Project page: https://sites.google.com/nvidia.com/conv-tt-lstm/home

- Project contributors: Jiahao Su* (UMD), Wonmin Byeon* (NVIDIA), Jean Kossaifi (NVIDIA), Furong Huang (UMD), Jan Kautz (NVIDIA), Animashree Anandkumar (NVIDIA)
- Code Optimization contributor: Sangkug Lym (NVIDIA)

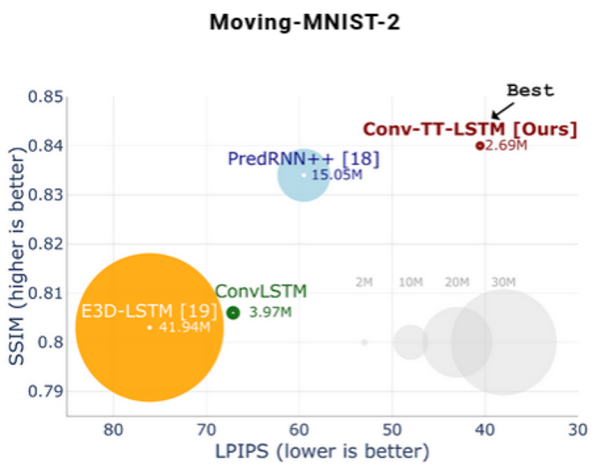
NVIDIA

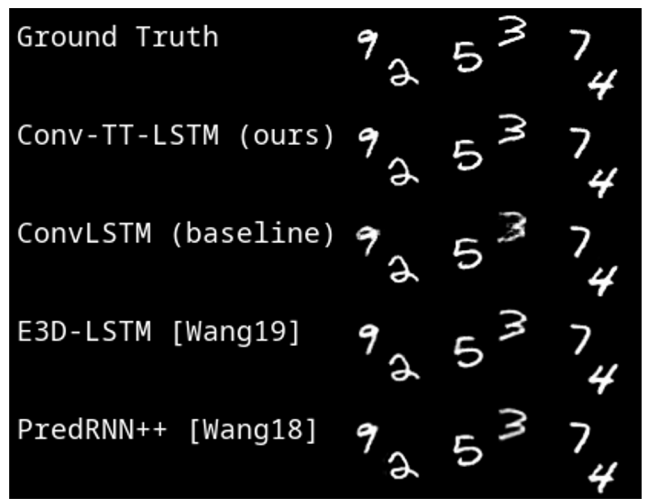
Mixed Precision Training for Convolutional Tensor-Train LSTM

Wonmin Byeon

Applications

Video Prediction

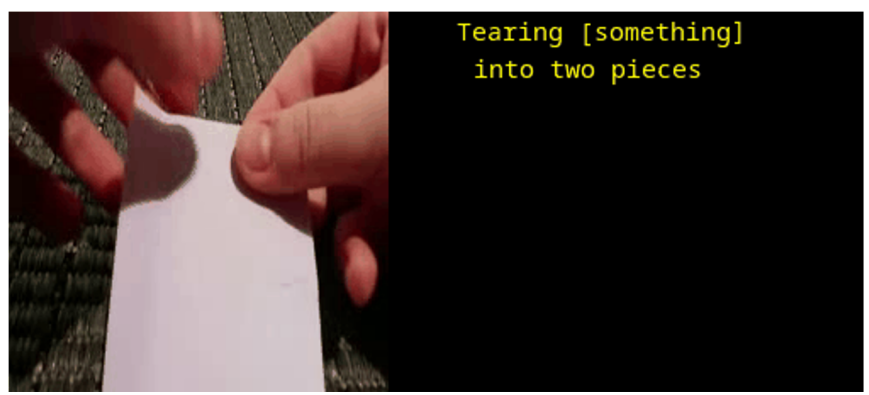




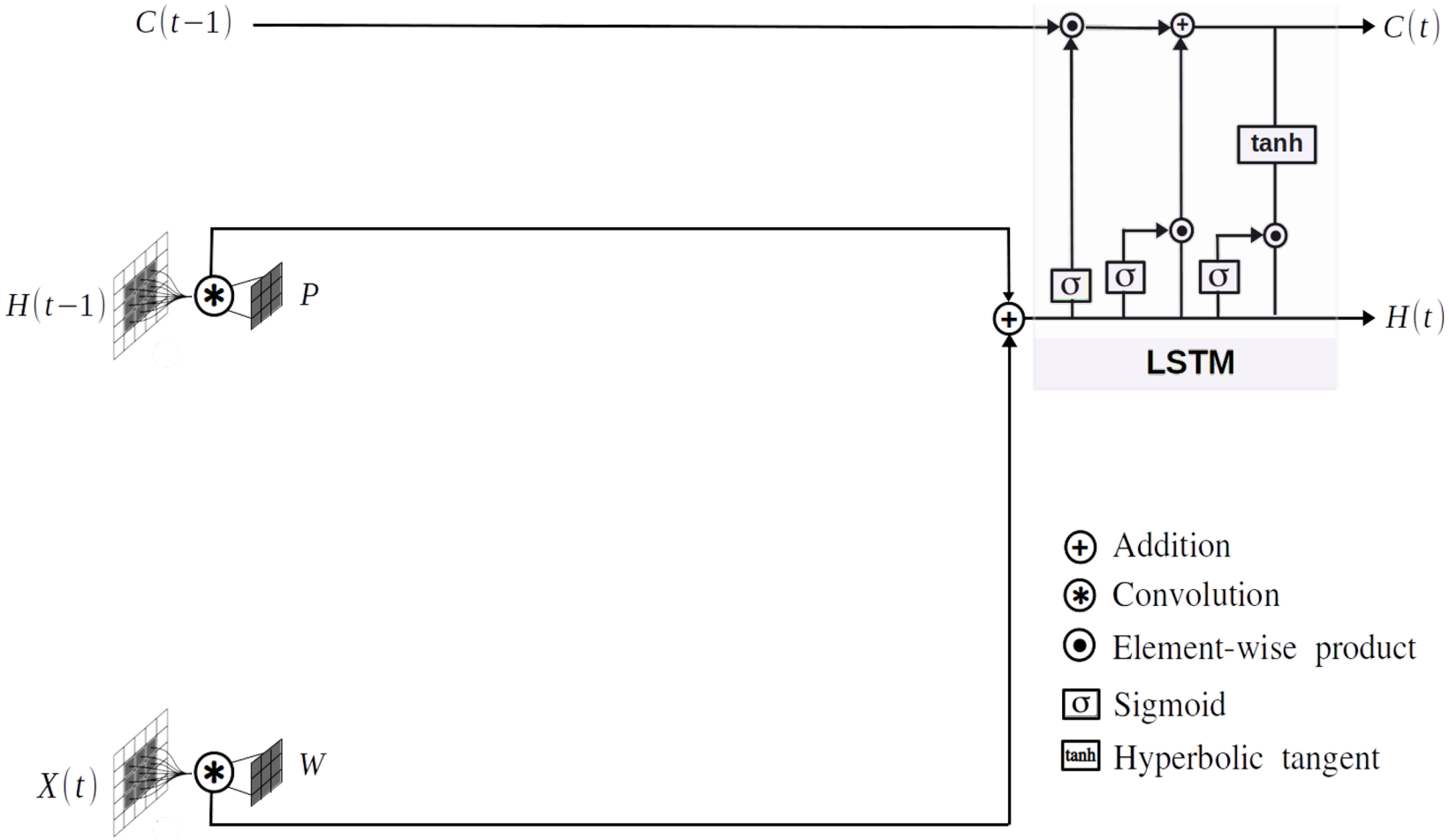
Early Activity Recognition

Model	Input Ratio	
	Front 25%	Front 50%
3D-CNN*	9.11	10.30
E3D-LSTM* [7]	14.59	22.73
3D-CNN	13.26	20.72
ConvLSTM	15.46	21.97
Conv-TT-LSTM (ours)	**19.53**	**30.05**

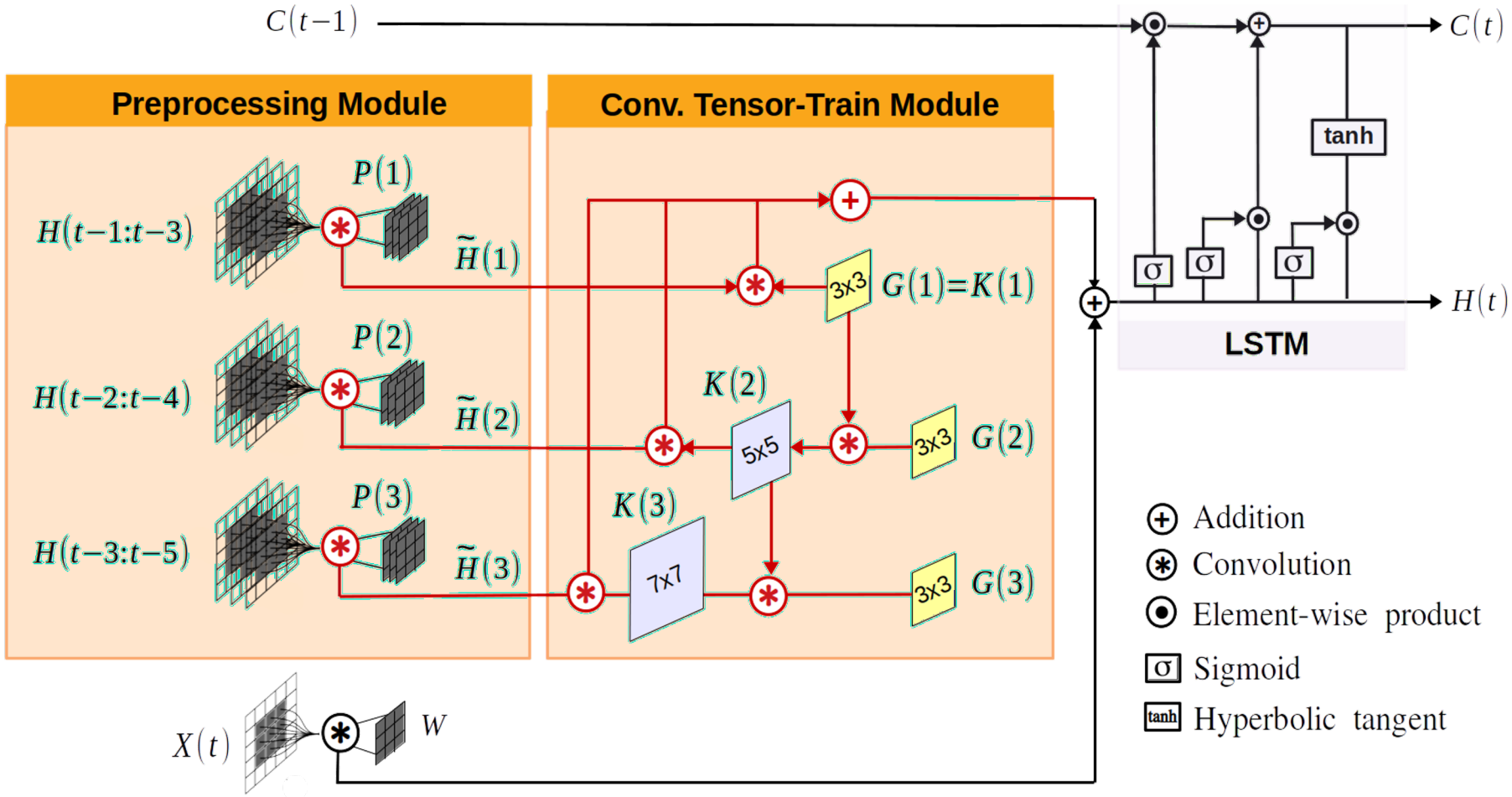
Table 2: **Early activity recognition on the Something-Something V2 dataset** using 41 categories as [7]. (*) indicates the result by [7].



Convolutional LSTM



Convolutional Tensor-Train LSTM



Application: video prediction
Machine: V100 x 8, 16GB
12 Conv. Tensor-Train LSTM layers
Input/output resolution: 128x128

Optimization Tricks

Optimization	batch size (per GPU)	Time (s)	Incremental speedup	Total speedup
Baseline	1	1.14	-	1x
#1. Enabling AMP	2	0.68	1.67x	1.67x
#2. GPU Optimization	2	0.645	1.05x	1.77x
#3. Activation Checkpointing	4	0.48	1.34x	2.38x

Application: video prediction
Machine: V100 x 8, 16GB
12 Conv. Tensor-Train LSTM layers
Input/output resolution: 128x128

Optimization Tricks

Optimization	batch size (per GPU)	Time (s)	Incremental speedup	Total speedup
Baseline	1	1.14	-	1x
#1. Enabling AMP	Without performance change!			
#2. GPU Optimization				
#3. Activation Checkpointing	4	0.48	1.34x	**2.38x**

#1. Enabling AMP

```
from apex import amp

model, optimizer = amp.initialize(model, optimizer, opt_level="O1")


with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
if gradient_clipping:
    grad_norm = torch.nn.utils.clip_grad_norm_(
        amp.master_params(optimizer), clipping_threshold)
```

```
 # save the checkpoint
checkpoint_info['amp'] = amp.state_dict()

 # load the checkpoint
amp.load_state_dict(checkpoint['amp'])
```

#2. GPU Optimization

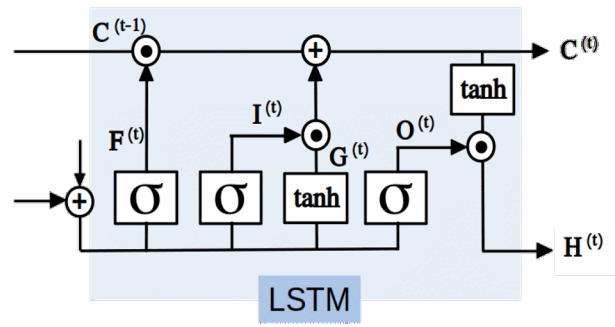
- **Fused Adam: Adam optimizer**

```
from apex.optimizers import FusedAdam

optimizer = FusedAdam(model.parameters(), lr = learning_rate, eps = eps)
```

#2. GPU Optimization

- **Fused optimizer: Adam optimizer**

```
from apex.optimizers import FusedAdam

optimizer = FusedAdam(model.parameters(), lr = learning_rate, eps = eps)
```



- **Fused math kernels: LSTM cell**

```
@torch.jit.script
def fuse_mul_add_mul(f, cell_states, i, g):
    return f * cell_states + i * g
```

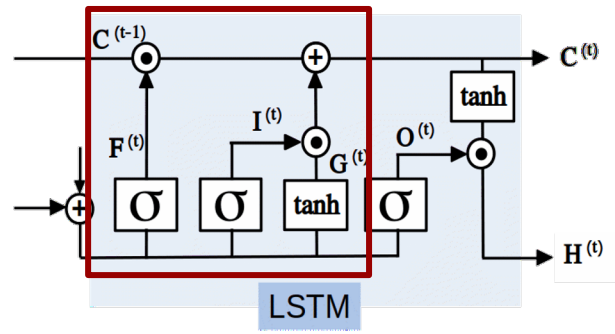
```
self.cell_states  = f * self.cell_states + i * g
```

```
self.cell_states = fuse_mul_add_mul(f, self.cell_states, i, g)
```

#2. GPU Optimization

- **Fused optimizer: Adam optimizer**

```
from apex.optimizers import FusedAdam

optimizer = FusedAdam(model.parameters(), lr = learning_rate, eps = eps)
```



- **Fused math kernels: LSTM cell**

```
@torch.jit.script
def fuse_mul_add_mul(f, cell_states, i, g):
    return f * cell_states + i * g
```

```
self.cell_states  = f * self.cell_states + i * g
```

```
self.cell_states = fuse_mul_add_mul(f, self.cell_states, i, g)
```

#2. GPU Optimization

- **Fused optimizer: Adam optimizer**

```
from apex.optimizers import FusedAdam

optimizer = FusedAdam(model.parameters(), lr = learning_rate, eps = eps)
```

- **Fused math kernels: LSTM cell**

```
@torch.jit.script
def fuse_mul_add_mul(f, cell_states, i, g):
    return f * cell_states + i * g
```

```
self.cell_states  = f * self.cell_states + i * g
```

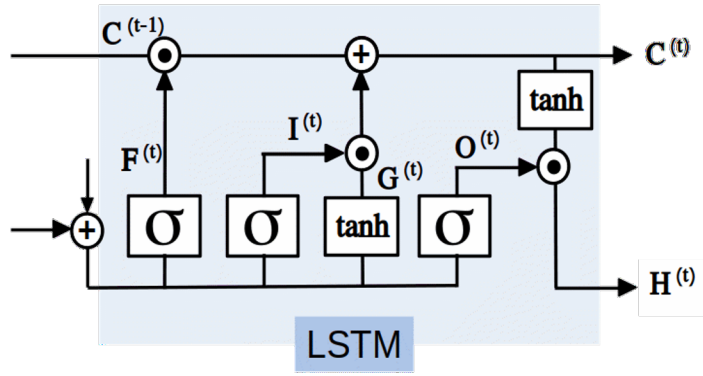
```
self.cell_states = fuse_mul_add_mul(f, self.cell_states, i, g)
```

- **Thread affinity binding: distributed computing**

```
from utils.gpu_affinity import set_affinity

set_affinity(args.local_rank)
```

#3. Activation Checkpointing

```
from torch.utils.checkpoint import checkpoint

def chkpt_blk(cc_i, cc_f, cc_o, cc_g, cell_states):

    i = torch.sigmoid(cc_i)
    f = torch.sigmoid(cc_f)
    o = torch.sigmoid(cc_o)
    g = torch.tanh(cc_g)

    cell_states = fuse_mul_add_mul(f, cell_states, i, g)
    outputs = o * torch.tanh(cell_states)

    return outputs, cell_states
```




```
outputs, self.cell_states = checkpoint(chkpt_blk, cc_i, cc_f, cc_o, cc_g, self.cell_states)
```

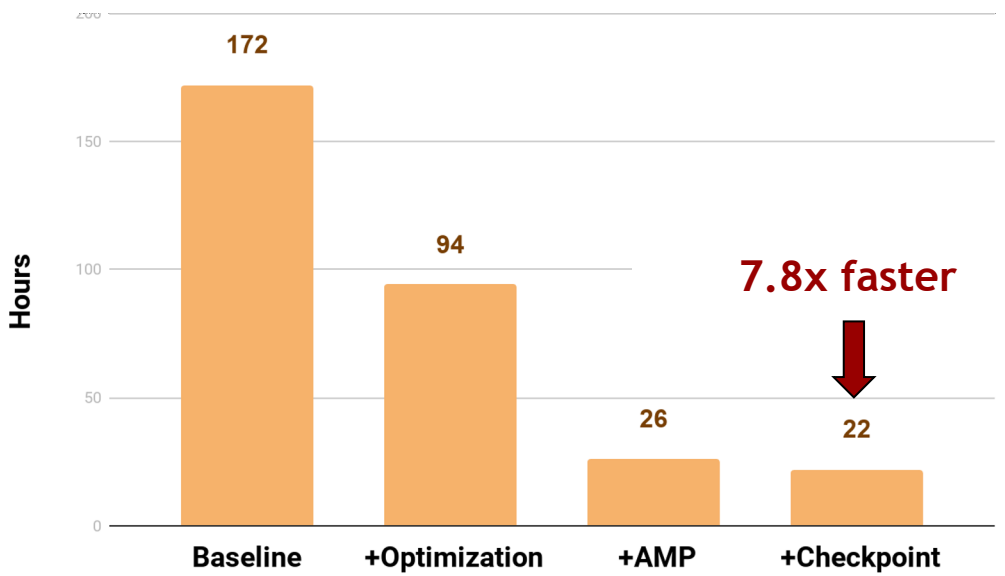
Application: video prediction
Machine: V100 x 8, 16GB
Batch Size: 16 videos
12 Conv. LSTM layers
Input/output image resolution: 128x128

Training

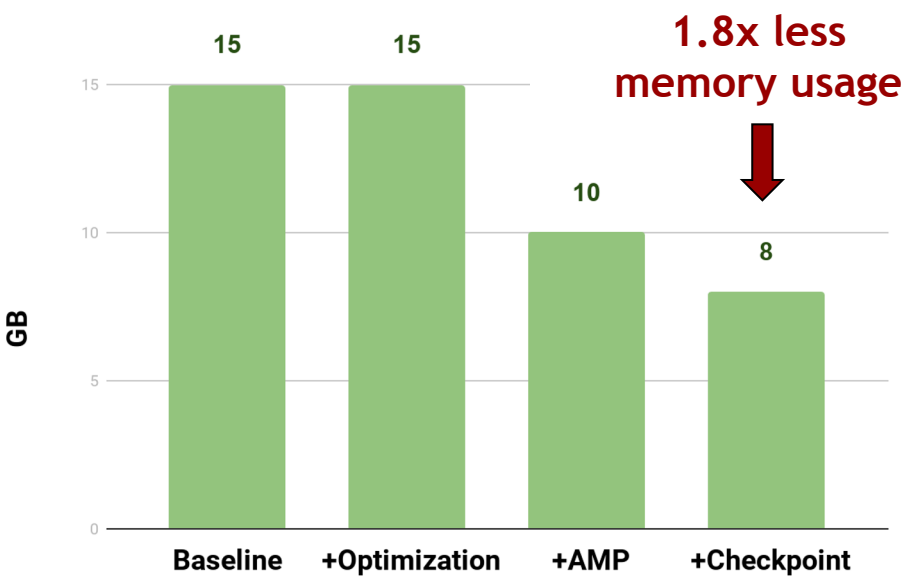
Convolutional LSTM

No performance change!

Speed (*convergence time*)



GPU Memory



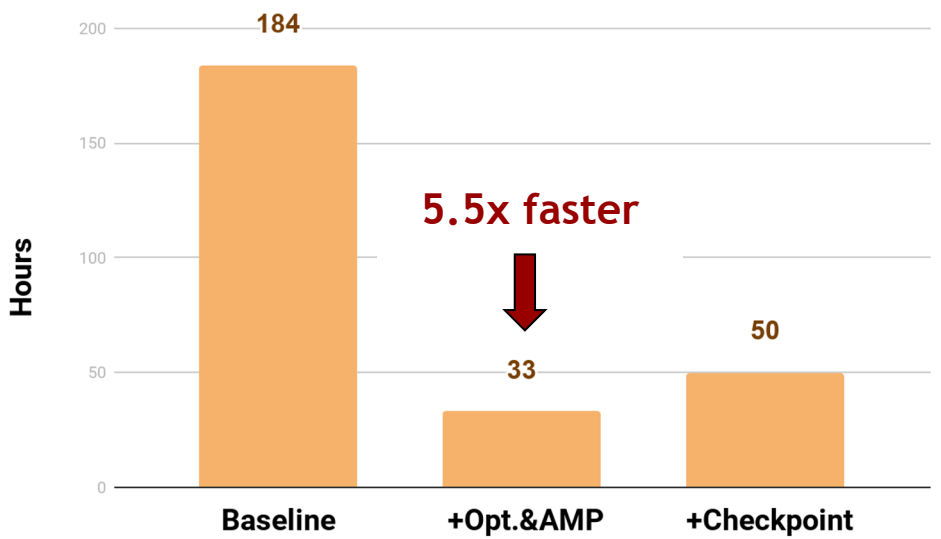
Application: video prediction
Machine: V100 x 8, 16GB
Batch Size: 16 videos
12 Conv. LSTM layers
Input/output image resolution: 128x128

Training

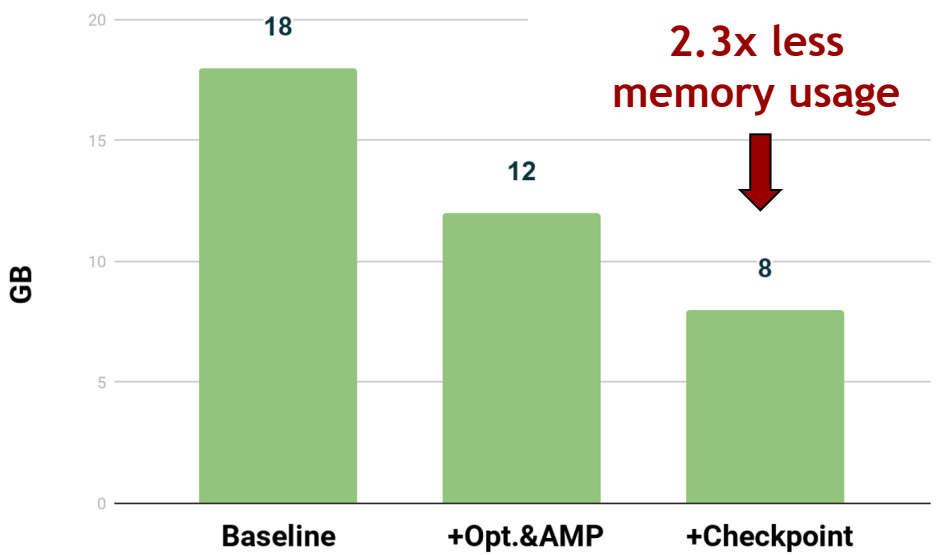
Convolutional Tensor-Train LSTM

No performance change!

Speed *(convergence time)*



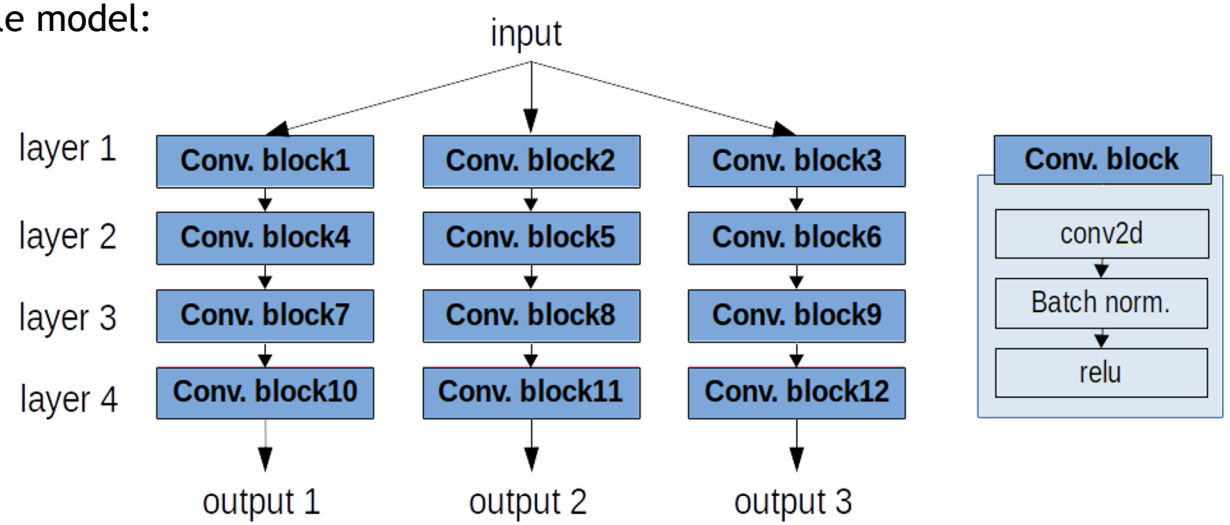
GPU Memory



Model Parallel: Multi-Streams

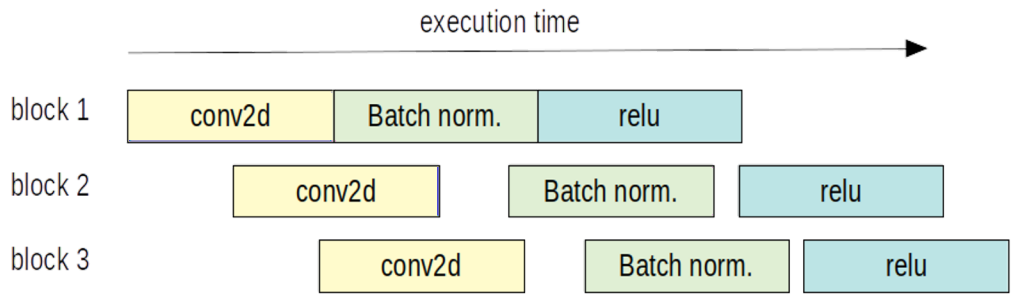
- Use multiple cuda-streams
- Execute multiple kernels that do not have data dependency in parallel

example model:



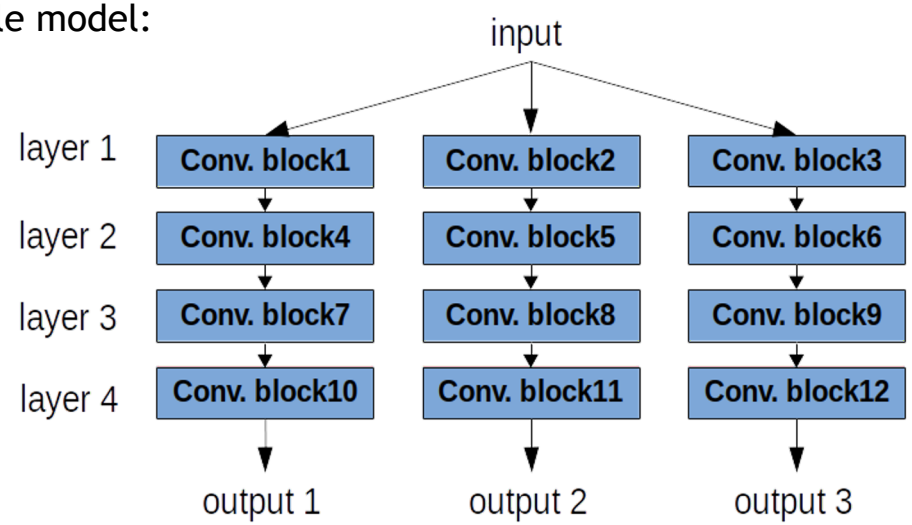
Model Parallel: Multi-Streams

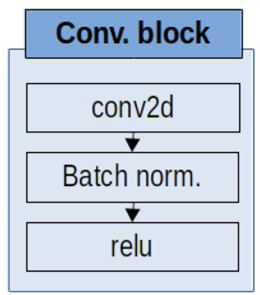
- Use multiple cuda-streams
- Execute multiple kernels that do not have data dependency in parallel



Operations of layers at different branches are overlapped.

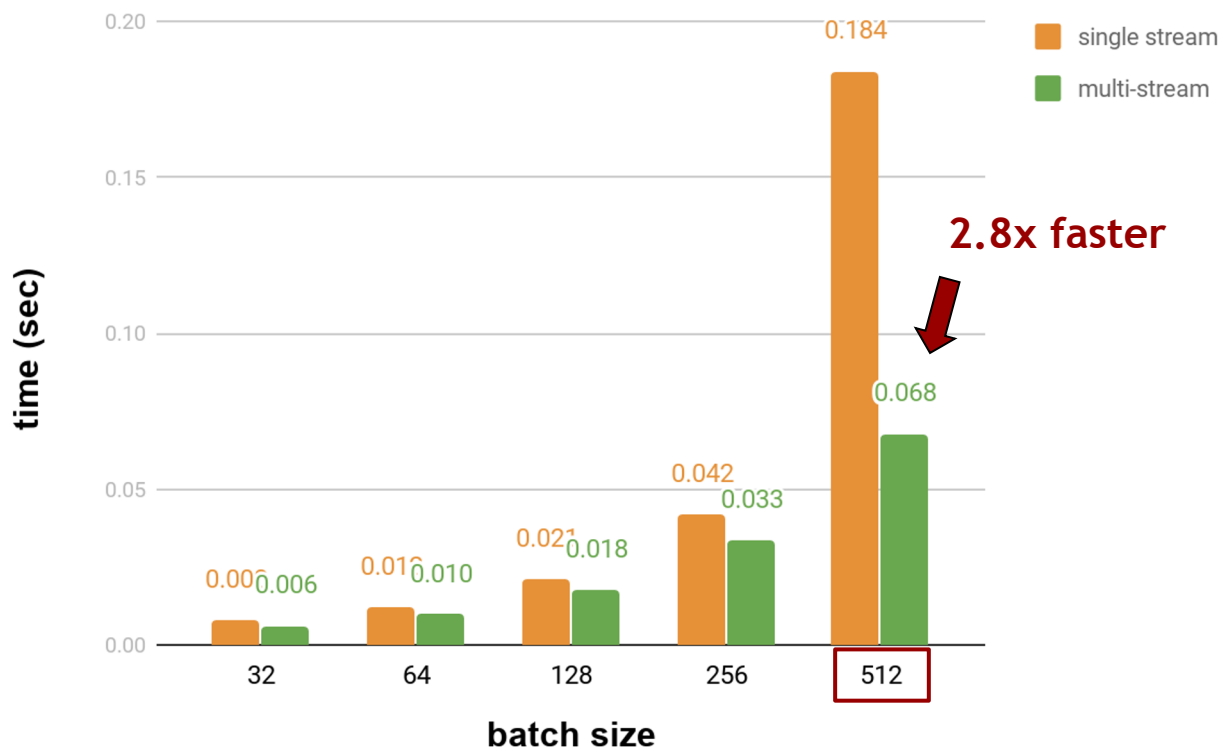
example model:





Model Parallel: Multi-Streams

Computation time comparison




NVIDIA