
- Deconflict Reads and Writes
- Edge Handling

Ops
- Elementwise Fusion
- Memory Layout

HAL
- Driver Interface
- API Interactions

V1 (Stripe)

Tile
- Differentiation
- Defractionalization

Stripe
- Tensorization
- Affine Fusion
- Elementwise Fusion
- Memory Blocking (Caching)
- Paging
- Padding
- Edge Handling
- Bank Splitting
- Cost-based Optimization
- Detailed Estimation

HAL
- Driver Interface
- API Interactions

PlaidML: Tile DSL

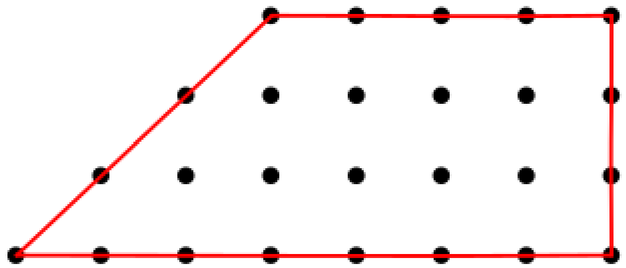
Tensor DSLs

Compiler	Matrix Multiplication in Native DSL
PlaidML	`C[i, j: I, J] = +(A[i, k] * B[k, j]);`
(taco)	`c(i, j) = a(i,k) * b(k,j)`
TVM	`tvm.sum(a[i, k] * b[j, k], axis=k)`
Tensor Comprehensions	`C(i, j) +=! A(i, k) * B(k, j)`

Polyhedral Model

- Represent index space of a tensor operation by specifying bounding polyhedron
- Alternative to nested for loops
- Often a more natural representation of a tensor operation
- Constrains problem space to that which can be bounded by a polyhedron, making subsequent optimizations simpler (vs e.g., halide)

```
for (y = 0; y < 4; ++y) {
    for (x = y; x < 8; ++x) {
        // Do stuff
    }
}
```



Tile: Contractions

- Written directly in polyhedral form; no nested for loops until writing optimized kernels
- For every valid index, compute right hand side; multiple writes to same output merged using the aggregation operation.
- Special, simple case of polyhedral model – **no complex data dependencies**

```
function (I[N, X, CI], F[W, CI, CO]) -> (O) {
    O[n, x, c: N, (X+1)/2, CO] = +(I[n, 2*x + i, d] * F[i, d, c]);
}
```

Tile: Automatic Differentiation

… start with a dilated & strided convolution:

```
function (I[N, H, W, CI], K[KH, KW, CI, CO]) -> (O) {
    O[n, y, x, co: N, H/3, W/3, CO] =
      +(I[n, 3*y + 2*j, 3*x + 2*i, ci] * K[j, i, ci, co]);
  }
```

… DI/DO is obtained by swapping the input I and the output O:

```
function (DO[N, OH, OW, CO], K[KH, KW, CI, CO]) -> (DI) {
    DI[n, 3*y + 2*j, 3*x + 2*i, ci: N, 3*OH, 3*OW, CI] =
      +(DO[n, y, x, co] * K[j, i, ci, co]);
  }
```

PlaidML v0

PlaidML v0: Code Generation

```
function matmul(A[M, L], B[L, N]) ->
    (C) {
    C[i, j: M, N] = +(A[i, k] * B[k,
    j]);
}

function maxpool(I[M, N]) -> (O) {
    O[i, j: M/2, N/2] = >(A[2*i + k,
    2*j + l]), k < 2, l < 2;
}
```

Idx	Range	O	D	K
ci	64	0	1	1
co	64	1	0	64
i	3	0	14336	12288
j	3	0	64	4096
n	32	3211264	3211264	0
x	224	14336	14336	0
y	224	64	64	0
off		0	-14400	0

Optimizer


```
"settings": {
   "threads": 256,
   "vec_size": 1,
   "mem_width": 128,
   "max_mem": 32768,
   "max_regs": 16384,
   "goal_groups": 16,
   "goal_flops_per_byte": 50
}
```

PlaidML v0: Optimization

Fixed passes, locally optimal, config driven

Vectorize

- Find a stride-1 dimension such that v = N^2 : v < vec_size , constrain tiling to multiples of v

Tile

- For each index hill climb and use cost model to maximize reuse while fitting in cache & registers

Load

- Create a loading pattern designed to minimize bank conflicts for any number of parallel readers

Loop

- Order loops using a topological ordering to maximize cache reuse

Thread

- Rollup as many inner loops into hardware threads as possible

PlaidML v0: Runtime / HAL


```
__kernel void test_kernel(__global float4* out, __global const
     float* in1, __global const float4* in2)
{
  ssize_t tid = get_local_id(0);
  float4 agg[8] = {((float4) (sum_base_float)), ((float4)
      (sum_base_float)), ((float4) (sum_base_float)), ((float4)
      (sum_base_float)), ((float4) (sum_base_float)), ((float4)
      (sum_base_float)), ((float4) (sum_base_float)), ((float4)
      (sum_base_float)), };
  __local float in1_shared[4160];
  __local float4 in2_shared[520];
  ssize_t v1_gid = (get_group_id(1)*8);
  ssize_t v0_gid = (get_group_id(0)*64);
  for(ssize_t v2_gid = 0; v2_gid < 256; v2_gid += 64)
  {
    {
      ssize_t gbase = ((0+(v2_gid*1))+(v0_gid*256));
      ssize_t v2_tid = ((tid/1)%16);
      ssize_t v0_tid = ((tid/16)%4);
      for(ssize_t v2_lid = 0; v2_lid < 1; v2_lid += 1)
      {
        ssize_t v2 = ((64*v2_lid)+v2_tid);
        for(ssize_t v0_lid = 0; v0_lid < 16; v0_lid += 1)
        {
          ssize_t v0 = ((4*v0_lid)+v0_tid);
          ssize_t lidx = ((0+(4*v2))+(65*v0));
          ssize_t gidx = ((gbase+(4*v2))+(256*v0));
          float4 val = vload4(gidx, in1);
          vstore4(val, lidx, in1_shared);
        }
      }
    }
    {
      ssize_t gbase = ((0+(v1_gid*1))+(v2_gid*64));
      ssize_t v1_tid = ((tid/1)%8);
      ssize_t v2_tid = ((tid/8)%8);
      for(ssize_t v1_lid = 0; v1_lid < 1; v1_lid += 1)
      {
        ssize_t v1 = ((8*v1_lid)+v1_tid);
        for(ssize_t v2_lid = 0; v2_lid < 8; v2_lid += 1
….
```

PlaidML v0: Summary

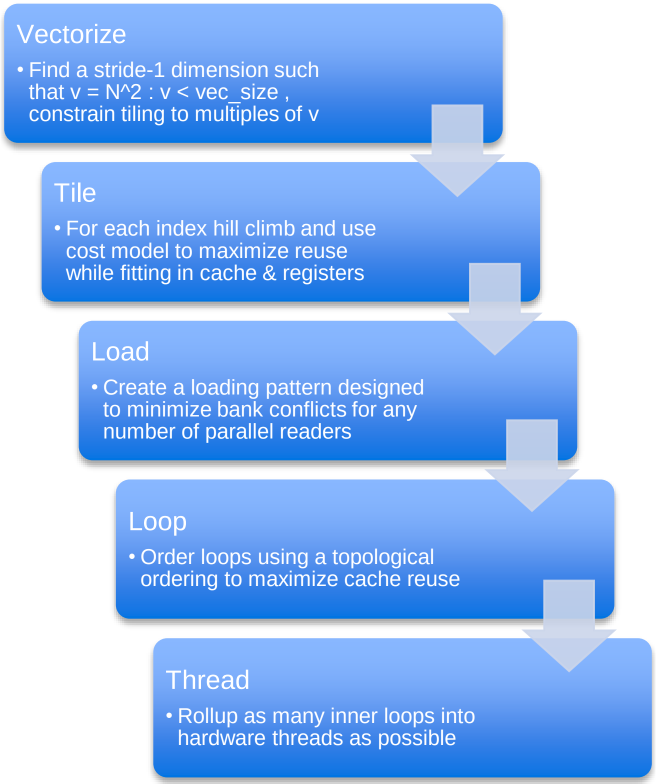
- Supports training & inference
- Supports most frameworks (except training via pyTorch)
- Performance portable for major GPU architectures
  - Fixed Optimization passes
  - Minimal hardware config
- Not well suited for deep learning accelerators or other architectures that benefit from micro-kernels
  - Volta, Mali, Myriad, etc

PlaidML v1: Stripe

Extending PlaidML to encompass the modern accelerator landscape

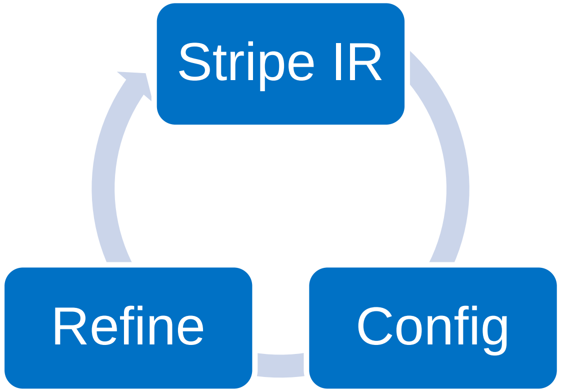
PlaidML v1: Evolution

- v0's fixed pass architecture can't extend past typical GPU architectures in a performance portable manner
- v0's fixed pass architecture is fundamentally brittle and tightly coupled
- v1's primary challenge was to invent an abstraction capable of modelling v0 as a config driven subset of v1.



PlaidML v1 / Stripe: Polyhedral IR

- PlaidML v1 introduces **Stripe**: a polyhedral IR that is highly amenable to optimization.
- **Stripe** enables distinct passes that process stripe and emit more stripe
- **Stripe** fundamentally represents operations over a polyhedral tensor space.

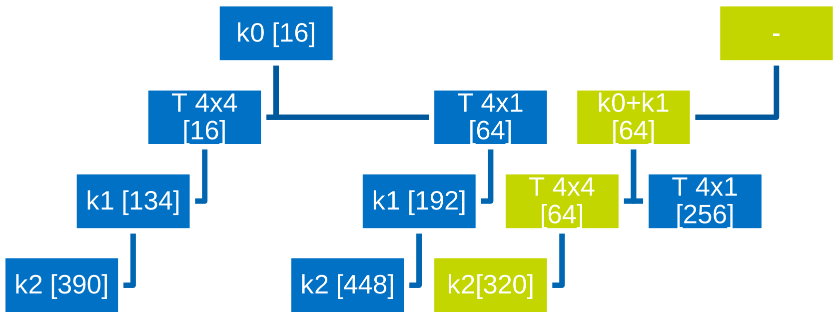


PlaidML v1 / Stripe

- Stripe enables:
  - Arbitrary tensorization
  - Affine vertical fusion
  - Arbitrarily complex memory hierarchry
  - Heterogenous compute topologies
  - Detailed performance / cost estimates
  - Software / hardware co-design

PlaidML v1 / Stripe: Pathfinding Optimizer

- Add a computation node
- Compute the min cost for each potential optimization branch for the subgraph so far
- Add nodes and explore according to A*



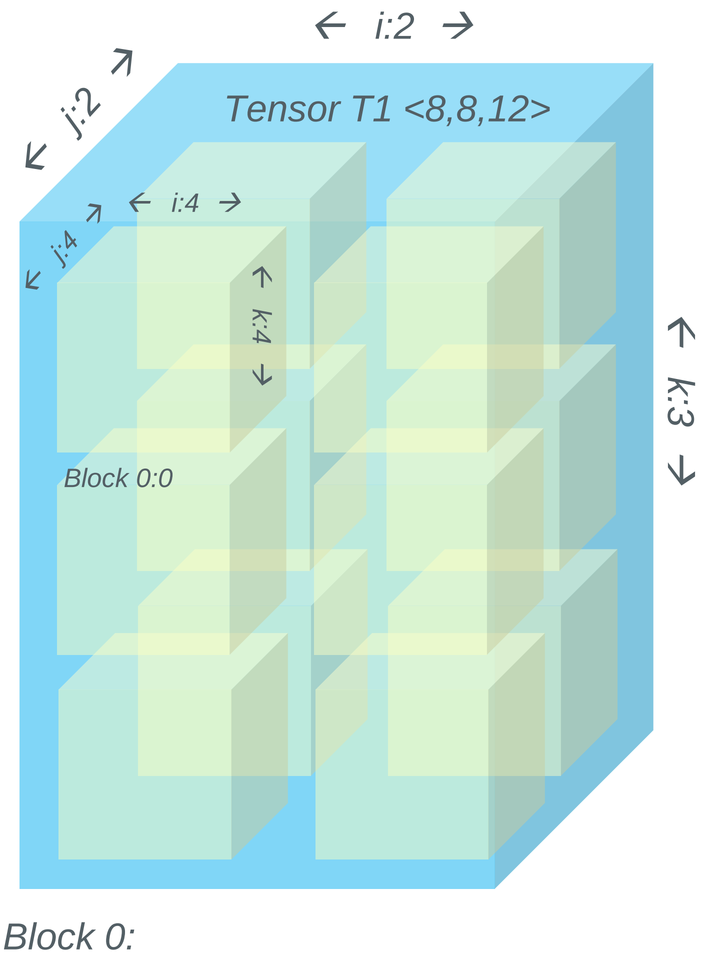
PlaidML v1 / Stripe : Mapping to PlaidML v0

Pass	Branches	Strategy	Comment
Tensorize	[8x1],[4x1],[2x1],[1x1]	Top 1	Pick best applicable vectorization
Tile	prod(range(idxs))	Hill-climb pow(2), top 1	Increase size by powers of 2 until memory is exceeded, pick best tiling
L1 Cache	-	-	Load memory into shared L1, avoid bank conflicts
Thread	[16, 32, 64, 128, 256]	Hill-climb pow(2), top 1	Find the most threads that can be used without exceeding problem domain
Elementwise Fusion	-	-	Fuse this kernel with the next if it is an elementwise kernel
Flatten	-	-	Flatten and order loops to minimize cost

Stripe in Depth

Stripe Conceptual Model

- Describes nested and repeated computational **BLOCKS,** each **BLOCK** represents a set of parallelizable computations
- **BLOCKS** are described by **INDEXES** and **CONSTRAINTS** that create polyhedral bounds over views of tensors called **REFINEMENTS**
- Nested **BLOCKS** have their own **INDEXES**
- Nested **BLOCKS** can create polyhedral sub regions of **REFINEMENTS** in the parent block by creating more **REFINEMENTS** which are automatically offset.
- The interior of a **BLOCK** nest contains code that is executed for every valid value of every **INDEX** of every containing **BLOCK**.



Stripe IR Explained: Stripe Top (HW Independent)

Tags

Allocations

Refinements

Indexes

Nested Blocks

```
0: #program block [] ( // layer_test7
    none new@0x00000000<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
    none new@0x00000000<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
    …
    none new@0x00000000<[0]> O3[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
) {
  0: #main block [] ( // main
      in<[0]> I[0, 0, 0] i8(1024:32768, 1024:32, 32:1)
      in<[0]> K1[0, 0, 0, 0] i8(3:6144, 3:2048, 32:64, 64:1)
      out<[0]> O1[0, 0, 0]:assign i8(1024:65536, 1024:64, 64:1)
      none new@0x00000000<[0]> O1[0, 0, 0] i8(1024:65536, 1024:64, 64:1)
  ) {
    0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] (
        // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
        -1 + kx + x >= 0
        1024 - kx - x >= 0
        -1 + ky + y >= 0
        1024 - ky - y >= 0
        out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
        in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
        in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I)
      1: $K1 = load(K1)
      2: $O1 = mul($I, $K1)
      3: O1 = store($O1)
    }
    1: …
  }
}
```

Constraints

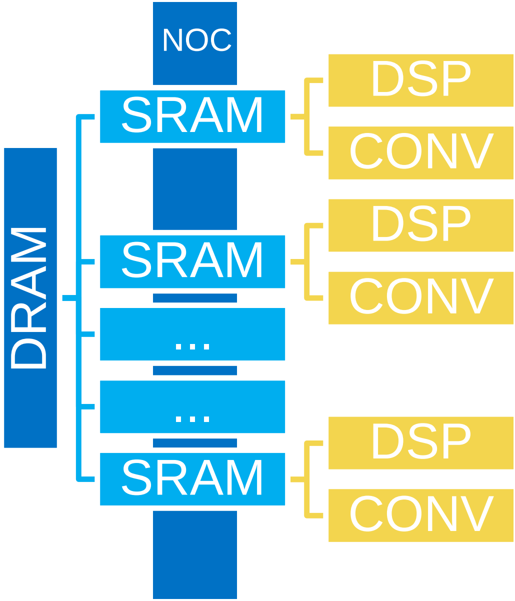
Tile Code

Aggregators

SSA IL

Stripe: Hardware Model

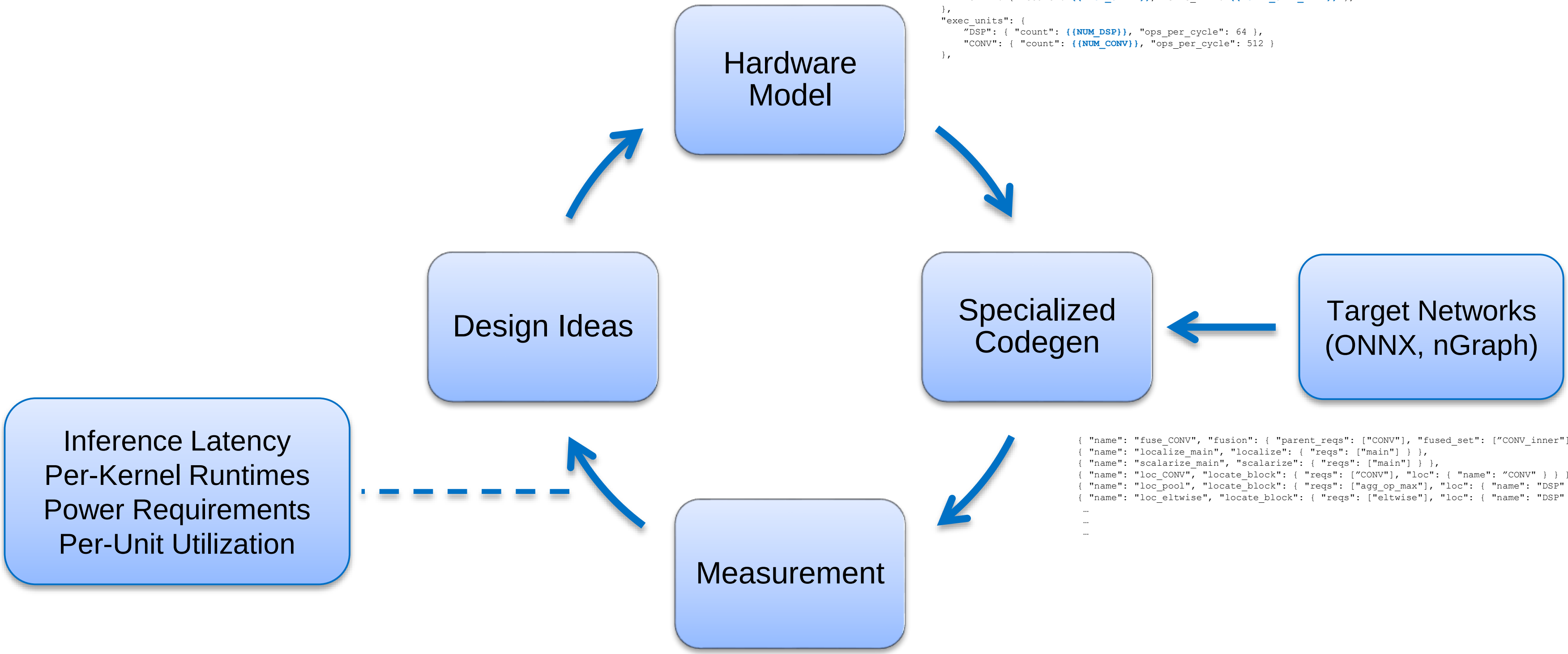
```
"clock_mhz": {{ CLOCK_MHZ }},
"mem_units": {
    "DRAM": { "count": 1, "size_KiB": 1048576 },
    "SRAM": { "count": {{ NUM_SRAM }}, "size_KiB": {{ SRAM_SIZE_KIB }} },
},
"exec_units": {
    ”DSP": { "count": {{NUM_DSP}}, "ops_per_cycle": 64 },
    "CONV": { "count": {{NUM_CONV}}, "ops_per_cycle": 512, “pipeline_depth”: 2 }
},
"tx_units": {
    "DMA": { "count": 1 },
    "NOC": { "count": 1 },
},
"buses": [
    { "sources": ["DRAM[0]"], "sinks": ["DMA[0]"], "bytes_per_cycle": 64 },
    { "sources": ["DMA[0]"], "sinks": ["DRAM[0]"], "bytes_per_cycle": 64 },
    {
        "sources": ["DMA[0]"],
        "sinks": [{% for i in range(NUM_SRAM) %} "SRAM[{{i}}]"{endfor %}],
        "bytes_per_cycle": 64
    },
    {
        "sources": ["NOC[0]"],
        "sinks": [{% for i in range(NUM_SRAM) %} "SRAM[{{i}}]"{% endfor %}],
        "bytes_per_cycle": 512
    },
.
.
.
```



Stripe: Optimizer Config

```
{ "name": "fuse_CONV_add", "fusion": { "a_reqs": ["CONV"], "b_reqs": ["eltwise_add"], "fused_set": ["CONV"] } },
{ "name": "fuse_CONV_zelu", "fusion": { "a_reqs": ["CONV"], "b_reqs": ["eltwise_zelu"], "fused_set": ["CONV"] } },
{ "name": "fuse_CONV", "fusion": { "parent_reqs": ["CONV"], "fused_set": [”CONV_inner"] } },
{ "name": "localize_main", "localize": { "reqs": ["main"] } },
{ "name": "scalarize_main", "scalarize": { "reqs": ["main"] } },
{ "name": "loc_CONV", "locate_block": { "reqs": [”CONV"], "loc": { "name": ”CONV" } } },
{ "name": "loc_pool", "locate_block": { "reqs": ["agg_op_max"], "loc": { "name": "DSP" } } },
{ "name": "loc_eltwise", "locate_block": { "reqs": ["eltwise"], "loc": { "name": "DSP" } } },
 …
 …
 …
{ "name": "deps_main", "compute_deps": { "reqs": ["main"] } },
{
    "name": "schedule_main",
    "schedule": {
        "reqs": ["main"],
        "mem_loc": { "name": ”SRAM" },
        "mem_KiB": {{ SRAM_SIZE_KIB / NUM_SRAM }},
        "alignment": 16,
        "xfer_loc": { "name": "DMA" },
        "allow_out_of_range_accesses": true,
        "num_banks": {{ NUM_SRAM }}
    }
},
{ "name": "place_program", "memory_placement": { "reqs": ["program"], "locs": [{ "name": "DRAM" }], "alignment": 4 } }
```

Stripe: Enabling Hardware / Software Co-Design

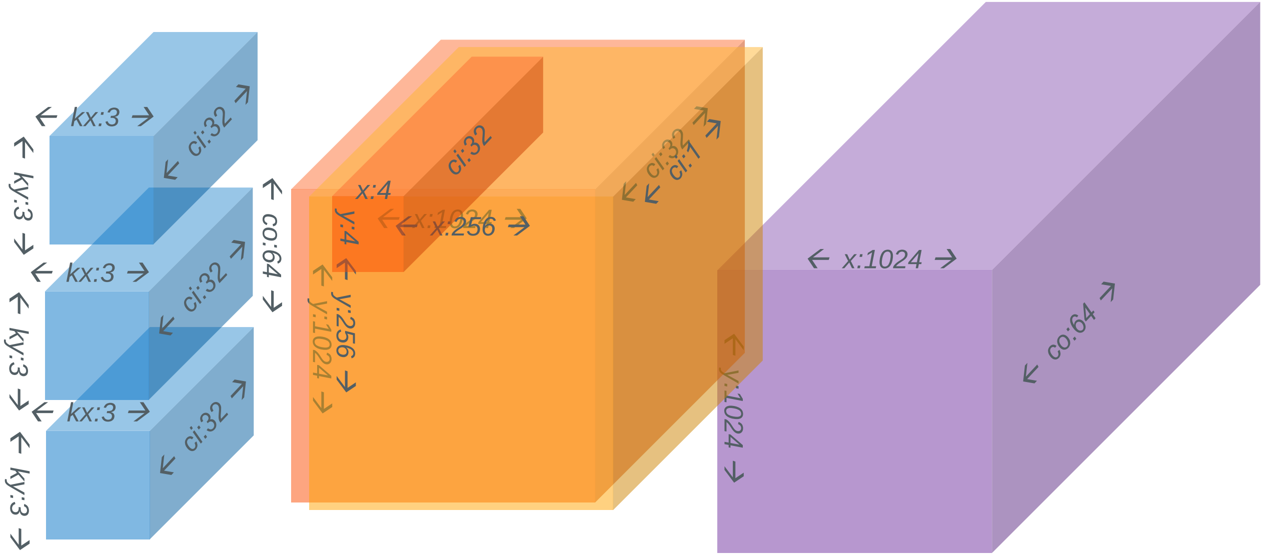



```
"clock_mhz": {{ CLOCK_MHZ }},
"mem_units": {
    "DRAM": { "count": 1, "size_KiB": 1048576 },
    "SRAM": { "count": {{ NUM_SRAM }}, "size_KiB": {{ SRAM_SIZE_KIB }} },
},
"exec_units": {
    ”DSP": { "count": {{NUM_DSP}}, "ops_per_cycle": 64 },
    "CONV": { "count": {{NUM_CONV}}, "ops_per_cycle": 512 }
},
```

```
{ "name": "fuse_CONV", "fusion": { "parent_reqs": ["CONV"], "fused_set": [”CONV_inner"] } },
{ "name": "localize_main", "localize": { "reqs": ["main"] } },
{ "name": "scalarize_main", "scalarize": { "reqs": ["main"] } },
{ "name": "loc_CONV", "locate_block": { "reqs": [”CONV"], "loc": { "name": ”CONV" } } },
{ "name": "loc_pool", "locate_block": { "reqs": ["agg_op_max"], "loc": { "name": "DSP" } } },
{ "name": "loc_eltwise", "locate_block": { "reqs": ["eltwise"], "loc": { "name": "DSP" } } },
…
…
…
```

Stripe: Tensorization

```
"tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": "i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": "c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": "i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": "i2", "size": 4, "outs": [-1], "ins": [ 0, -1] }, …
]},] } },
```



Stripe: Tensorization

```
"tensorize": {
"reqs": [ "agg_op_add", "comb_op_mul" ], "outer_set": [ "CONV" ], "inner_set": [ "CONV_inner" ],
"stencils": [
  {"idxs": [{ "name": "i1", "size": 32, "outs": [-1], "ins": [-1, 0] }, { "name": "c", "size": -1, "outs": [ 0], "ins": [-1, -1] }]},
  {"idxs": [{ "name": "i1", "size": 4, "outs": [-1], "ins": [-1, 0] }, { "name": "i2", "size": 4, "outs": [-1], "ins": [ 0, -1] }, …
]},] } },
```

BEFORE:

```
0: #agg_op_add #comb_op_mul #contraction #kernel block [ci:32, co:64, kx:3, ky:3, x:1024, y:1024] ( // kernel_0
        // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
        out<[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
        in<[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
        in<[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:64, 1:1)
    ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }
```

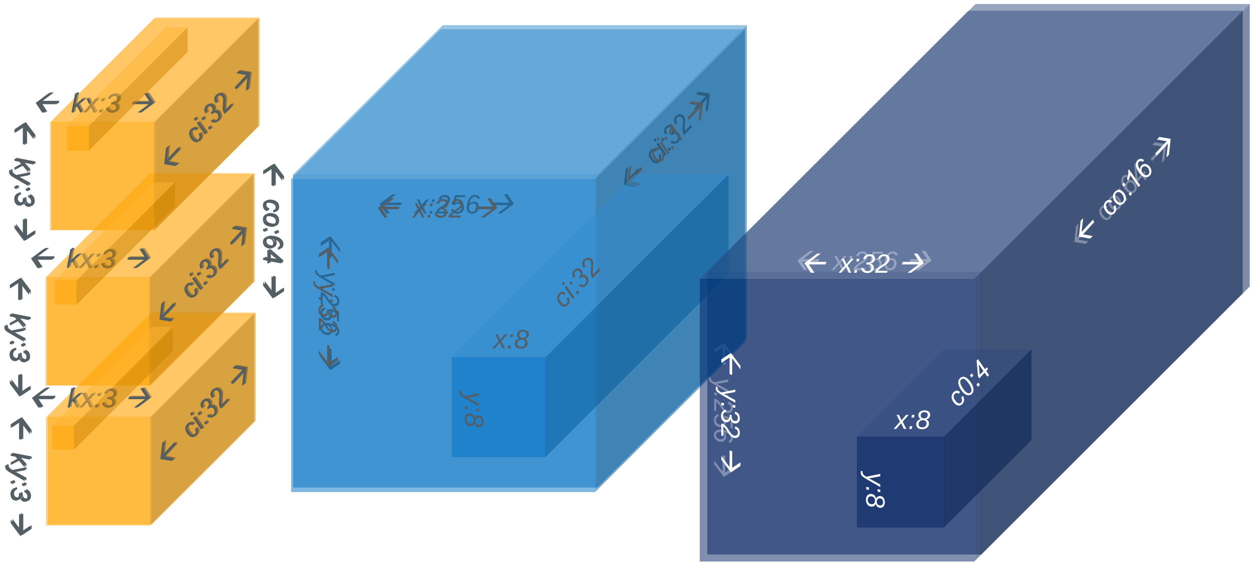
AFTER:

```
0: #agg_op_add #comb_op_mul #contraction #CONV #kernel block [ci:1, co:1, kx:1, ky:1, x:256, y:256] ( // kernel_0
        // O1[x, y, co : X, Y, C2] = +(I[-1 + kx + x, -1 + ky + y, ci] * K1[kx, ky, ci, co])
        out<DRAM[0]> O1[4*x, 4*y, 16*co]:add i8(4:65536, 4:64, 16:1)
        in<DRAM[0]> I[kx + 4*x, ky + 4*y, 32*ci] i8(4:32768, 4:32, 32:1)
        in<DRAM[0]> K1[kx, ky, 32*ci, 16*co] i8(1:6144, 1:2048, 32:1, 16:32)
    ) {
      0: #CONV_inner block [ci:32, co:64, kx:3, ky:3, x:4, y:4] ( // kernel_0
          out<DRAM[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
          in<DRAM[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
          in<DRAM[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:1, 1:32)
      ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }}
```

Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : true,
  “memory” : “SRAM” // ”pipeline_depth” : 2
}
```

kx	ky	ci	co	x	y	cost
1	1	32	4	8	8	120
1	1	16	8	8	8	140
1	1	32	5	4	4	270
3	3	32	1	6	6	310
3	3	16	1	9	9	340



Stripe: Auto-Tile

```
"autotile": {
  "reqs" : ["conv"], "outer_set" : ["conv"], "inner_set" : ["conv_inner"],
  "only_po2" : true,
  "memory" : "SRAM" // "pipeline_depth" : 2
}
```

BEFORE:

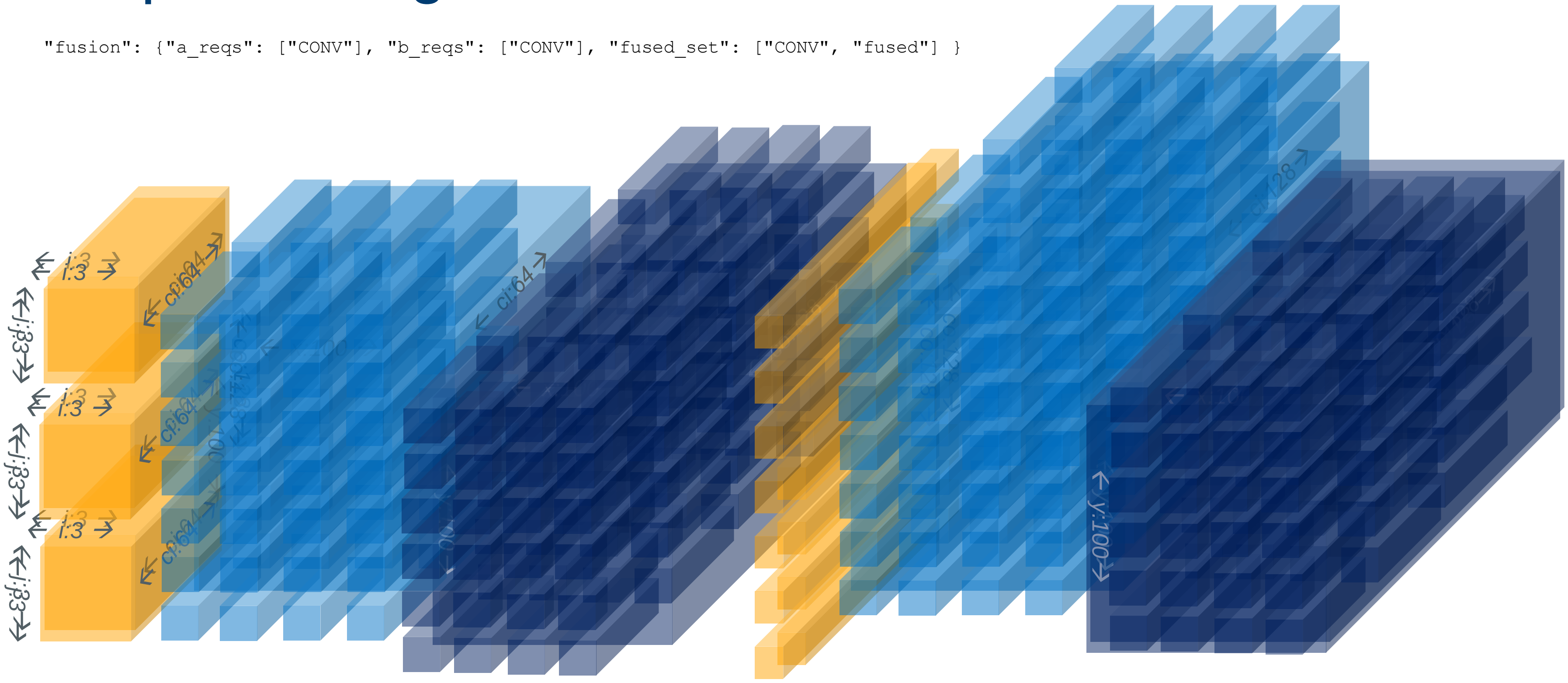
```
0: #conv block<CONV[0]> [ci:32, co:64, kx:3, ky:3, x:256, y:256] (
       out<DRAM[0]> O1[4*x, 4*y, 16*co]:add i8(4:65536, 4:64, 16:1)
       in<DRAM[0]> I[kx + 4*x, ky + 4*y, 32*ci] i8(4:32768, 4:32, 32:1)
       in<DRAM[0]> K1[kx, ky, 32*ci, 16*co] i8(1:6144, 1:2048, 32:1, 16:32)
) {
   0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
}
```

AFTER:

```
0: #conv block<CONV[0]> [ci:1, co:16, kx:3, ky:3, x:32, y:32] ( // kernel_0
    out<DRAM[0]> O1[16*x, 16*y, 64*co]:add i8(16:65536, 16:64, 64:1)
    in<DRAM[0]> I[kx + 16*x, ky + 16*y, 32*ci] i8(16:32768, 16:32, 32:1)
    in<DRAM[0]> K1[kx, ky, 32*ci, 64*co] i8(1:6144, 1:2048, 32:1, 64:32)
) {
    0: <Elided memory xfers>
    1: #conv_inner block<CONV[0]> [ci:32, co:4, kx:1, ky:1, x:8, y:8] ( // No halos as the tiling makes lots of 1x1 convolutions
        out<SRAM[0]> O1[x, y, co]:add i8(1:65536, 1:64, 1:1)
        in<SRAM[0]> I[-1 + kx + x, -1 + ky + y, ci] i8(1:32768, 1:32, 1:1)
        in<SRAM[0]> K1[kx, ky, ci, co] i8(1:6144, 1:2048, 1:1, 1:32)
    ) {
      0: $I = load(I);  1: $K1 = load(K1);  2: $O1 = mul($I, $K1);  3: O1 = store($O1)
    }
}
```

Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```



Stripe: Fusing Contractions

```
"fusion": {"a_reqs": ["CONV"], "b_reqs": ["CONV"], "fused_set": ["CONV", "fused"] }
```

BEFORE:

```
0: #agg_op_add #comb_op_mul #CONV #contraction #kernel block [ci:64, co:128, i:3, j:3 x:100, y:100] ( // kernel_0
  // O1[x, y, co : X, Y, CO1] = +(In[-1 + i + x, -1 + j + y, ci] * K1[i, j, ci, co])
  ) {
    0: $In = load(In);  1: $K1 = load(K1);  2: $O1 = mul($In, $K1);  3: O1 = store($O1)
  }
1: #agg_op_add #comb_op_mul #CONV #contraction #kernel block [ci:128, co:128, x:100, y:100] ( // kernel_1
  // O2[x, y, co : X, Y, CO2] = +(O1[i + x, j + y, ci] * K2[i, j, ci, co])
  ) {
    0: $O1 = load(O1);  1: $K2 = load(K2);  2: $O2 = mul($O1, $K2);  3: O2 = store($O2)
  }
```

AFTER:

```
0: #fused block [co:8, x:100, y:100] ( // kernel_0+kernel_1 … ) {
  0: block [ci:64, co:16, i:3, j:3, x:1, y:1] (…){
    out<SRAM[0]> O1[x, y, co]:add fp32(1:16, 1:16, 1:16, 1:1)
    in<[0]> In[-1 + i + x, -1 + j + y, ci] fp32(1:640000, 1:6400, 1:64, 1:1)
    in<[0]> K1[i, j, ci, co] fp32(1:24576, 1:8192, 1:128, 1:1)
  ) {
    0: $In = load(In);  1: $K1 = load(K1);  2: $O1 = mul($In, $K1);  3: O1 = store($O1)
  }
  1: block [ci:64, co:16, x:1, y:1] (…) {
    out<[0]> O2[x, y, co]:add fp32(1:1280000, 1:12800, 1:128, 1:1)
    in<SRAM[0]> O1[x, y, ci] fp32(1:16, 1:16, 1:16, 1:1)
    in<[0]> K2[0, 0, ci, co] fp32(1:16384, 1:16384, 1:128, 1:1)
  ) {
    0: $O1 = load(O1);  1: $K2 = load(K2);  2: $O2 = mul($O1, $K2);  3: O2 = store($O2)
  }
}
```

PlaidML v1 / Stripe

- Stripe enables:
    - Arbitrary tensorization
    - Affine vertical fusion
    - Arbitrarily complex memory hierarchry
    - Heterogenous compute topologies
    - Detailed performance / cost estimates
    - Software / hardware co-design

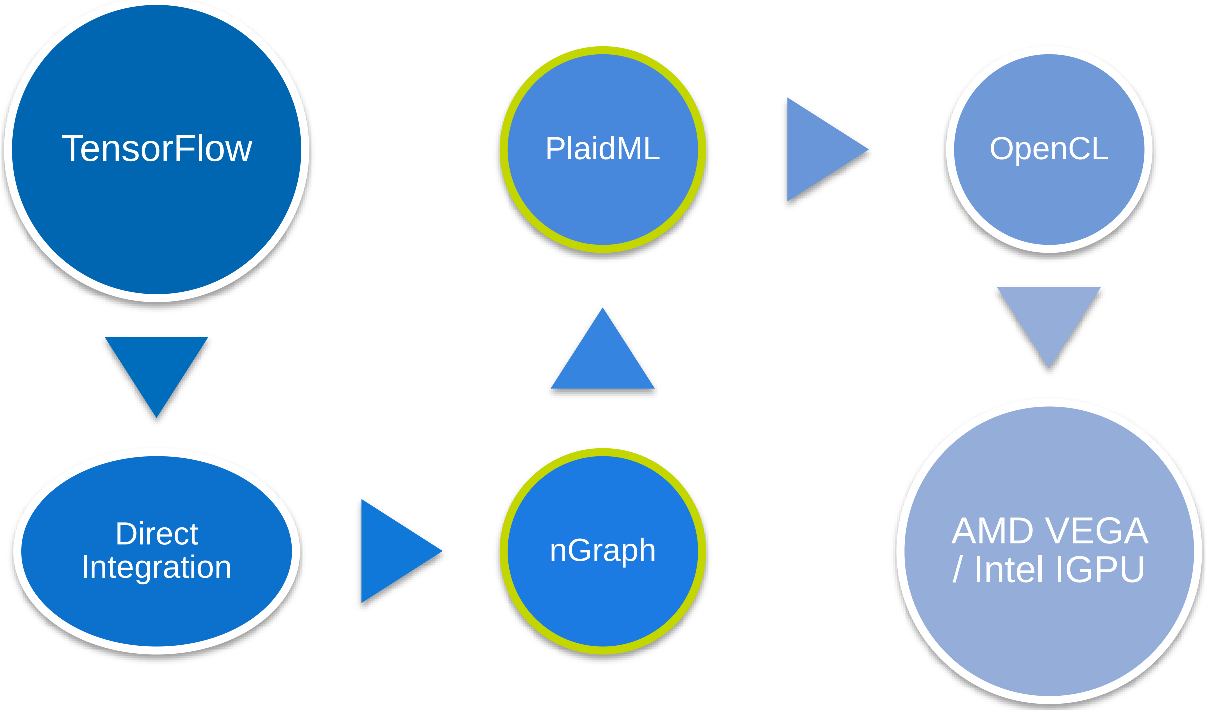
PlaidML v1.x / Stripe : Status

- Initial code upstreamed to public as of 0.5
- Configurations for GPUs, CPUs & porting v0 to Stripe in progress
- Extensions for conditionals, loops, and indirection (scatter / gather) coming in v1
- Paper coming out early next year
- Specification available on request to: tim.zerrell@intel.com

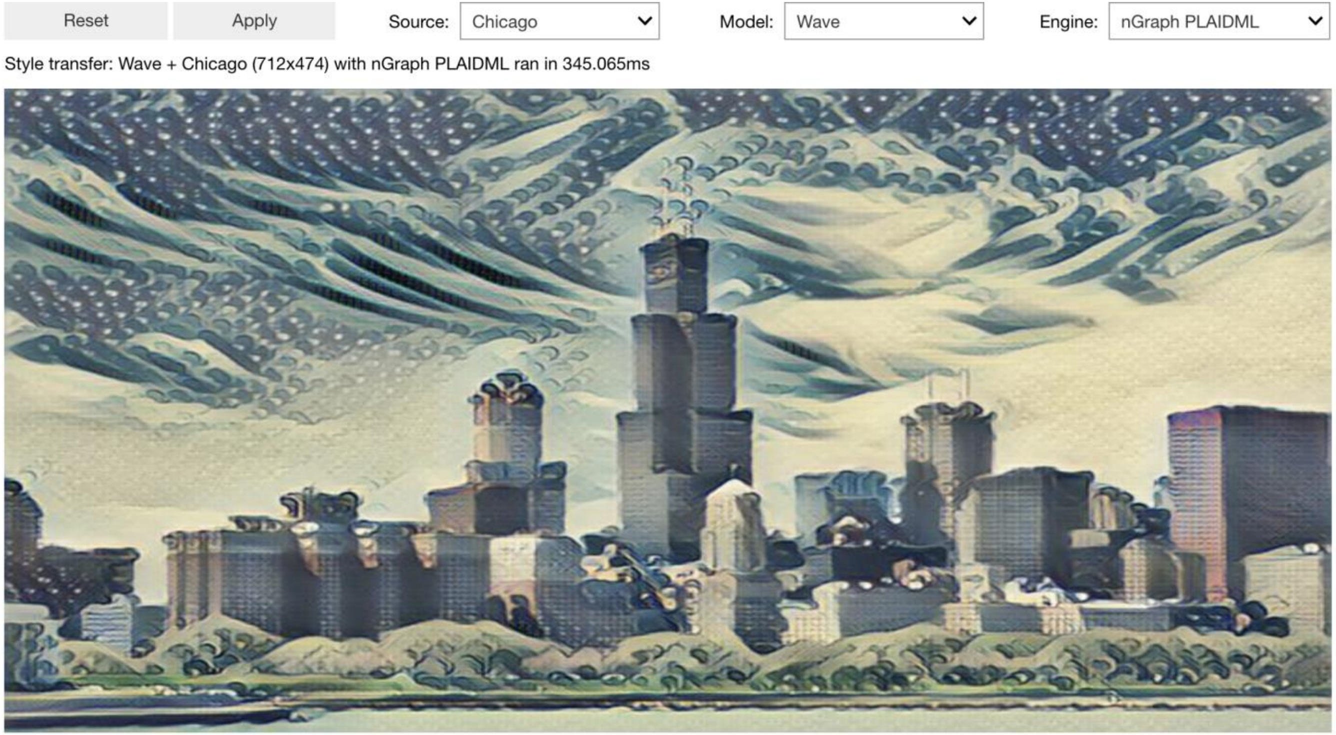
Demo: nGraph + PlaidML

Accelerated Neural Style Transfer on a Macbook

Tengplocl: TensorFlow nGraph PlaidML OpenCL



nGraph on Iris & Radeon vs Coffee Lake i7



Conclusion

Call to Action

- **Try nGraph out now**!
    - **nGraph Beta** works out of box with TensorFlow, MXNet, ONNX
    - **nGraph is open source.** Clone the repo and get started today!

https://ngra.ph/repo

intel® AI

Some further reading

Intel nGraph: An Intermediate Representation, Compiler, and Executor for Deep Learning. Scott Cyphers et al. SysML 2018. (https://arxiv.org/abs/1801.08058)

nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data. Fabian Boemer, Yixing Lao, and Casimir Wierzynski. (https://arxiv.org/abs/1810.10121)