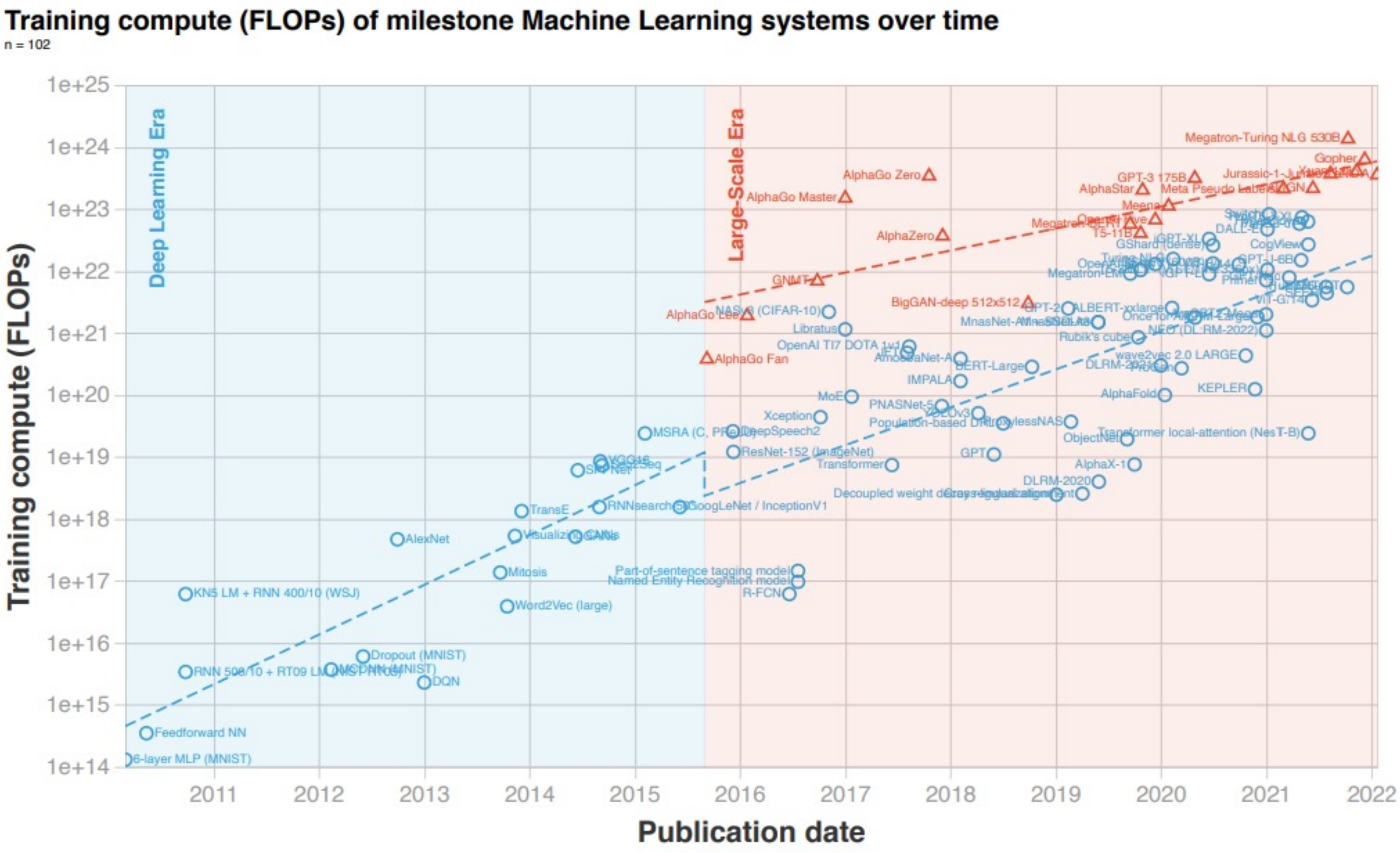
Oliver Rausch*, Tal Ben-Nun*, Nikoli Dryden, Andrei Ivanov, Shigang Li, Torsten Hoefler

DaCeML: A Data-Centric Optimization Framework for Machine Learning

This project received funding from the European Research Council under the European Union's Horizon 2020 programme (Project PSAP, No. 101002047); and receives EuroHPC-JU funding under grants MAELSTROM, No. 955513 and DEEP-SEA, No. 955606, with support from the Horizon 2020 programme

Large Scale Computation in ML



Sevilla et al. "Compute trends across three eras of machine learning." (2022).

Contemporary ML Systems are Inflexible

- **Researchers demand efficient, large-scale compute more than ever**
- **Almost all optimizations relate to the data movement**
- **But current compilers are highly specialized towards**
  - Operators
  - Transformations
  - Models
- **Tuning, or even just visualizing the output of a compiler is difficult**

DaCeML

- **Data-centric lowering and multi-level optimization of DNNs**

Usability

Generality

Interactivity

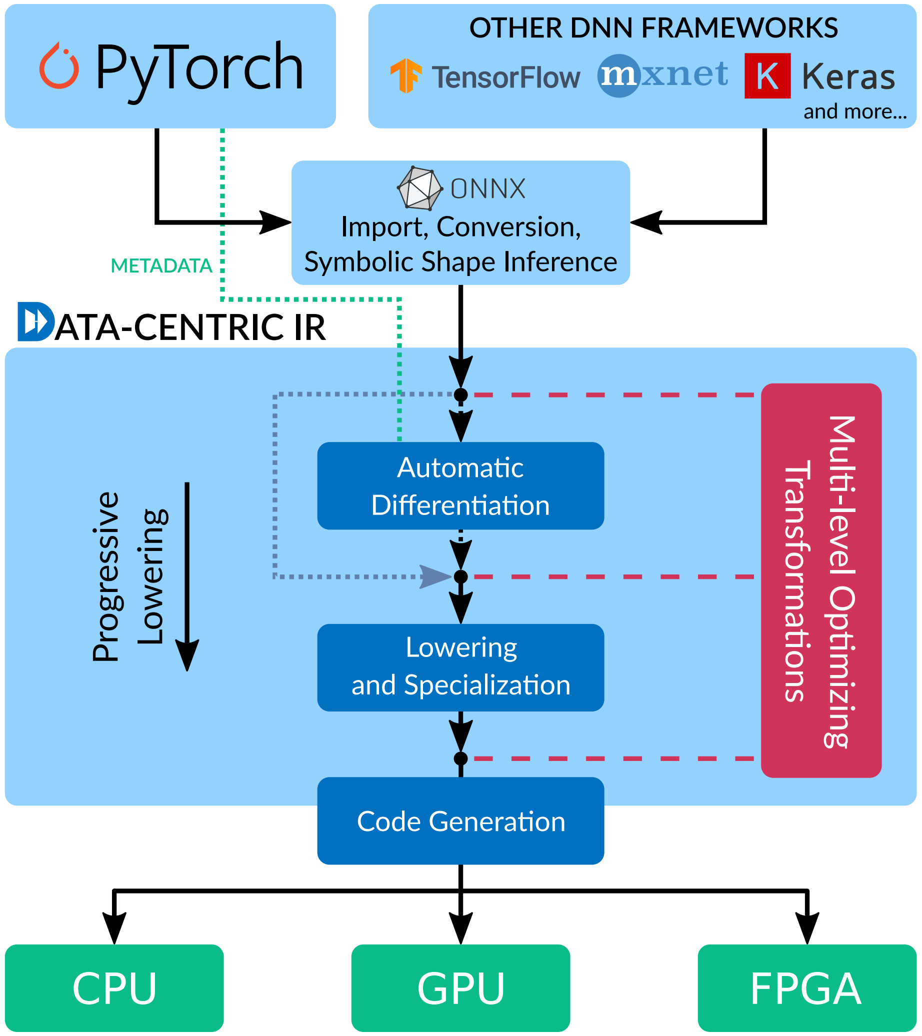
Usage

```
from torch import nn
from daceml.pytorch import dace_module

@dace_module
class MyModule(nn.Module):
  def __init__(self, n_in, n_out):
    super().__init__()
    self.linear = nn.Linear(n_in, n_out)
    self.fanout = n_out


  def forward(self, x):
    return self.linear(x) / self.fanout
```

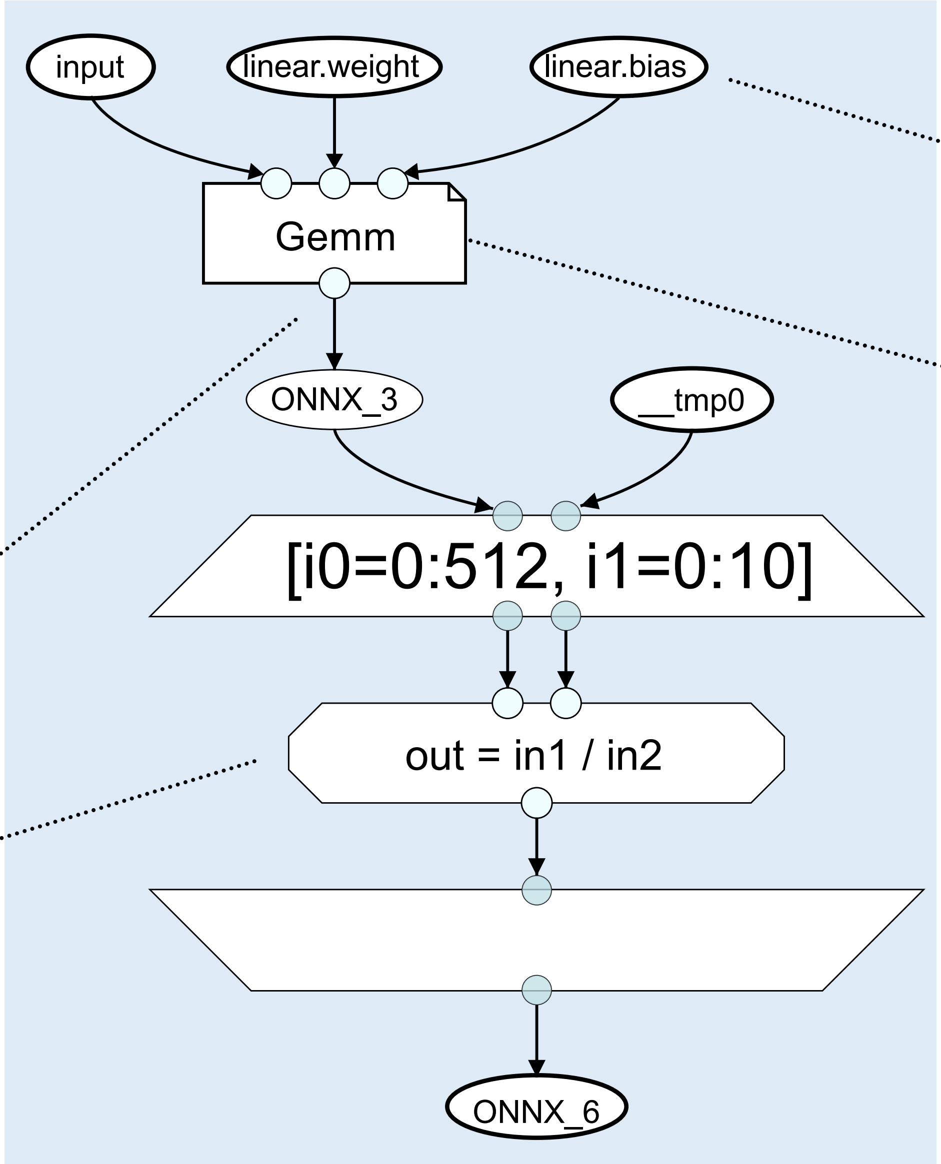
System Overview



```python
from torch import nn
from daceml.pytorch import dace_module

@dace_module
class MyModule(nn.Module):
  def __init__(self, n_in, n_out):
    super().__init__()
    self.linear = nn.Linear(n_in, n_out)
    self.fanout = n_out

  def forward(self, x):
    return self.linear(x) / self.fanout
```



Access nodes to data containers

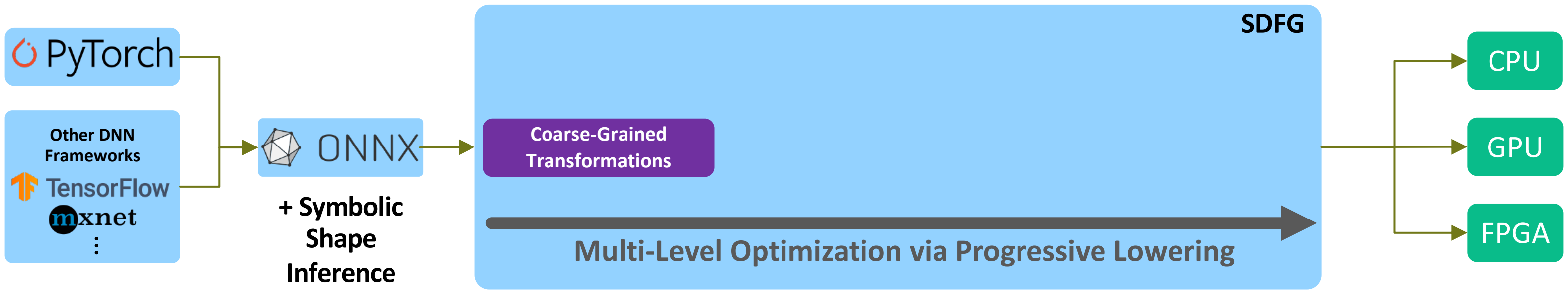
Library nodes with domain-specific semantics

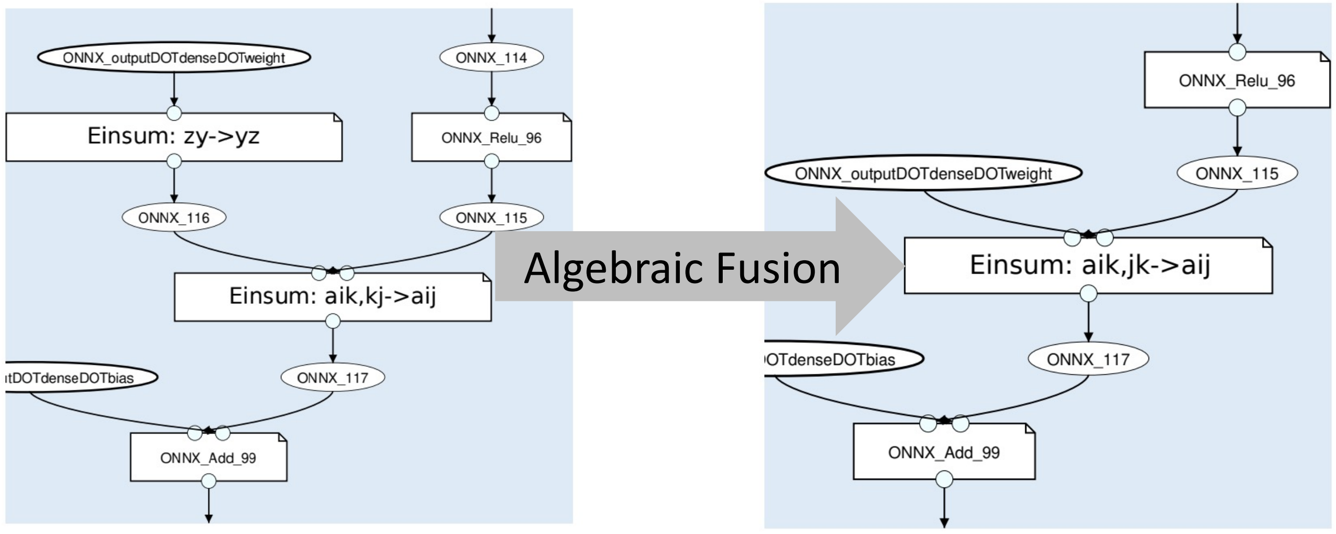
Memlets: explicit data movement at all granularities

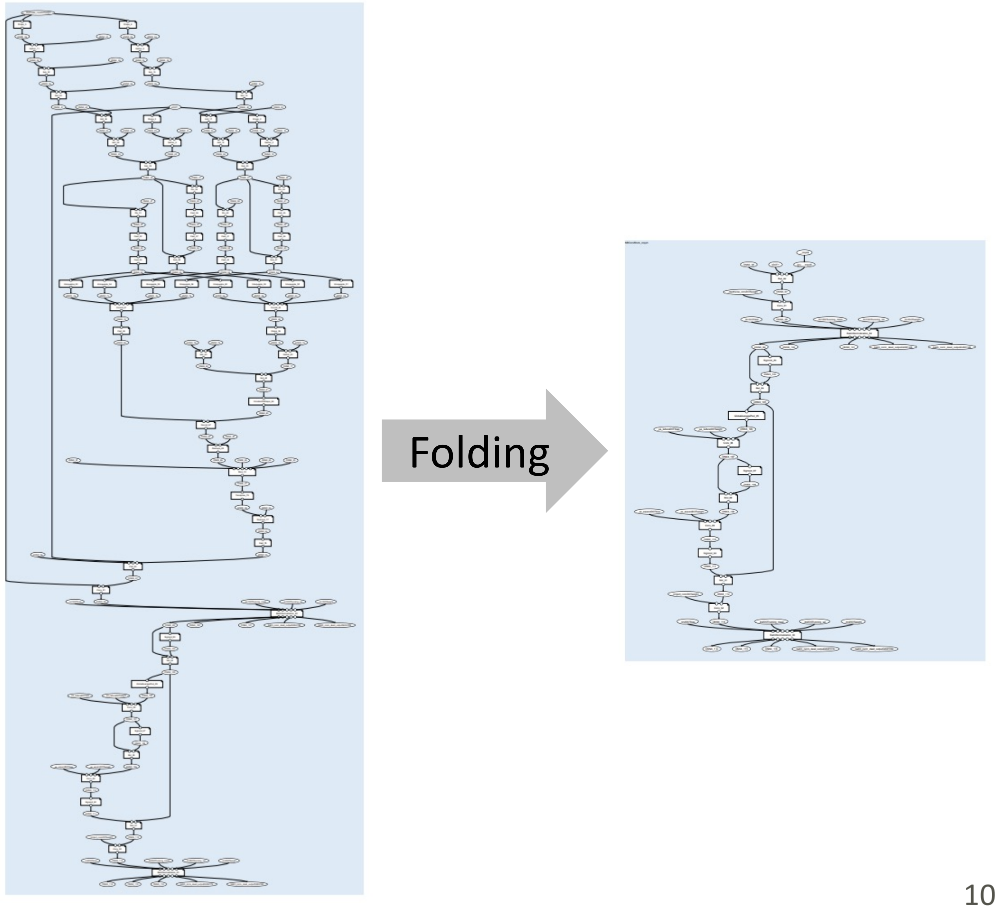
Maps: Parametric parallelism scopes

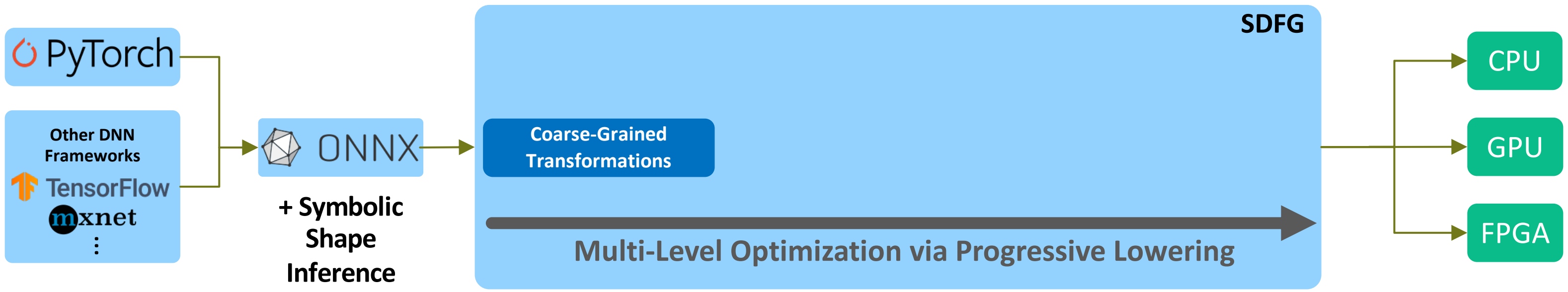
Tasklets: stateless computations

Optimization Pipeline


PyTorch
Other DNN Frameworks
TensorFlow
mxnet
⋮
ONNX
+ Symbolic Shape Inference
SDFG
Coarse-Grained Transformations
Multi-Level Optimization via Progressive Lowering
CPU
GPU
FPGA


ONNX_outputDOTdenseDOTweight
ONNX_114
Einsum: zy->yz
ONNX_Relu_96
ONNX_116
ONNX_115
Einsum: aik,kj->aij
ıtDOTdenseDOTbias
ONNX_117
ONNX_Add_99
Algebraic Fusion
ONNX_Relu_96
ONNX_outputDOTdenseDOTweight
ONNX_115
Einsum: aik,jk->aij
OTdenseDOTbias
ONNX_117
ONNX_Add_99

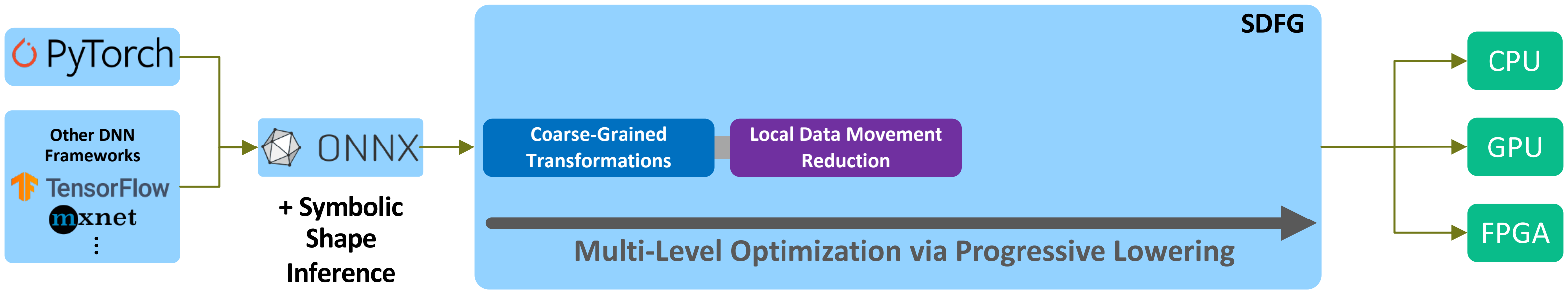

Folding

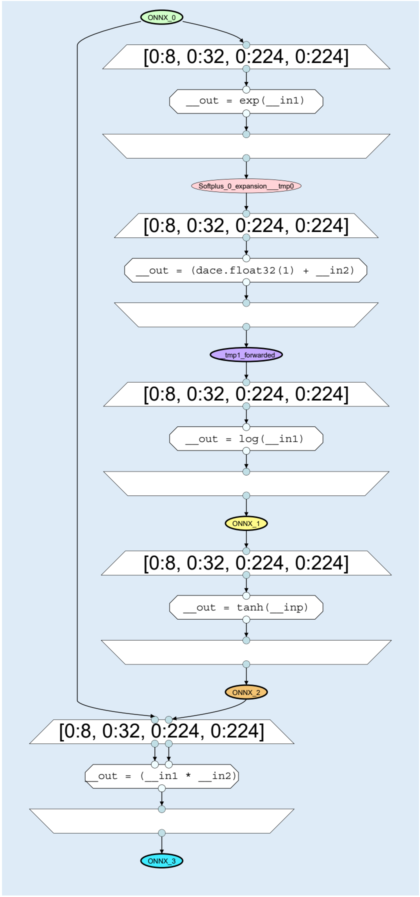


Lowering ONNX nodes

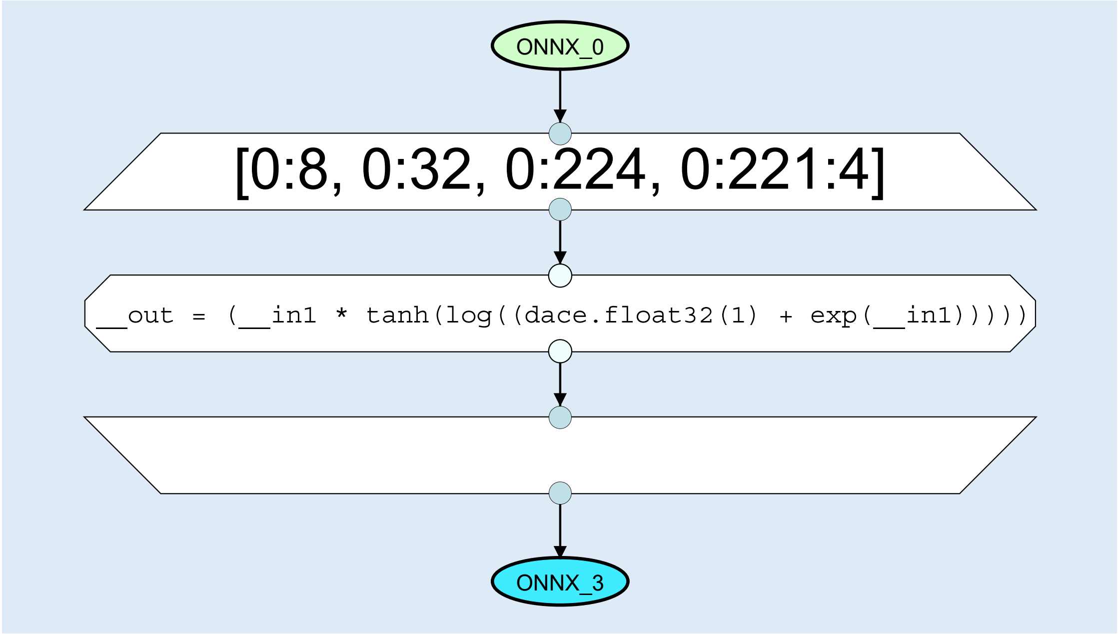
```python
@python_pure_op_implementation
def Softplus(X, Y):
  Y[:] = numpy.log(1 + numpy.exp(X))
```

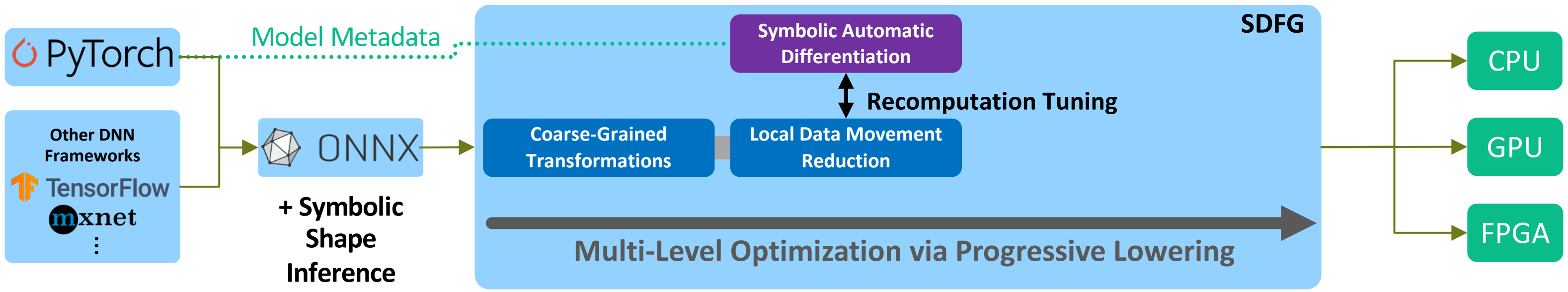
```python
def forward(self, x):
  y = F.softplus(x)
  y = torch.tanh(y)
  return x * y
```



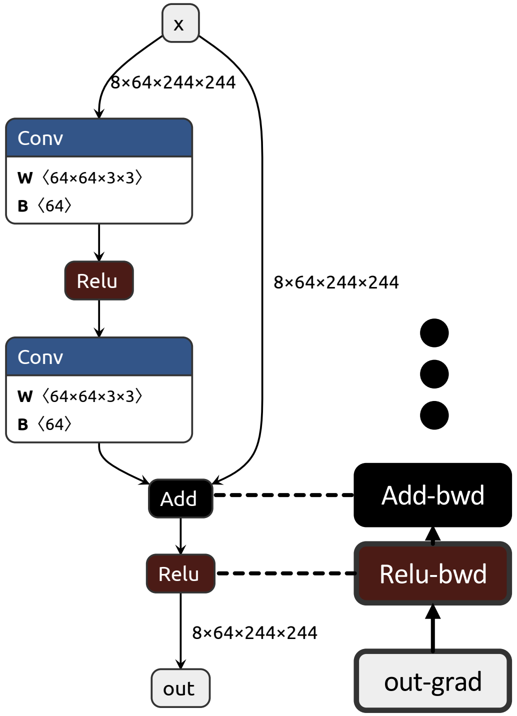


Map Fusion

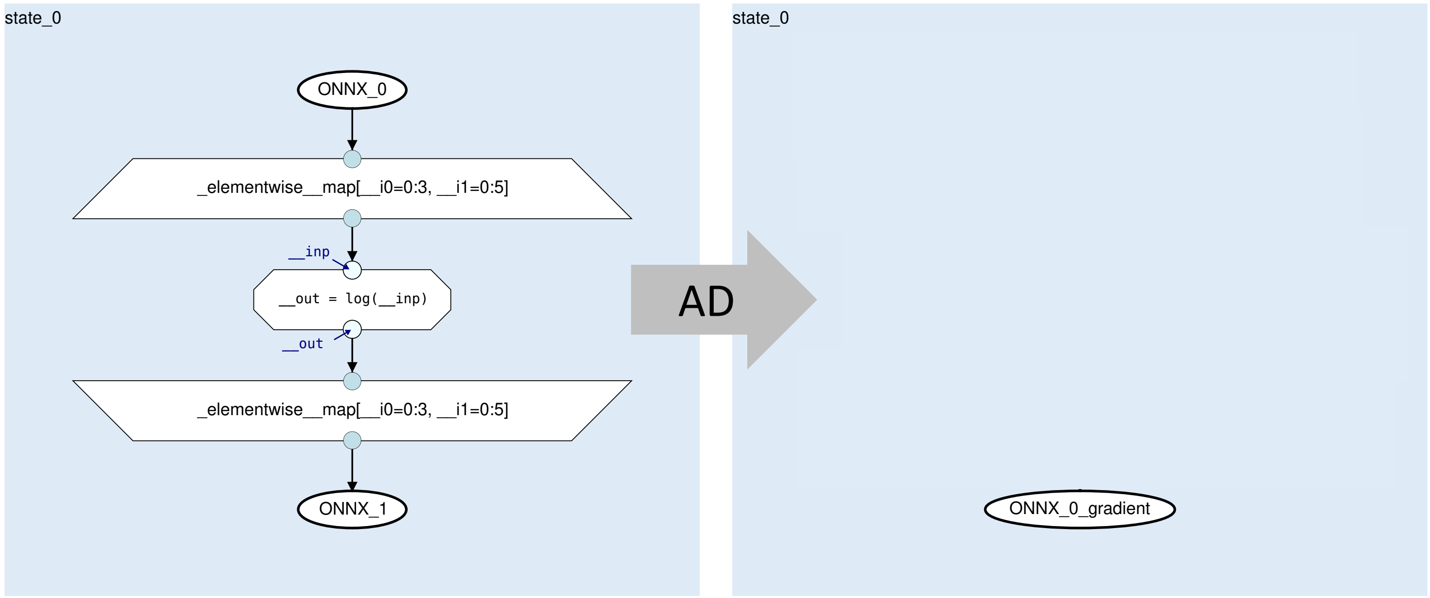




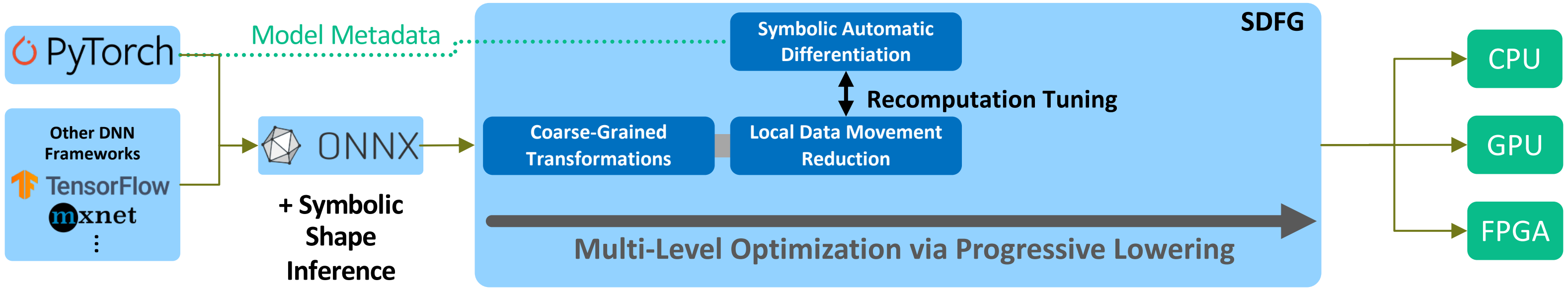
Operator-Level AD



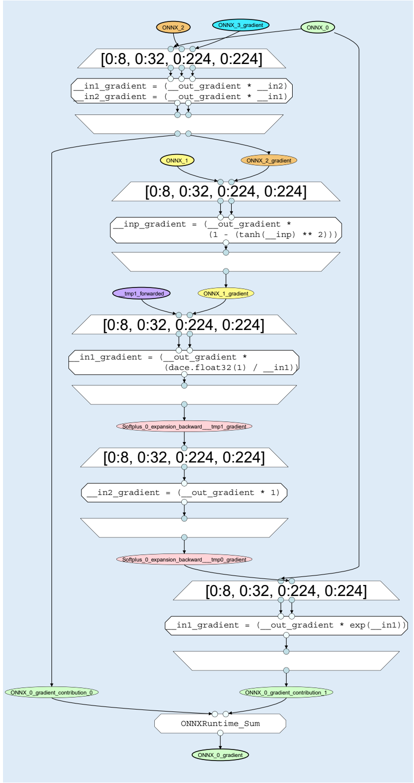
DaCeML: data-centric, Symbolic AD



Data-movement information enables automatic backward-kernel synthesis!

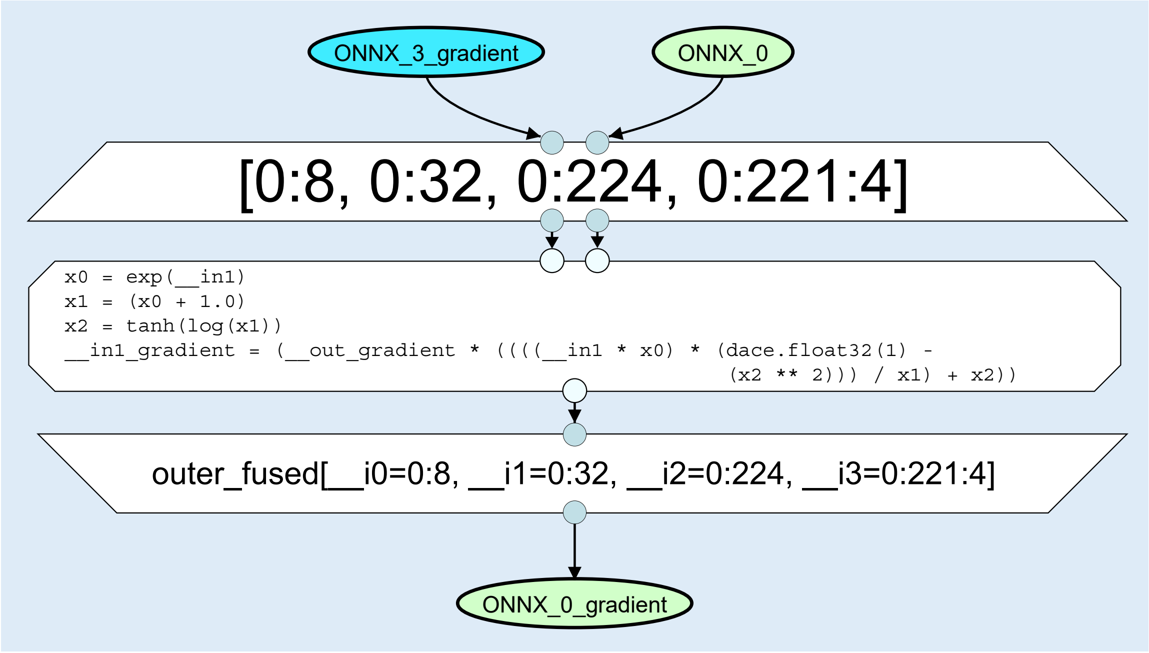


Post-AD Optimization



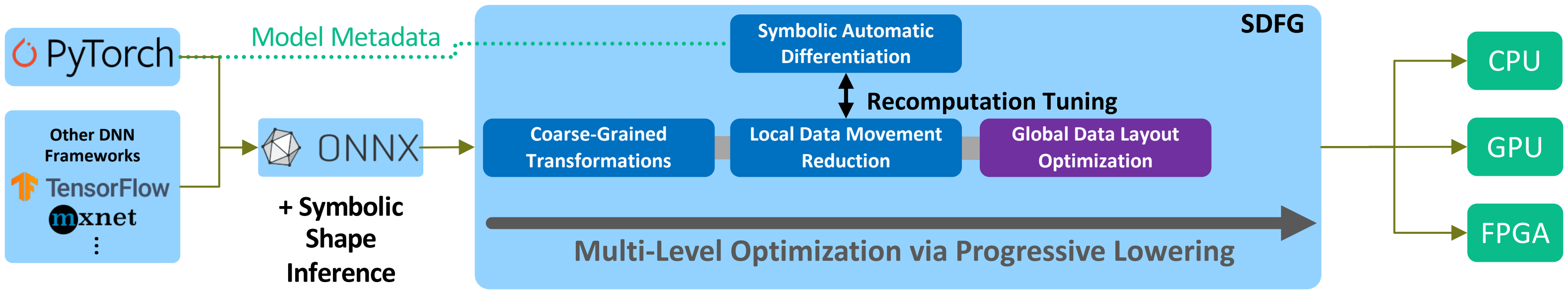
Intermediate Memory: 134.75MiB

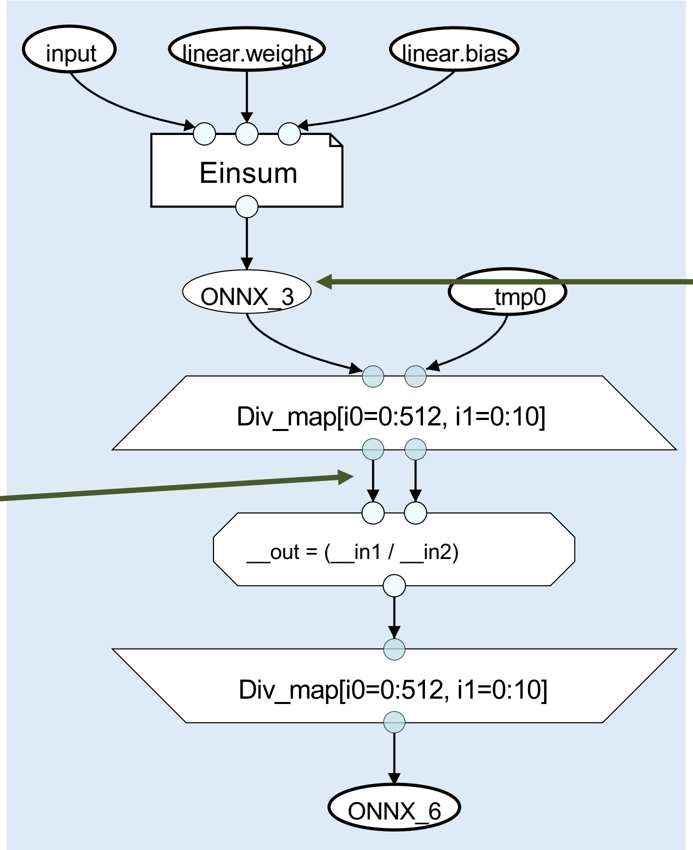
Pre-AD Optimization



Intermediate Memory: 0MiB

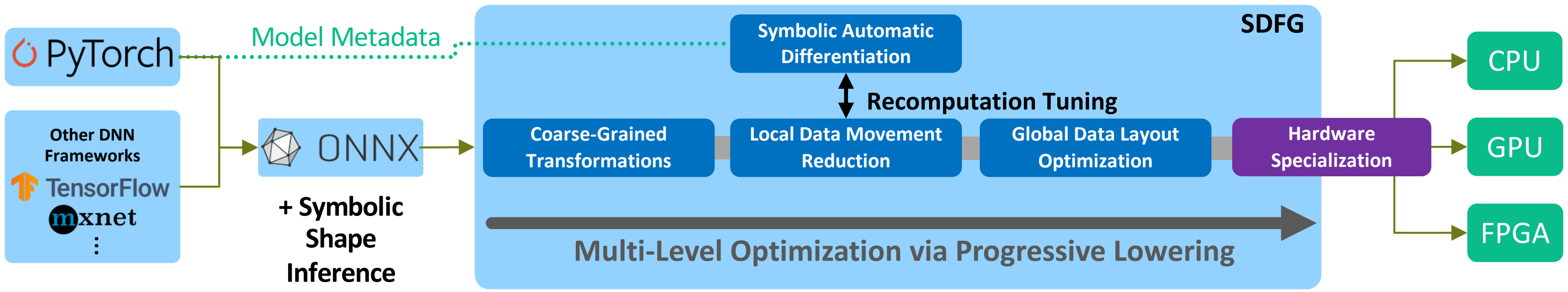
No other framework can perform this type of low-level pre-AD optimization





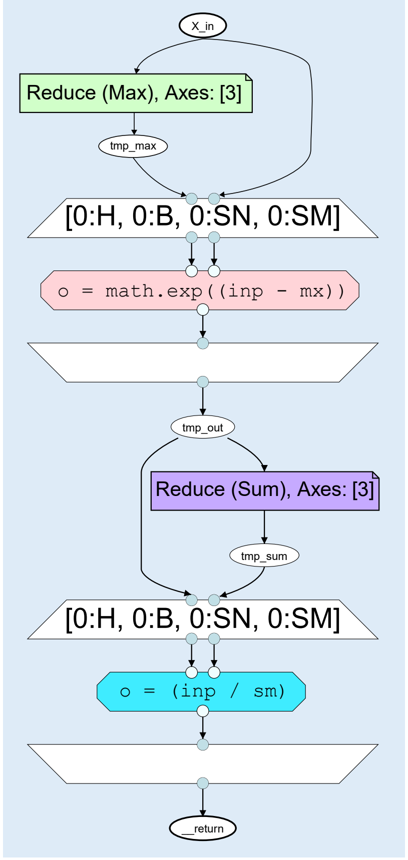
Prune layouts
That don't permit
lowering to BLAS

Memlet propagation
& codegen is data-
layout aware

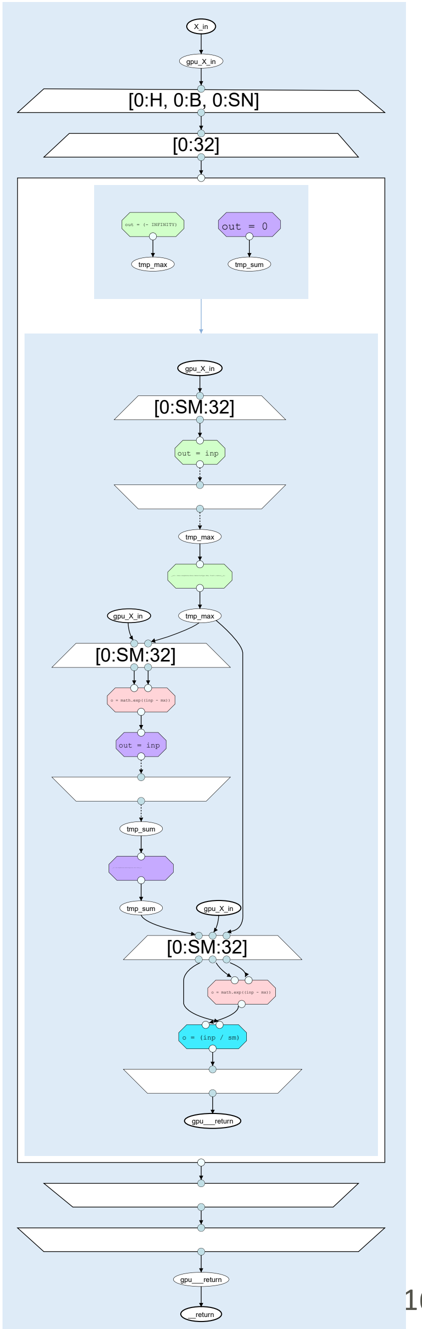


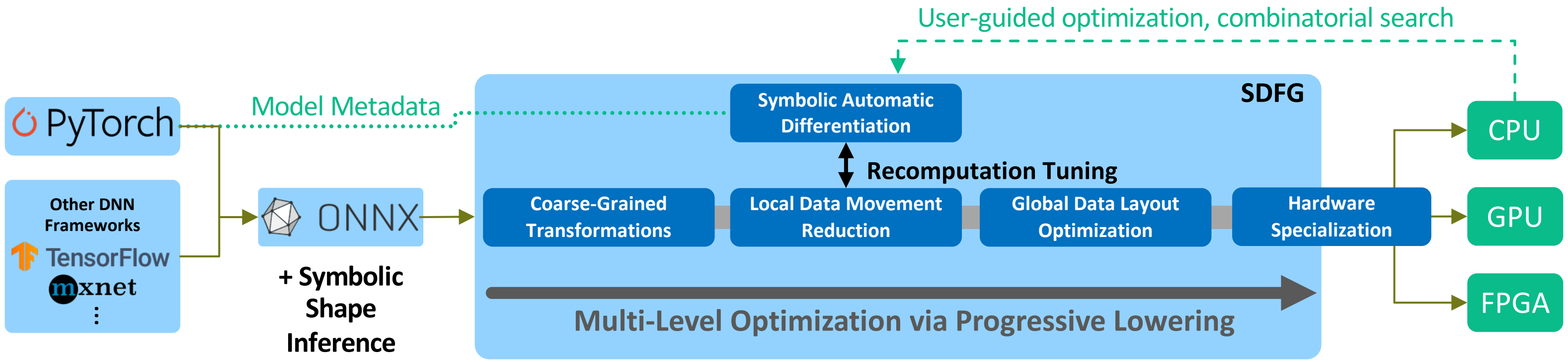

```python
# Operator implementation in NumPy
def Softmax(input, output):
  max = input.max(axis=axis, keepdims=True)
  exp = np.exp(input - max)
  sum = exp.sum(axis=axis, keepdims=True)
  output[:] = exp / sum
```

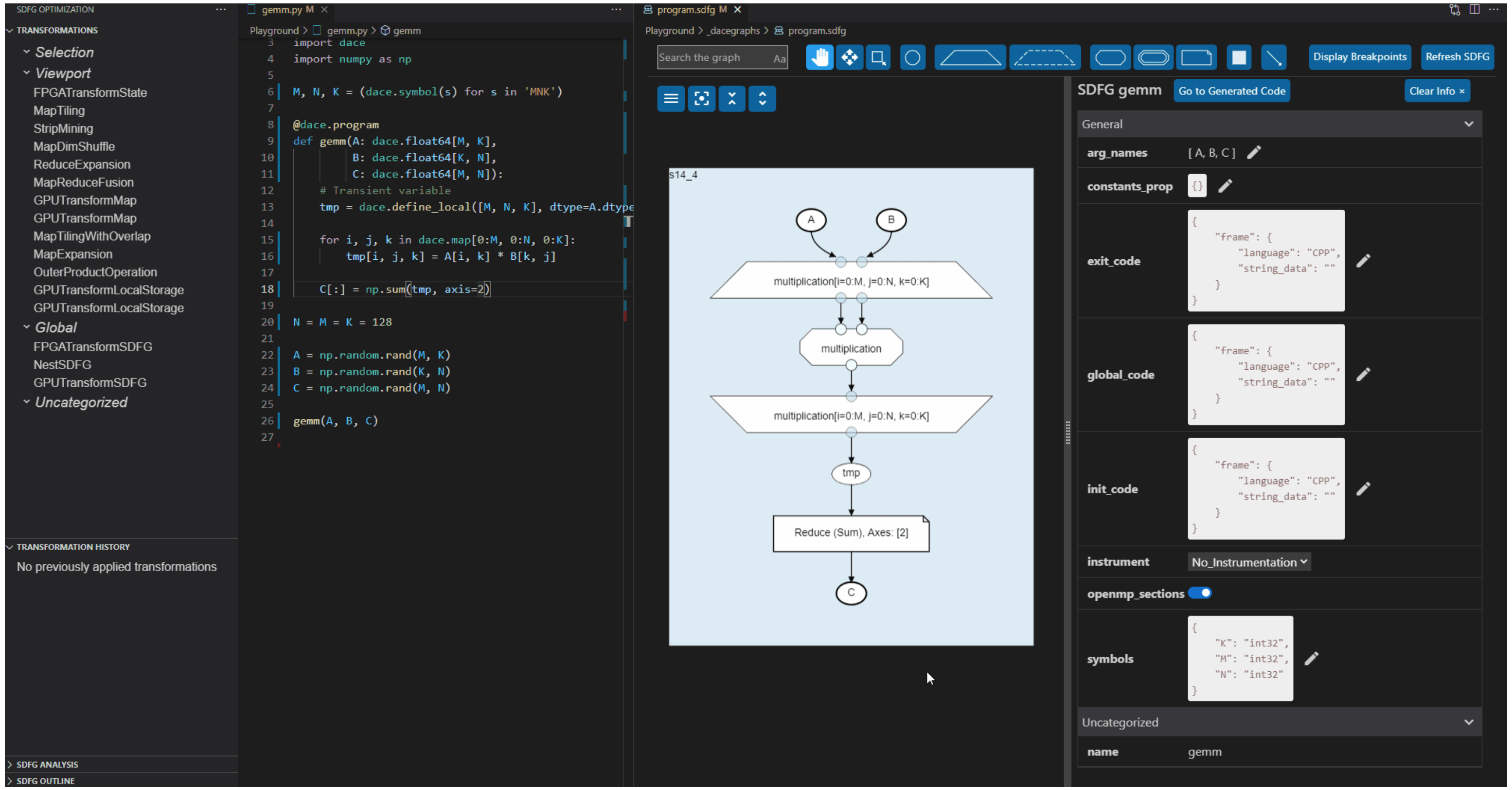
Naive Lowering

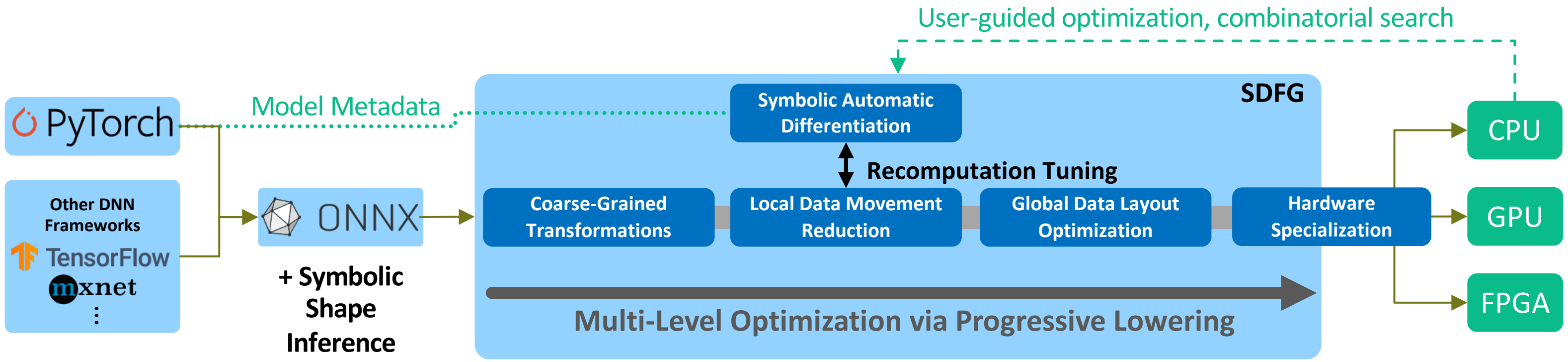


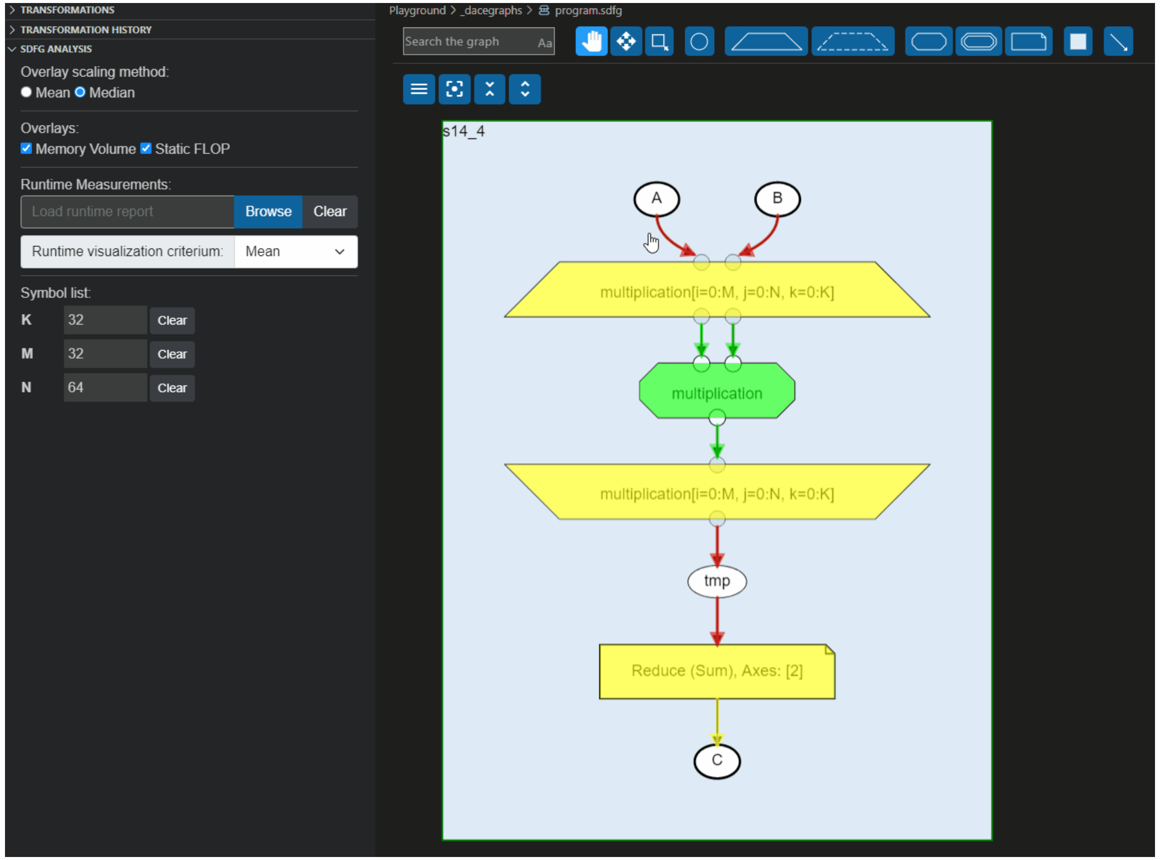
WarpTiling



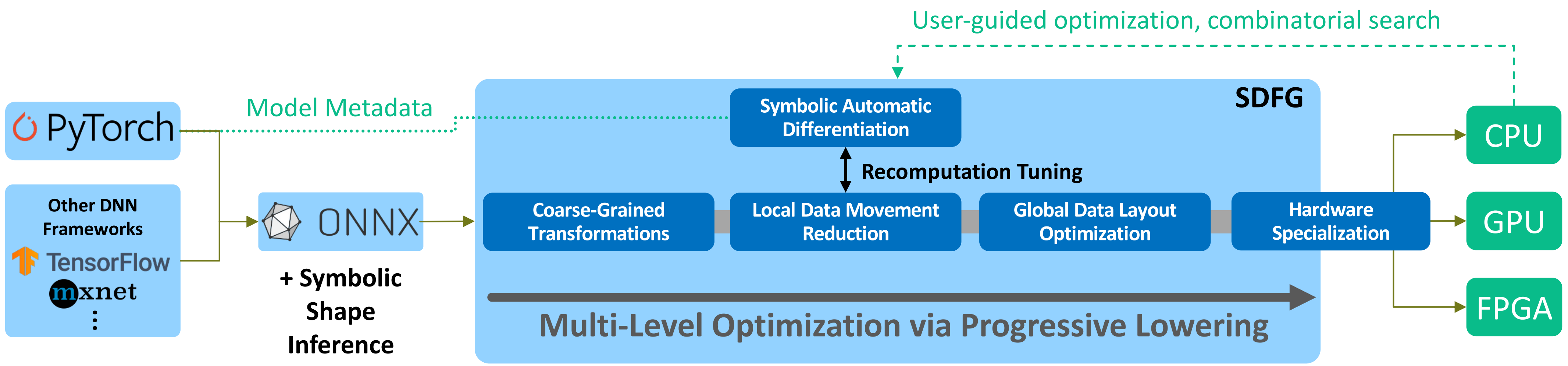

User-guided optimization, combinatorial search
PyTorch
Model Metadata
SDFG
Symbolic Automatic Differentiation
Recomputation Tuning
Other DNN Frameworks
TensorFlow
mxnet
ONNX
+ Symbolic Shape Inference
Coarse-Grained Transformations
Local Data Movement Reduction
Global Data Layout Optimization
Hardware Specialization
CPU
GPU
FPGA
Multi-Level Optimization via Progressive Lowering


SDFG OPTIMIZATION
TRANSFORMATIONS
Selection
Viewport
FPGATransformState
MapTiling
StripMining
MapDimShuffle
ReduceExpansion
MapReduceFusion
GPUTransformMap
GPUTransformMap
MapTilingWithOverlap
MapExpansion
OuterProductOperation
GPUTransformLocalStorage
GPUTransformLocalStorage
Global
FPGATransformSDFG
NestSDFG
GPUTransformSDFG
Uncategorized
TRANSFORMATION HISTORY
No previously applied transformations
SDFG ANALYSIS
SDFG OUTLINE
gemm.py
import dace
import numpy as np
M, N, K = (dace.symbol(s) for s in 'MNK')
@dace.program
def gemm(A: dace.float64[M, K],
B: dace.float64[K, N],
C: dace.float64[M, N]):
# Transient variable
tmp = dace.define_local([M, N, K], dtype=A.dtype
for i, j, k in dace.map[0:M, 0:N, 0:K]:
tmp[i, j, k] = A[i, k] * B[k, j]
C[:] = np.sum(tmp, axis=2)
N = M = K = 128
A = np.random.rand(M, K)
B = np.random.rand(K, N)
C = np.random.rand(M, N)
gemm(A, B, C)
program.sdfg
Display Breakpoints
Refresh SDFG
SDFG gemm
Go to Generated Code
Clear Info
multiplication[i=0:M, j=0:N, k=0:K]
multiplication
tmp
Reduce (Sum), Axes: [2]
General
arg_names
constants_prop
exit_code
global_code
init_code
instrument
No_Instrumentation
openmp_sections
symbols
Uncategorized
name
gemm

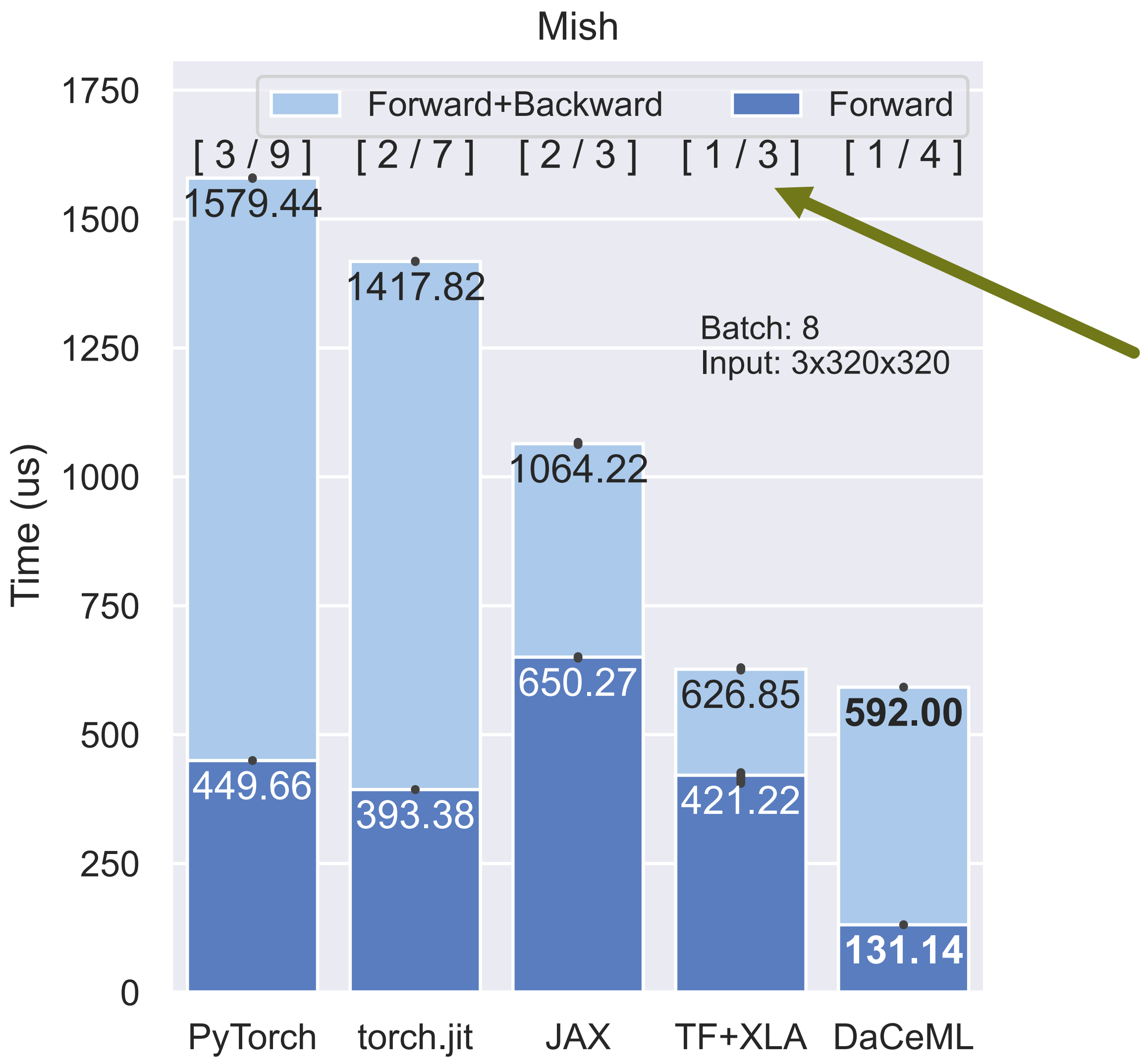




Ben-Nun et al. "Stateful Dataflow Multigraphs: A data-centric model for performance portability on heterogeneous architectures." (2019).

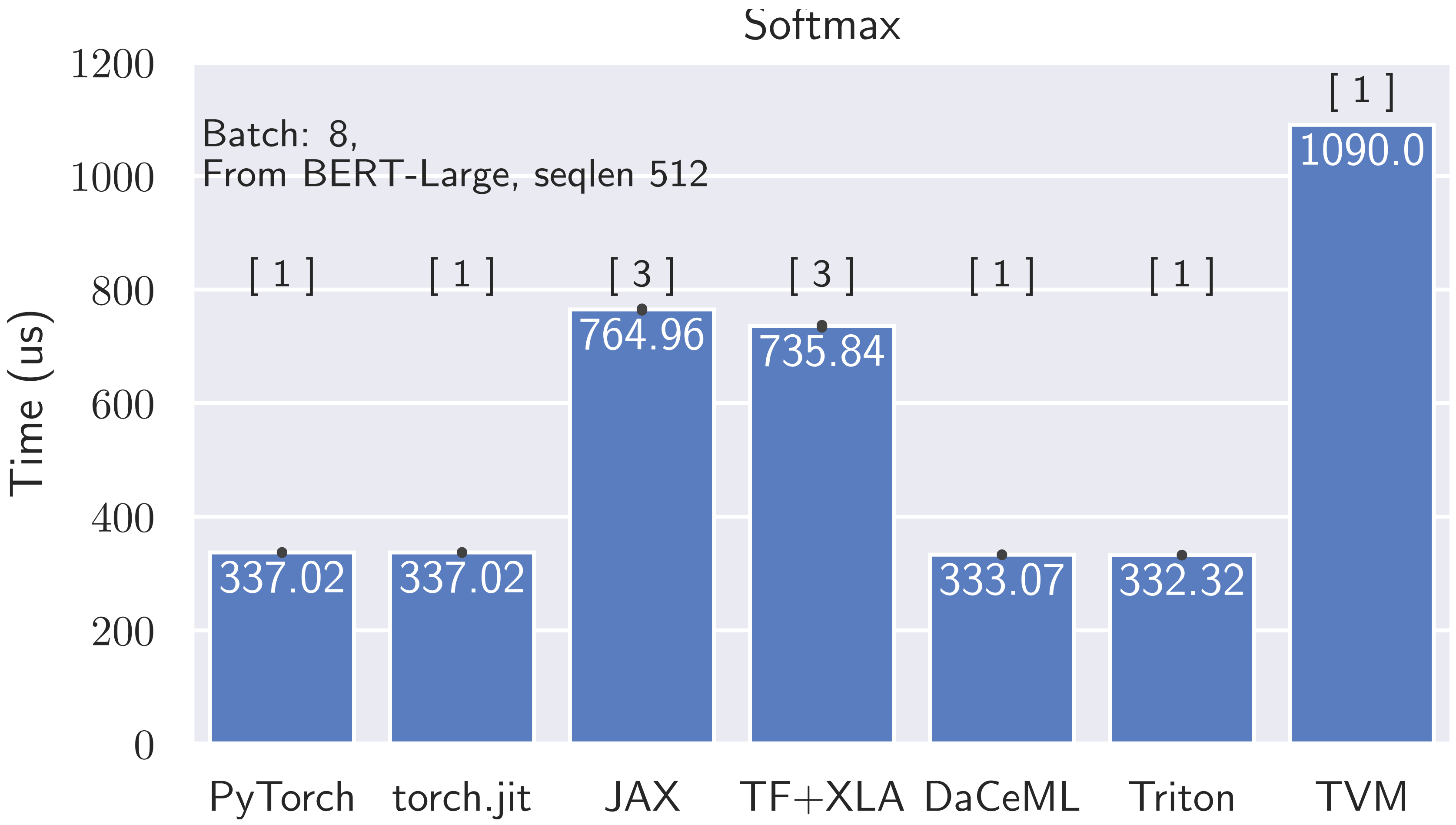

User-guided optimization, combinatorial search
PyTorch
Model Metadata
Other DNN Frameworks
TensorFlow
mxnet
⋮
ONNX
+ Symbolic Shape Inference
SDFG
Symbolic Automatic Differentiation
Recomputation Tuning
Coarse-Grained Transformations
Local Data Movement Reduction
Global Data Layout Optimization
Hardware Specialization
Multi-Level Optimization via Progressive Lowering
CPU
GPU
FPGA

Results - Pre-AD Fusion (Mish)



- XLA manages to produce a single fused kernel
- But fails to eliminate global loads/stores introduced to stash intermediate values

Results – Softmax

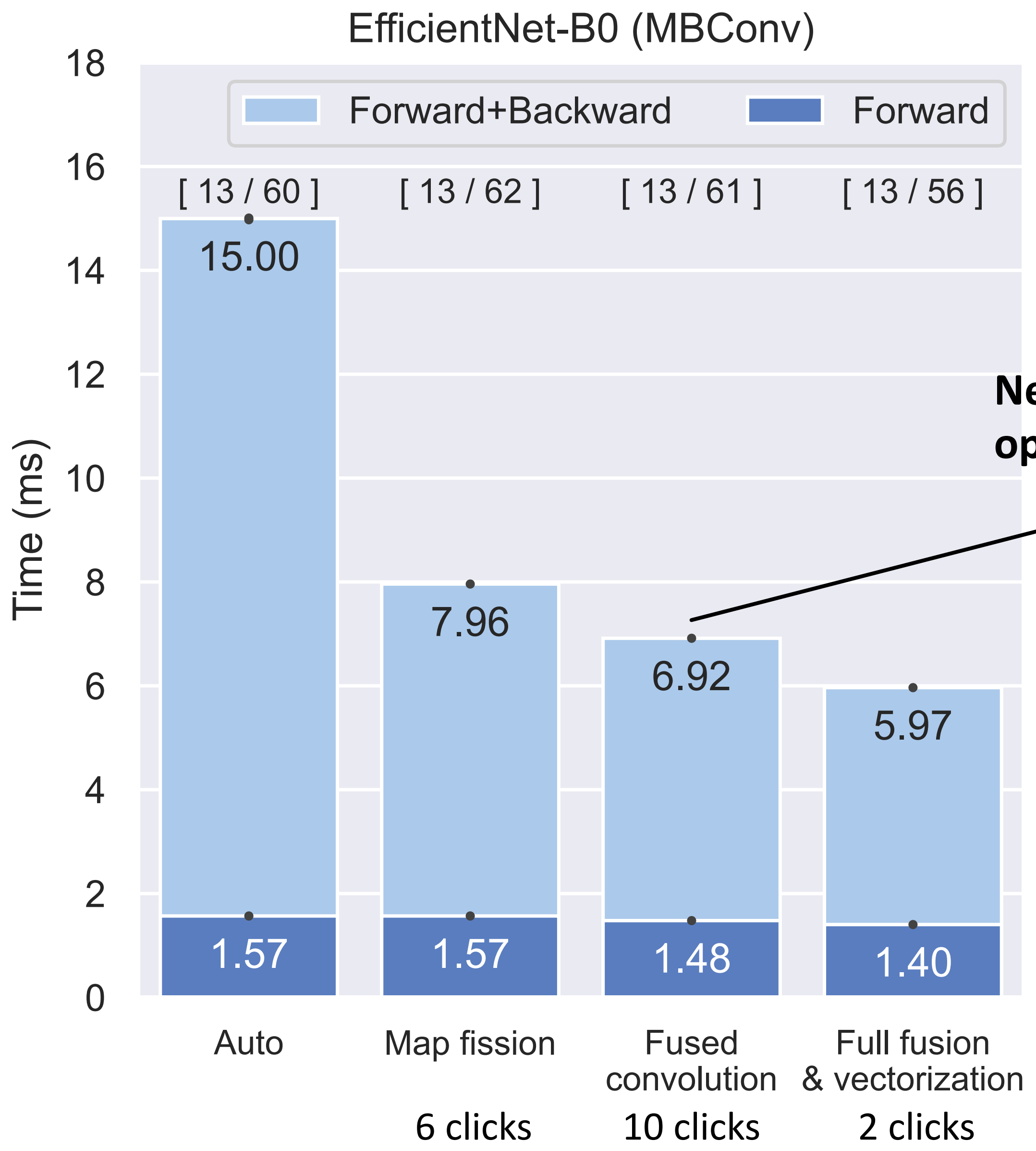


Transformation recipe **matches hand written kernels** and generalizes to other operators (e.g. Layer Normalization)

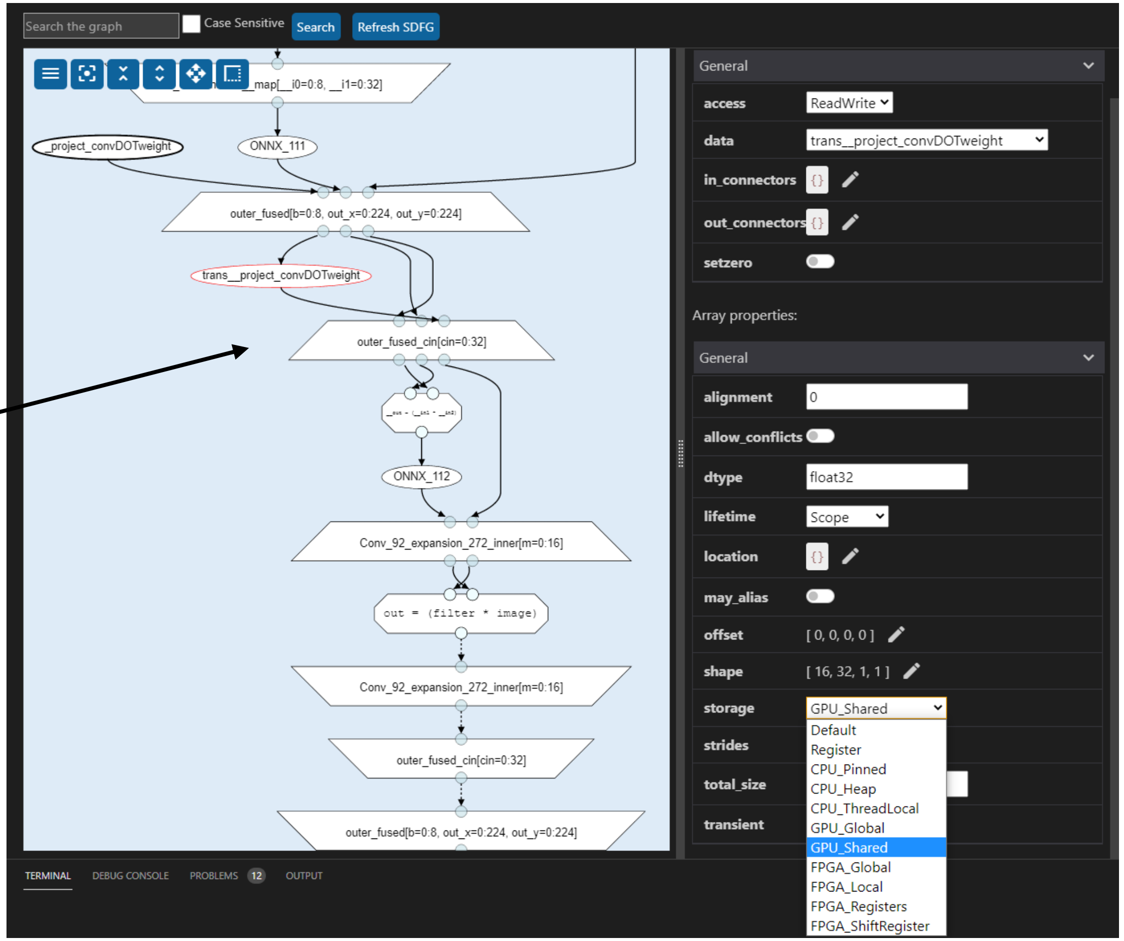
Results – Automatic Optimization

		PyTorch		torch.jit		JAX		TF+XLA		DaCeML	
		→	⇄	→	⇄	→	⇄	→	⇄	→	⇄
Automatic	ResNet-50	14.55	32.04	**9.98**	**31.94**	14.17	33.93	12.33	35.57	10.03	32.45
	Wide ResNet-50-2	22.50	70.94	22.45	70.83	40.49	98.13	32.79	99.06	**20.62**	**67.99**
	MobileNet V2	9.98	18.45	6.22	15.53	—	—	7.42	20.29	**4.74**	**14.77**
	EfficientNet	2.05	6.90	2.04	6.94	2.39	7.40	**1.54**	**6.37**	1.57	15.00
	MLP Mixer	1.63	**3.65**	**1.36**	3.66	1.77	4.01	—	—	1.48	4.25
	FCN8s	46.85	158.42	46.82	**158.40**	—	—	—	—	**45.97**	166.30
	WaveNet	23.21	46.39	**18.67**	41.49	—	—	—	—	26.16	**41.07**
	$\text{BERT}_{\text{LARGE}}$ (single)	11.05	31.76	11.05	31.82	**10.93**	**29.94**	11.14	38.73	11.44	32.98
	$\text{BERT}_{\text{LARGE}}$ (mixed)	2.94	8.18	**2.92**	8.20	3.19	**8.11**	3.80	10.76	3.34	9.25
	DLRM	118.07	126.55	**117.38**	126.83	—	—	—	—	117.69	**126.42**

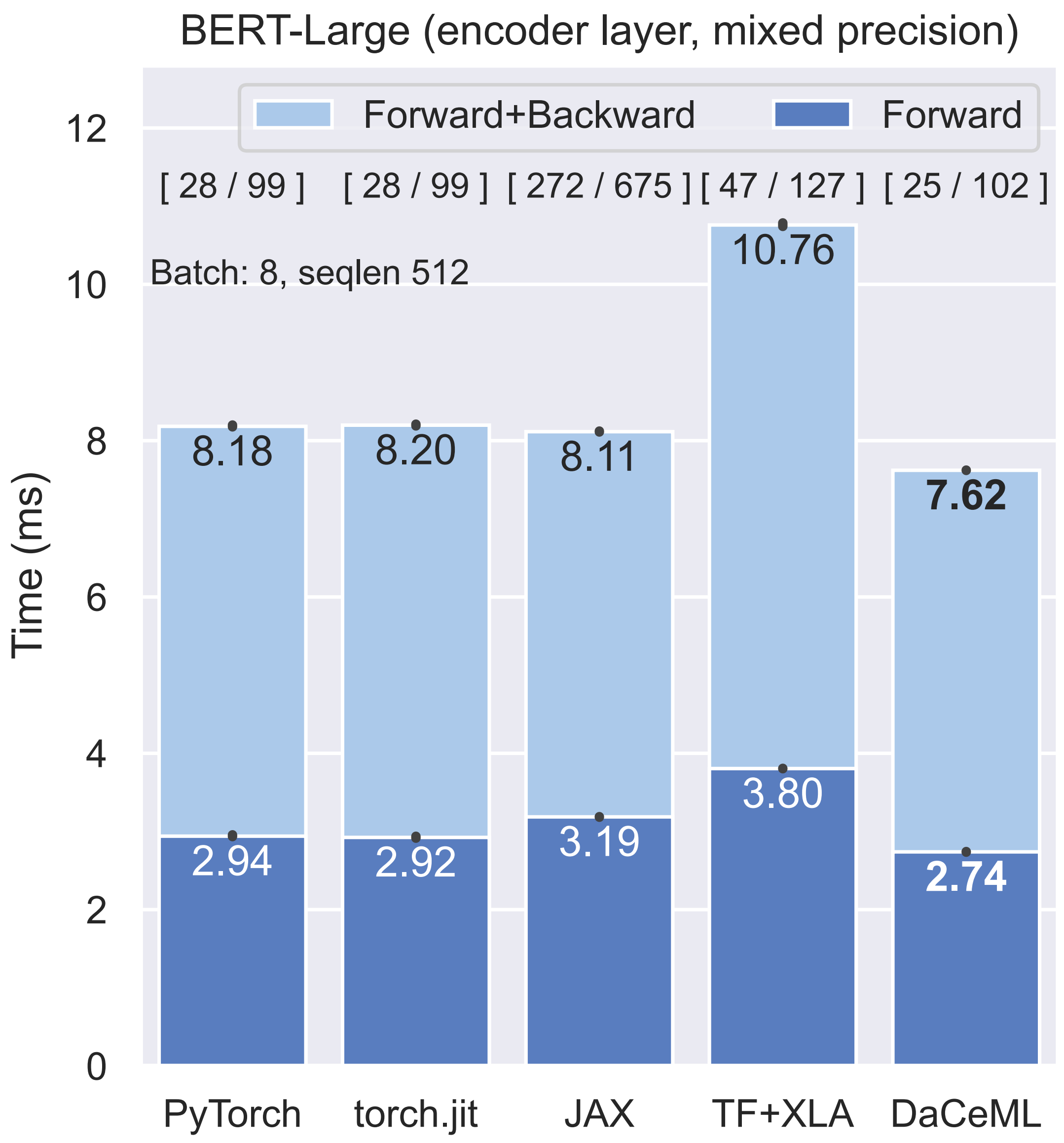
Results – Guided Optimization

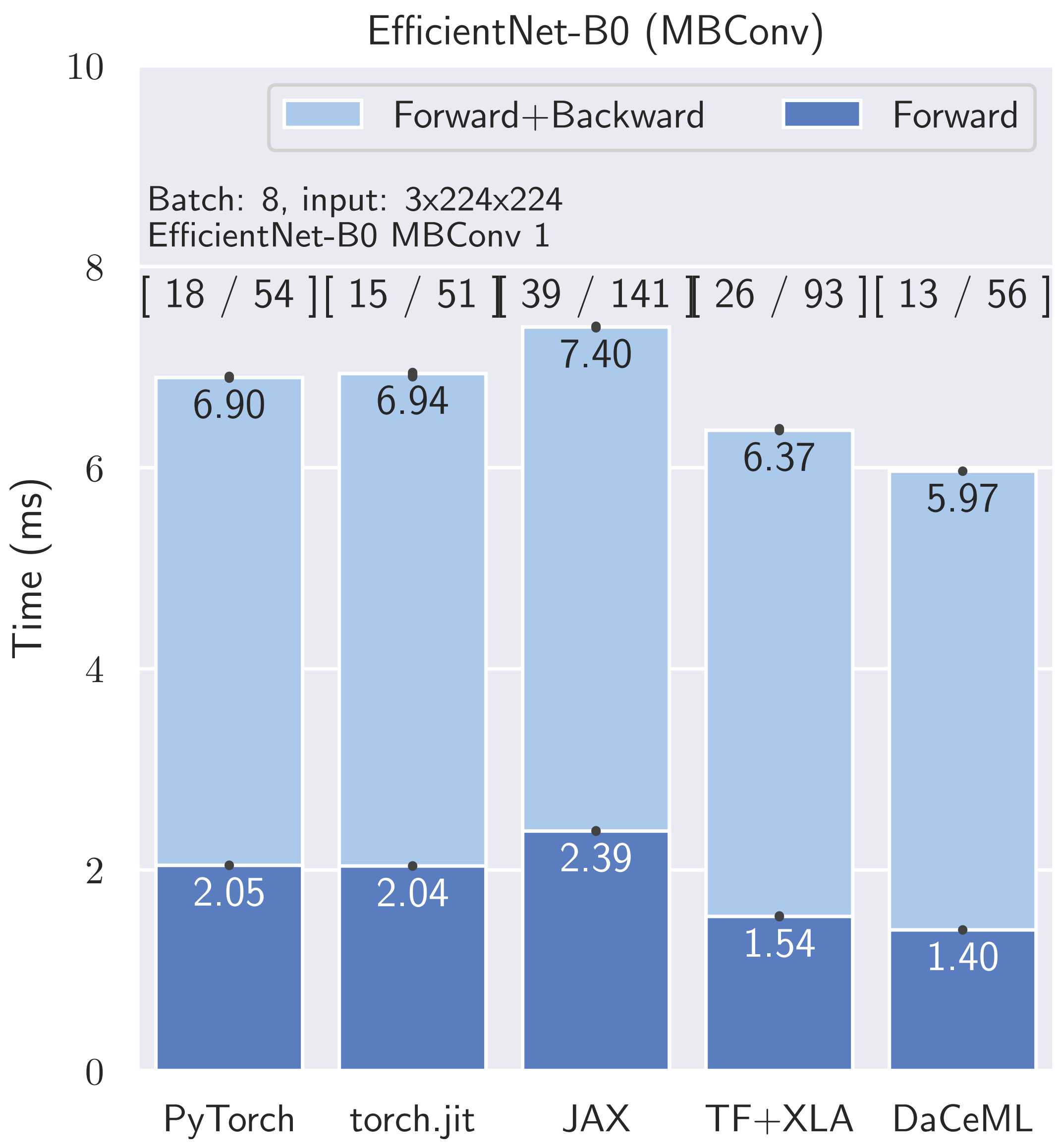


New fusion opportunity

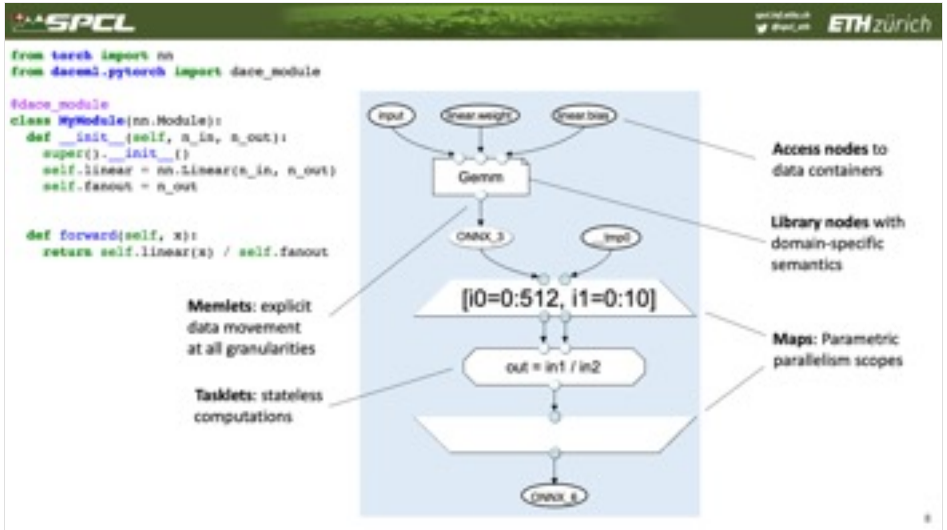


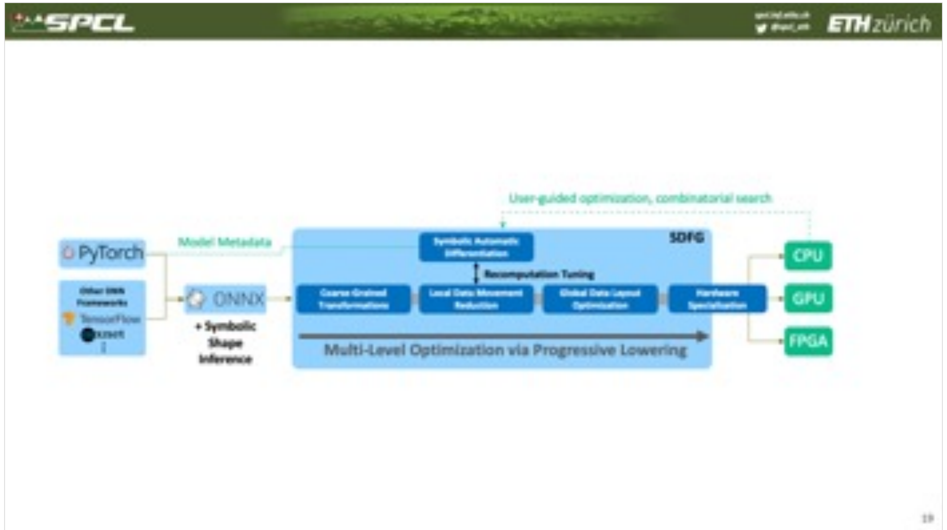
Results – Guided Optimization

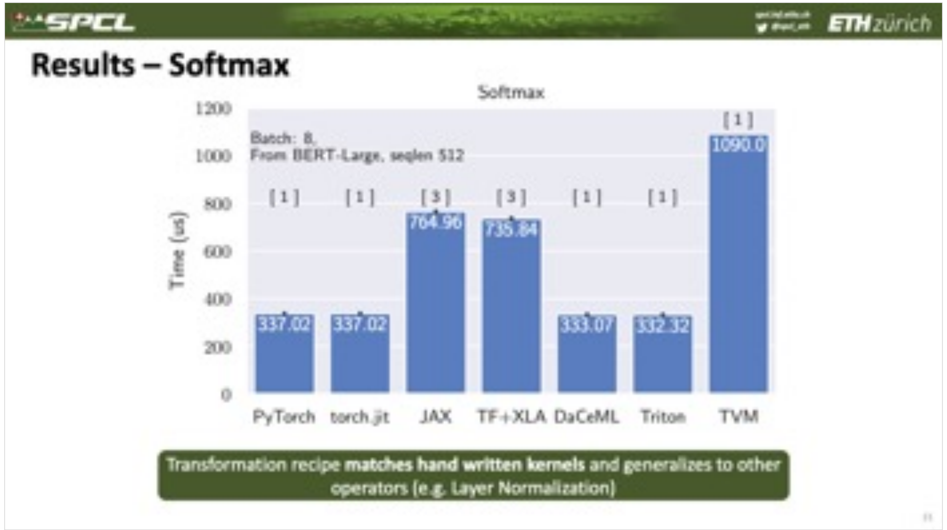




Summary







Results – Models

		PyTorch		torch.jit		JAX		TF+XLA		DaCeML	
		→	⇄	→	⇄	→	⇄	→	⇄	→	⇄
Automatic	ResNet-50	14.55	32.04	**9.98**	**31.94**	14.17	33.93	12.33	35.57	10.03	32.45
	Wide ResNet-50-2	22.50	70.94	22.45	70.83	40.49	98.13	32.79	99.06	**20.62**	**67.99**
	MobileNet V2	9.98	18.45	6.22	15.53	—	—	7.42	20.29	**4.74**	**14.77**
	EfficientNet	2.05	6.90	2.04	6.94	2.39	7.40	**1.54**	**6.37**	1.57	15.00
	MLP Mixer	1.63	**3.65**	**1.36**	3.66	1.77	4.01	—	—	1.48	4.25
	FCN8s	46.85	158.42	46.82	**158.40**	—	—	—	—	**45.97**	166.30
	WaveNet	23.21	46.39	**18.67**	41.49	—	—	—	—	26.16	**41.07**
	$\mathrm{BERT}_{\mathrm{LARGE}}$ (single)	11.05	31.76	11.05	31.82	**10.93**	**29.94**	11.14	38.73	11.44	32.98
	$\mathrm{BERT}_{\mathrm{LARGE}}$ (mixed)	2.94	8.18	**2.92**	8.20	3.19	**8.11**	3.80	10.76	3.34	9.25
	DLRM	118.07	126.55	**117.38**	126.83	—	—	—	—	117.69	**126.42**
Guided	EfficientNet	2.05	6.90	2.04	6.94	2.39	7.40	1.54	6.37	**1.40**	**5.97**
	$\mathrm{BERT}_{\mathrm{LARGE}}$ (mixed)	2.94	8.18	2.92	8.20	3.19	8.11	3.80	10.76	**2.74**	**7.62**

github.com/spcl/daceml