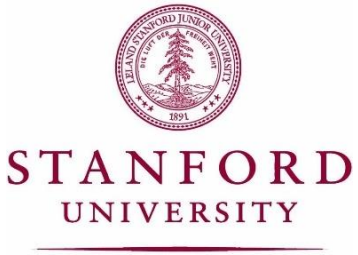
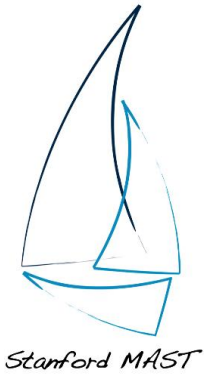




Scaling Neural Network Acceleration using Coarse-Grained Parallelism

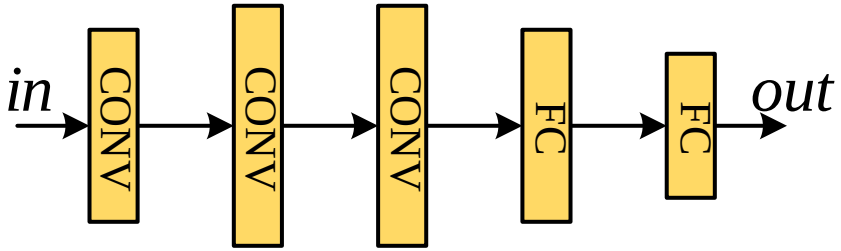
Mingyu Gao, Xuan Yang, Jing Pu,
Mark Horowitz, Christos Kozyrakis

Stanford University

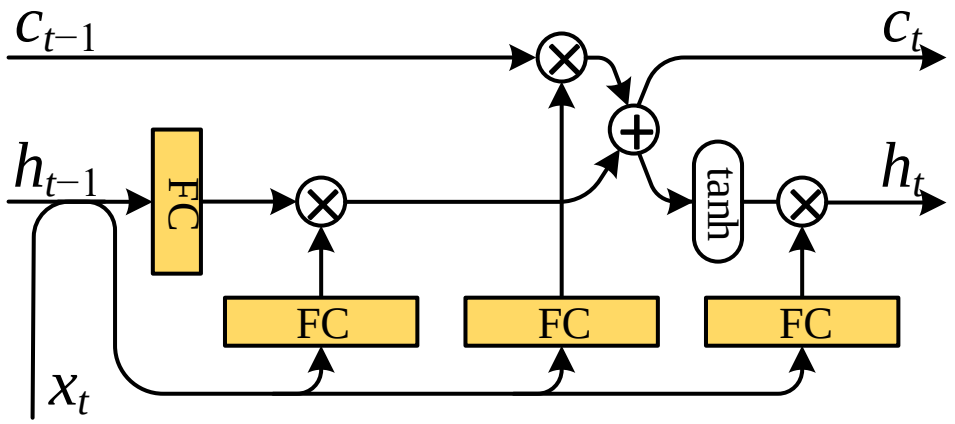
Neural Networks (NNs)

- Unprecedented accuracy for challenging applications
  - Fully-connected, Convolutional, Recurrent (LSTMs) NNs

- Layer-wise processing: direct acyclic graph (DAG)



Convolutional NN (CNN)

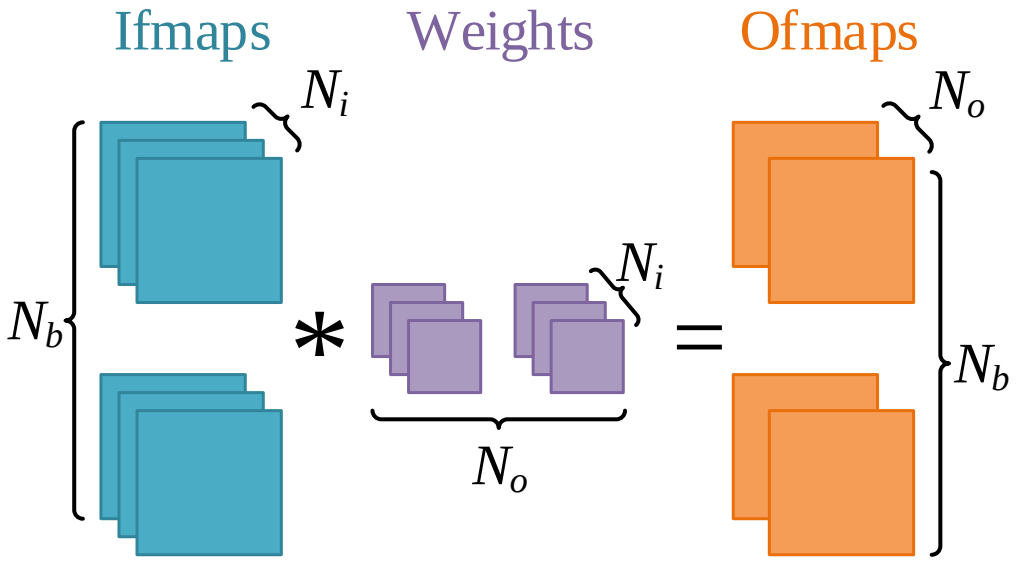


LSTM Cell

Layer Computation

- Convolutional (CONV) and fully-connected (FC) layers
  - FC is a CONV with 1×1 images

- Layer processes a 4D tensor
  - 2D image (feature maps or fmaps), multiple channels, mini-batch

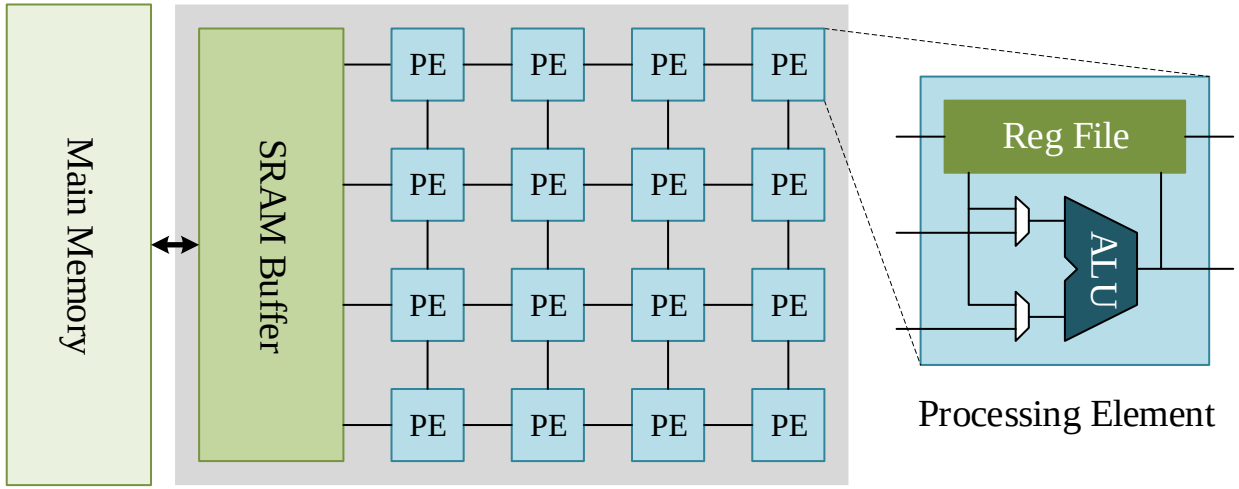



```
foreach b in batch Nb
  foreach ifmap i in Ni
    foreach ofmap o in No
      // 2D conv
      O(b,o) += I(b,i) * W(o,i)
```

Accelerating NNs

- Domain-specific NN processing engine
  - An array of specialized processing elements (PEs)
  - On-chip SRAM buffer

- 100x performance and energy efficiency
  - Diannao/Cambricon, Google TPU, Eyeriss, Cnvlutin, EIE, …



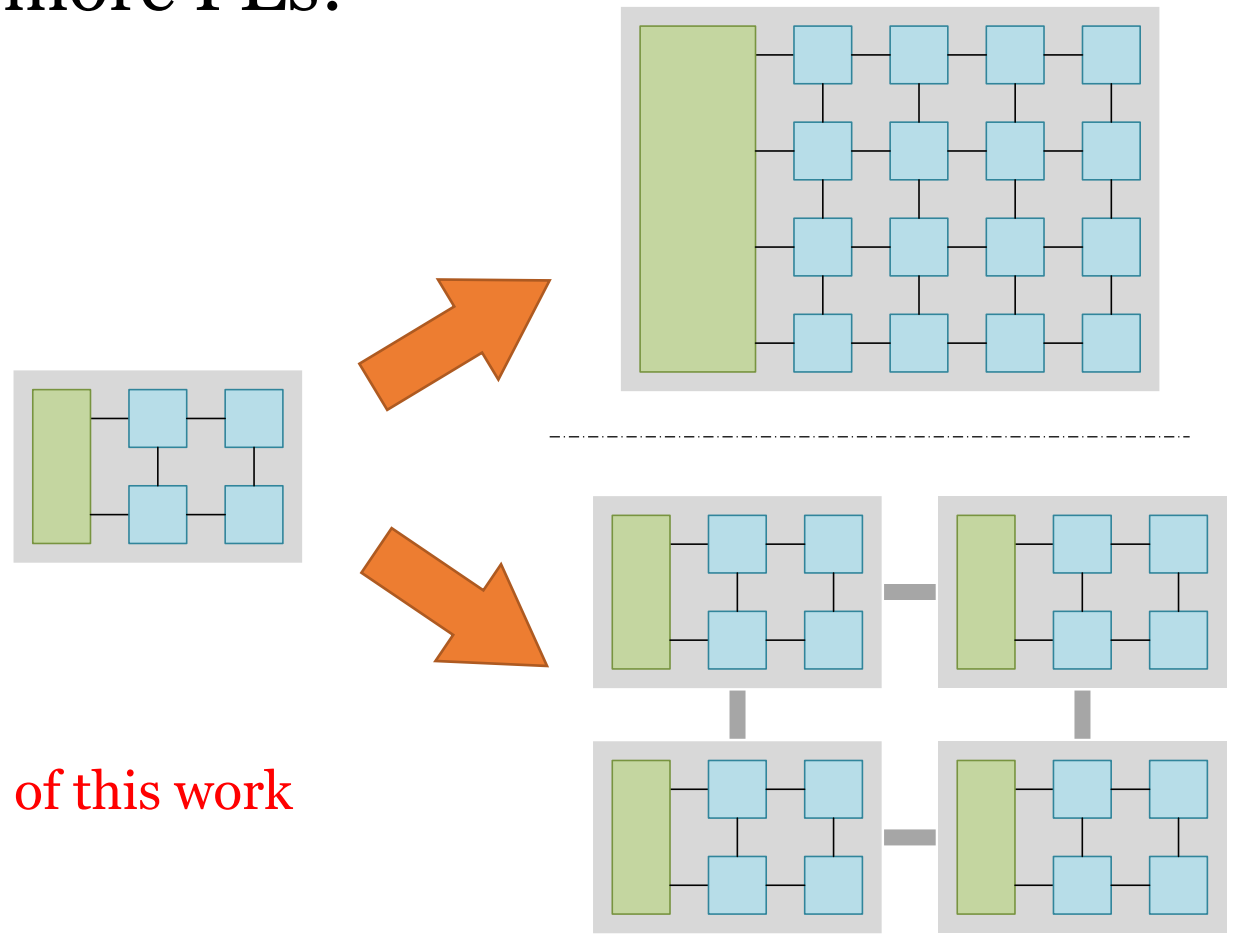
Scaling Performance

- Want higher performance? Use more PEs!

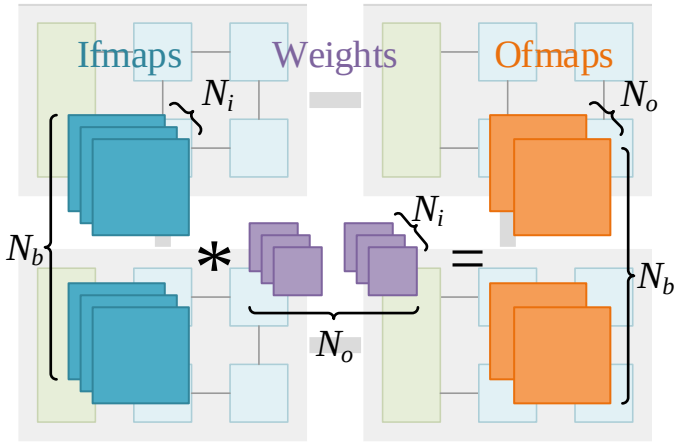
- Monolithic engine
  - ✗ Low resource utilization
  - ✗ Long array buses
  - ✗ Far from SRAM

- Tiled architecture
  - ? Workload parallelization
  - ? Data communication

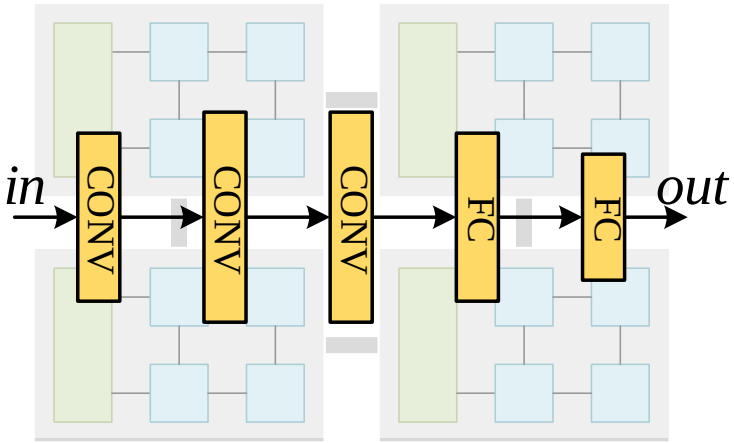
} Focus of this work



Exploiting Coarse-Grained Parallelism



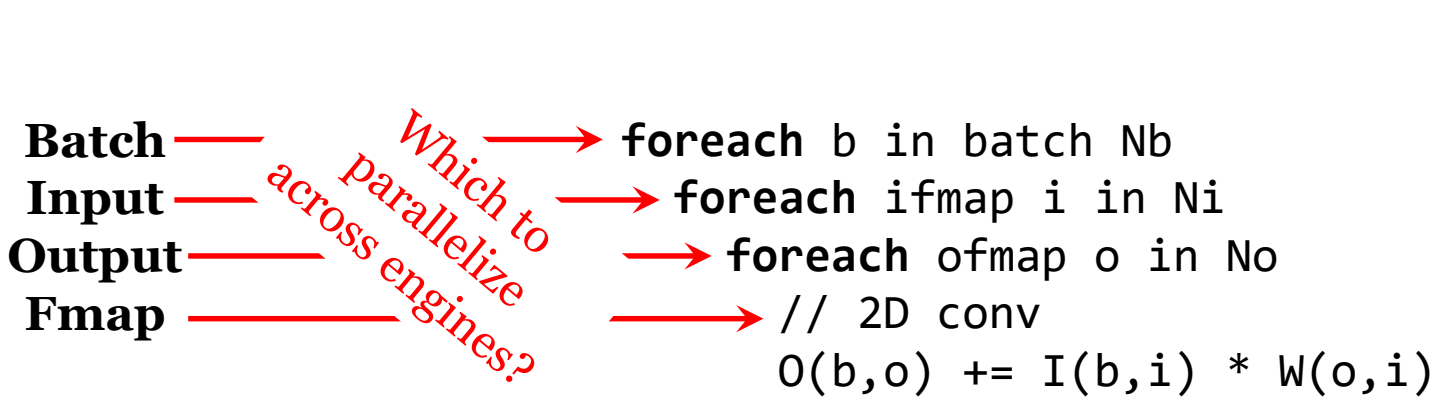
- Intra-layer parallelism
  - Partition/share data
  - Data reuse across engines
    → energy efficiency
  - Avoid on-chip data duplication
    → smaller buffer area

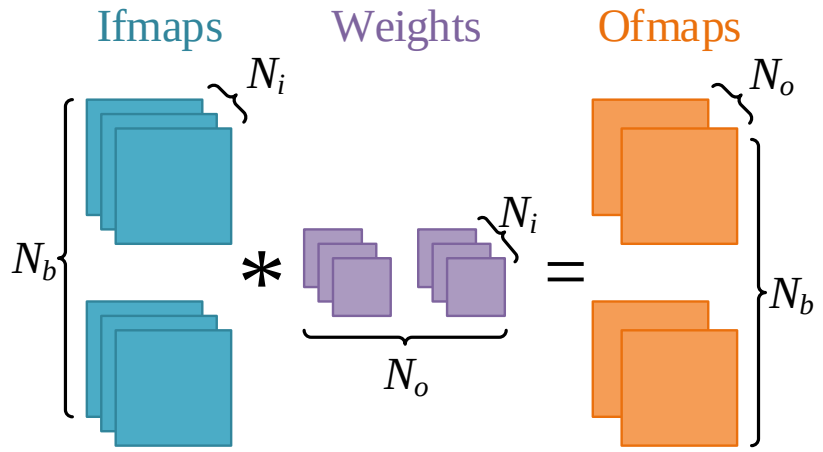


- Inter-layer pipelining
  - Forward data
  - Reduce pipeline stalls
    → performance
  - Manage intermediate data buffering
    → smaller buffer area

Intra-Layer Parallelism

Parallelizing a Single Layer





Scheme	Ifmaps	Ofmaps	Weights
Batch	Partitioned	Partitioned	*Shared*
Input	Partitioned	*Shared*	Partitioned
Output	*Shared*	Partitioned	Partitioned
Fmap	With overlaps	Partitioned	*Shared*

Goals:
1. Reduce sharing
2. Optimize sharing

Reducing Sharing: Hybrid Parallelization

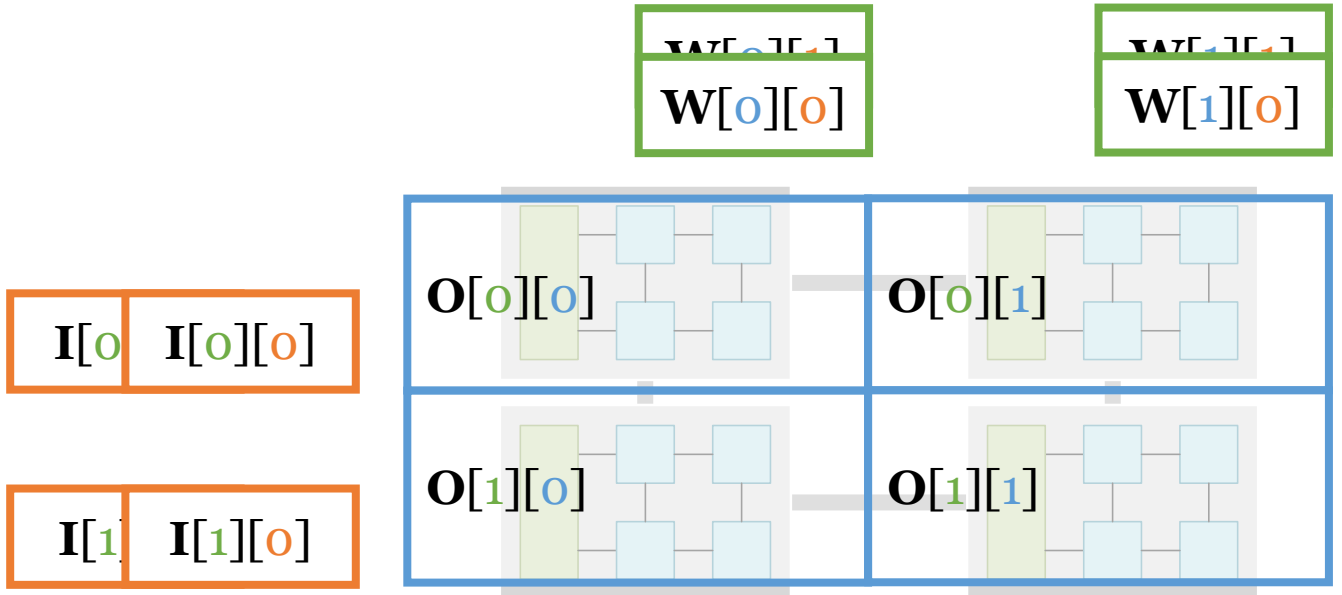
- First CONV layers (fmaps >> weights) → batch/fmap
- Last FC layers (fmaps << weights) → input/output
- Middle layers (fmaps ~~ weights)
  - An optimization problem to minimize accesses to DRAM & NoC
  - Use greedy search algorithm to find a hybrid parallelization

Scheme	Ifmaps	Ofmaps	Weights
Batch	Partitioned	Partitioned	*Shared*
Input	Partitioned	*Shared*	Partitioned
Output	*Shared*	Partitioned	Partitioned
Fmap	With overlaps	Partitioned	*Shared*

Optimizing Sharing: Buffer Sharing

- Skew computation order of engines
  - All engines start in parallel → high throughput
  - No on-chip data duplication → low area
- Rotate buffered data between engines
  - Fully reuse shared data → low energy

Optimal distributed buffer
&
Buffer Sharing dataflow



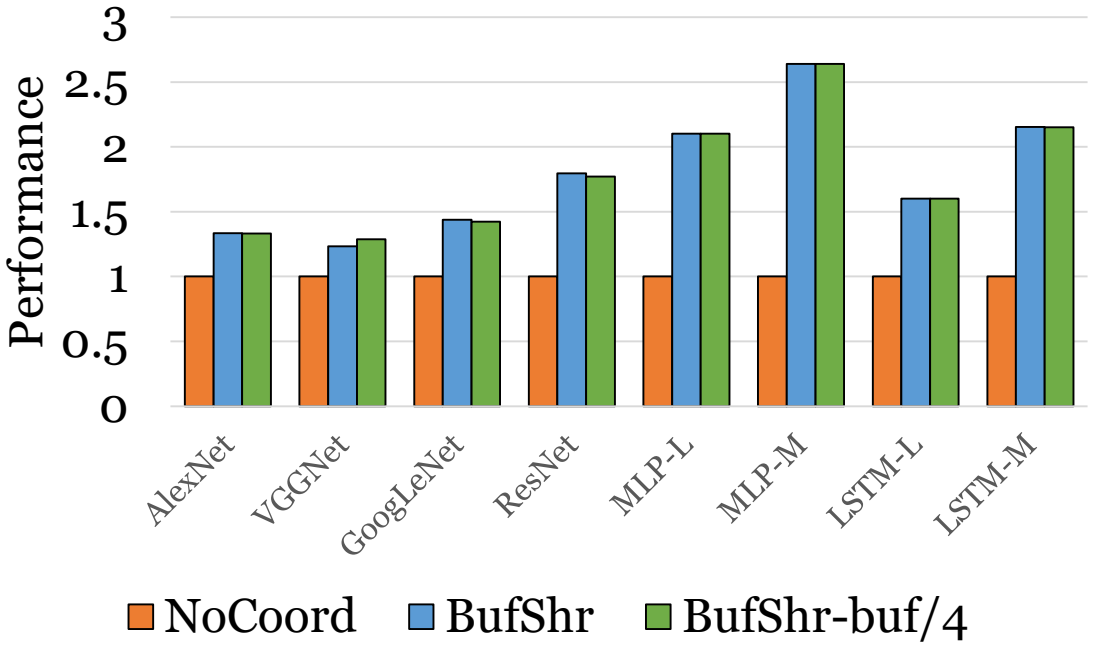
Methodology

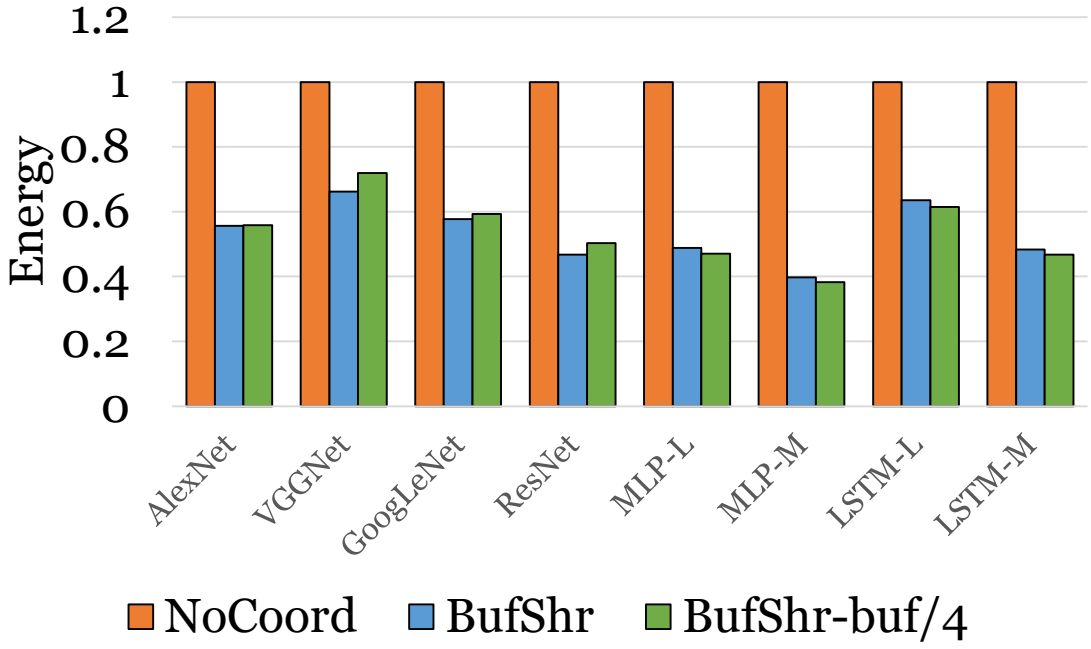
- State-of-the-art NNs
  - CNNs: AlexNet, VGGNet, GoogLeNet, ResNet
  - MLPs & LSTMs: medium and large scales

- Hardware
  - Engine: Eyeriss [ISCA'16], 8 × 8 PEs, 32 kB buffer, 500 MHz
  - Off-chip memory: LPDDR3-1600, 4 channels

Buffer Sharing Results

- NoCoord: same engine resources, but no coordination between
- BufShr: 1.7x performance, 47% less energy
- *AND* 20% less area (if using 4x smaller SRAM)

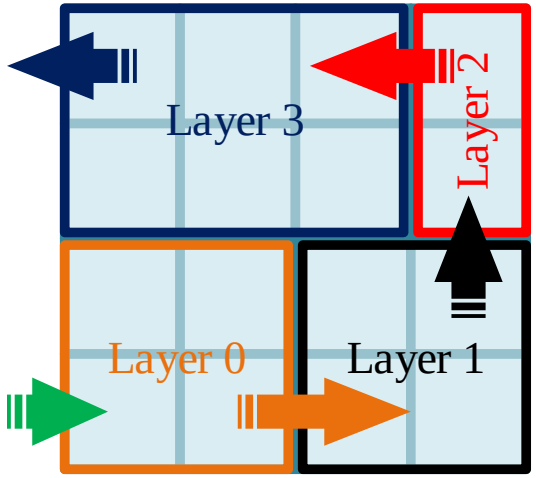




With 64 engines (70 mm², 8 × 8 tiled)

Inter-Layer Pipelining

Pipelining Multiple Layers



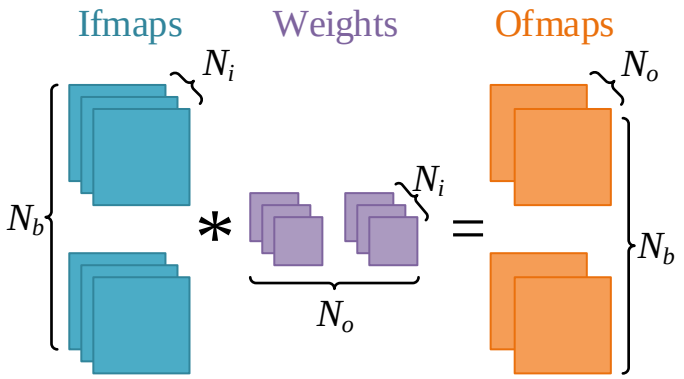
- Avoid off-chip access of intermediate data
  - Save DRAM bandwidth and energy
- Challenges
  - Pipeline filling/draining delays: inter-layer data dependencies
  - SRAM buffer capacity: fully store intermediate data

Optimizing Pipelining Dataflow

- Pipeline filling/draining delays
  - Batch pipelining
  - Alternate layer loop ordering (ALLO)
- Buffer usage for intermediate data
  - Intra-layer buffer sharing
  - Alternate layer loop ordering (ALLO)
  - Fmap temporal partitioning (FMTP)

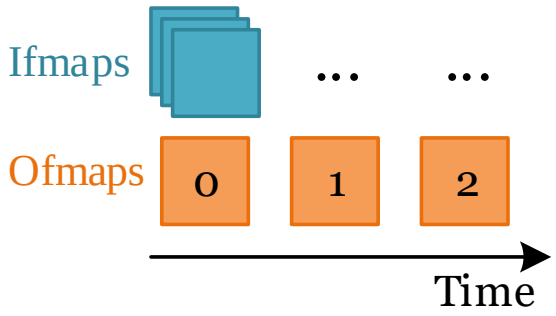
Optimized pipelining dataflow scheme

Ordering Layer Loops



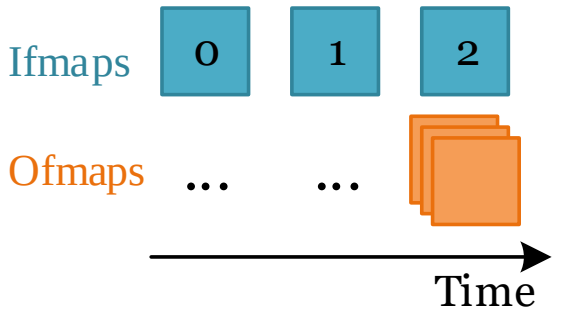
(A Single Layer)

```
foreach ofmap o in No
  foreach ifmap i in Ni
    // 2D conv
    O(o) += I(i) * W(o,i)
```



Take all ifmaps
Generate ofmaps sequentially

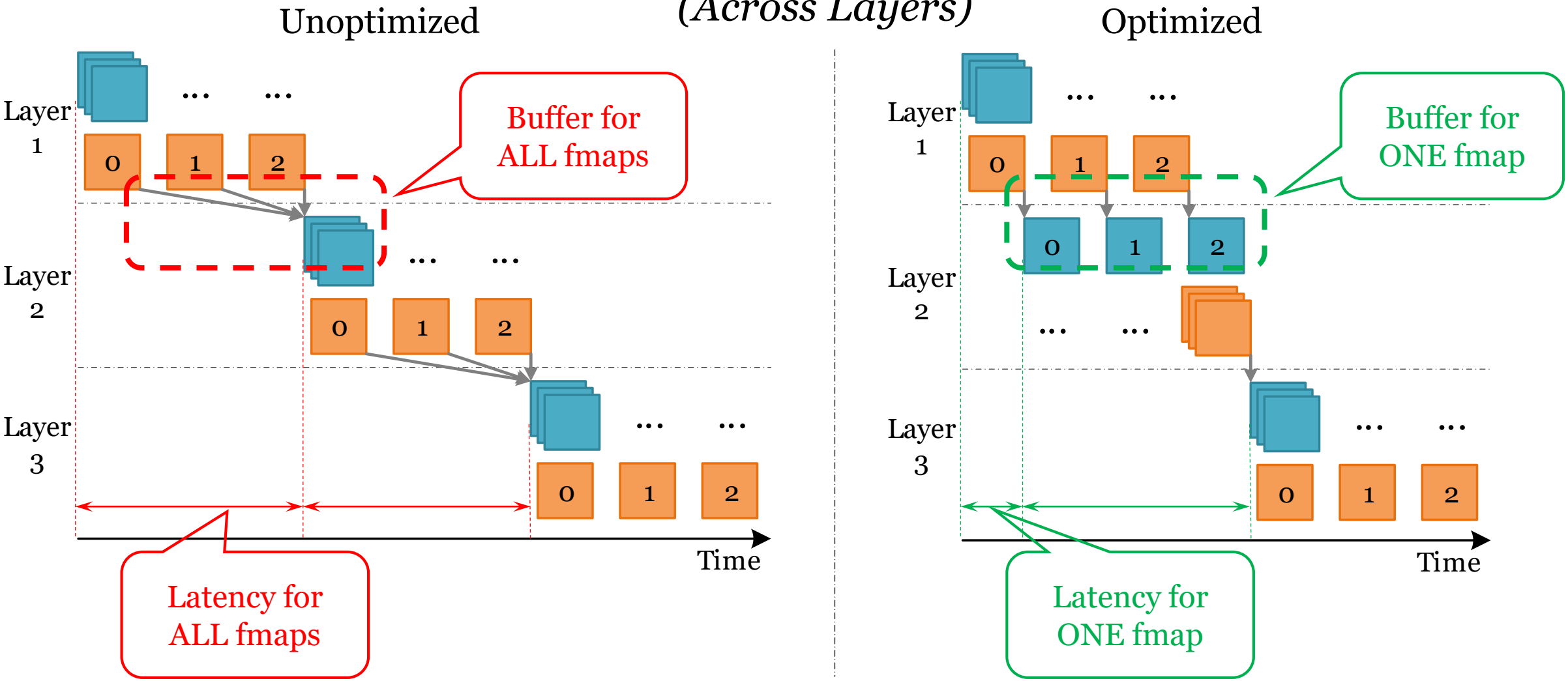
```
foreach ifmap i in Ni
  foreach ofmap o in No
    // 2D conv
    O(o) += I(i) * W(o,i)
```



Take ifmaps sequentially
Generate all ofmaps

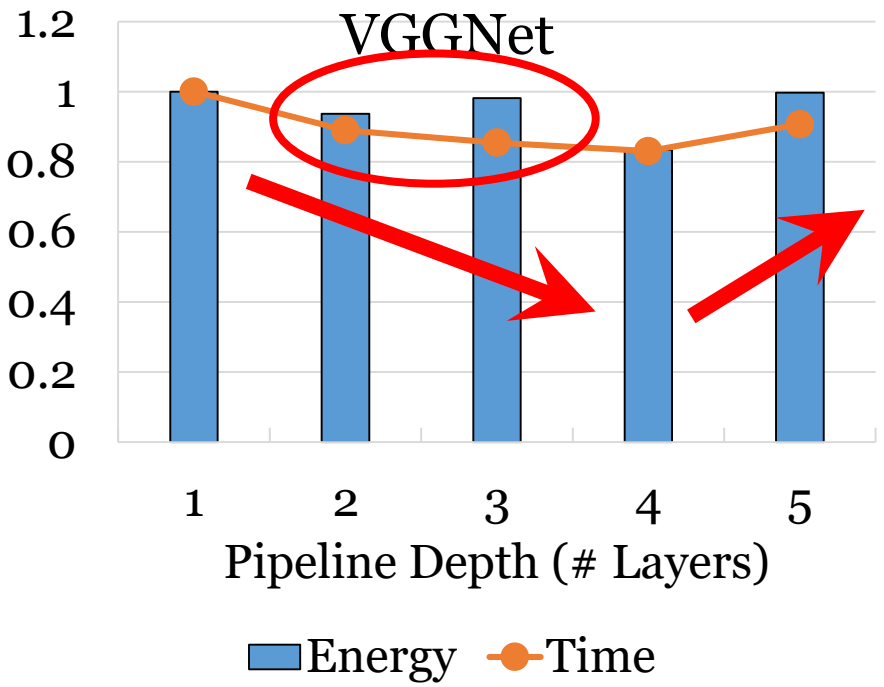
Alternate Layer Loop Ordering (ALLO)

(Across Layers)



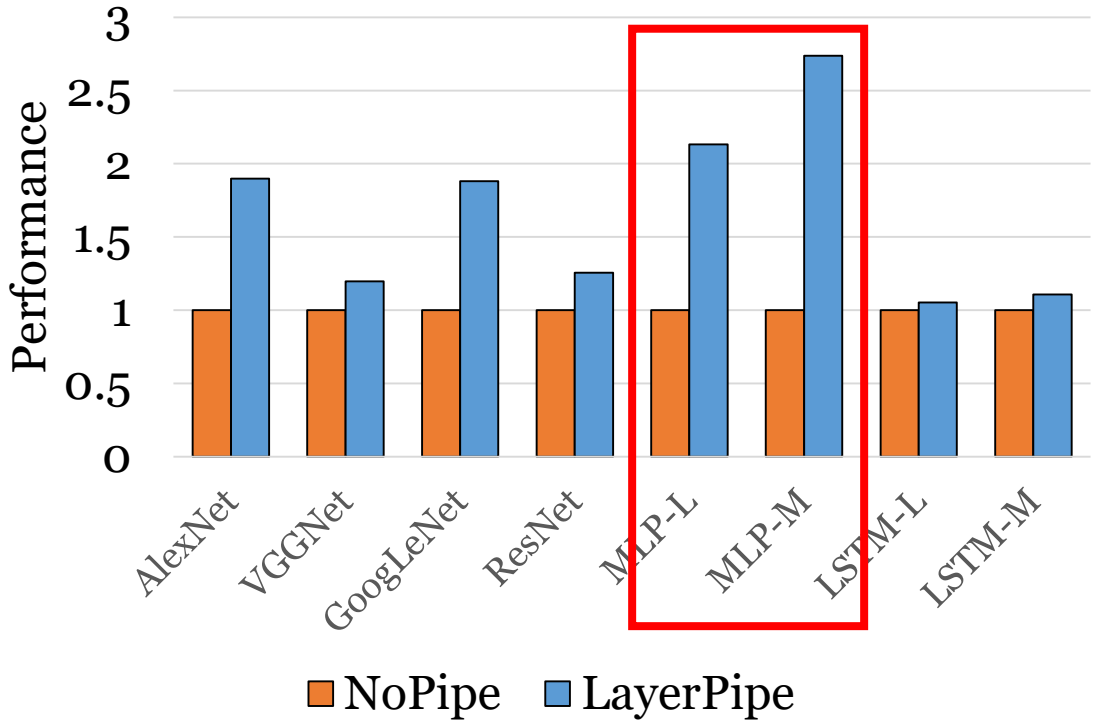
Selecting Pipeline Depth

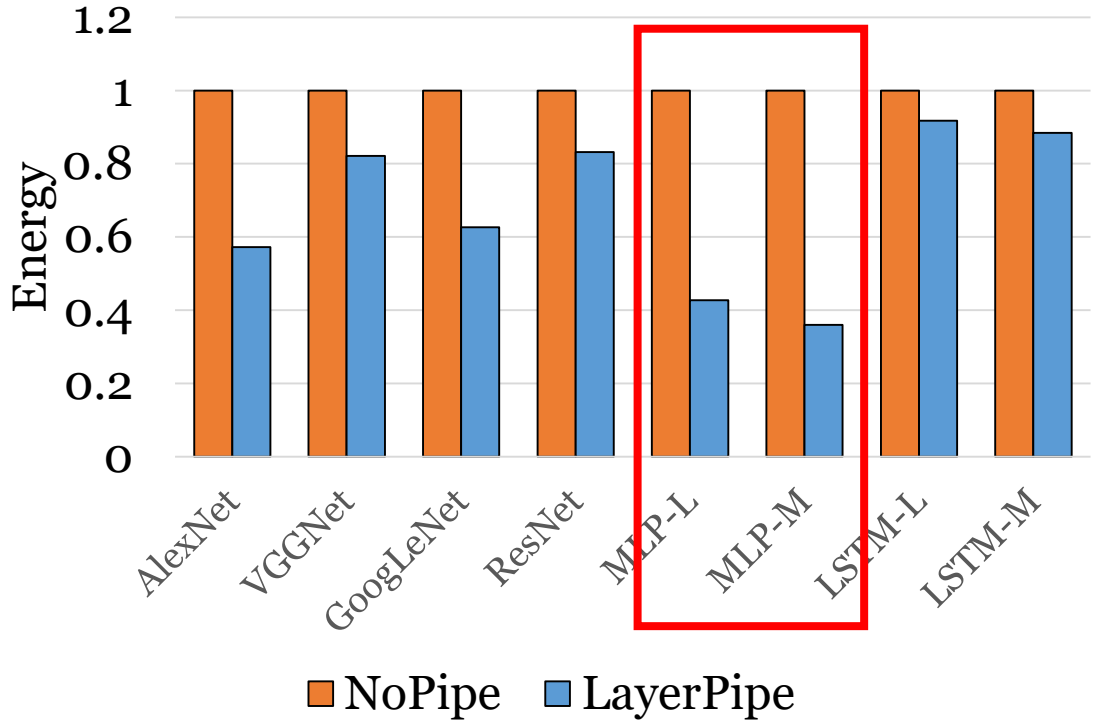
- Deeper pipeline
  - Fewer inter-layer DRAM accesses
- Shallower pipeline
  - More on-chip SRAM per each layer
- Auto-tuner
  - Optimize for total time and energy
  - Use beam search algorithm
  - Work for MLPs, CNNs, and RNNs



Layer Pipelining Results

- NoPipe: no pipelining, only intra-layer parallelism
- LayerPipe: 1.5x performance, 30% less energy





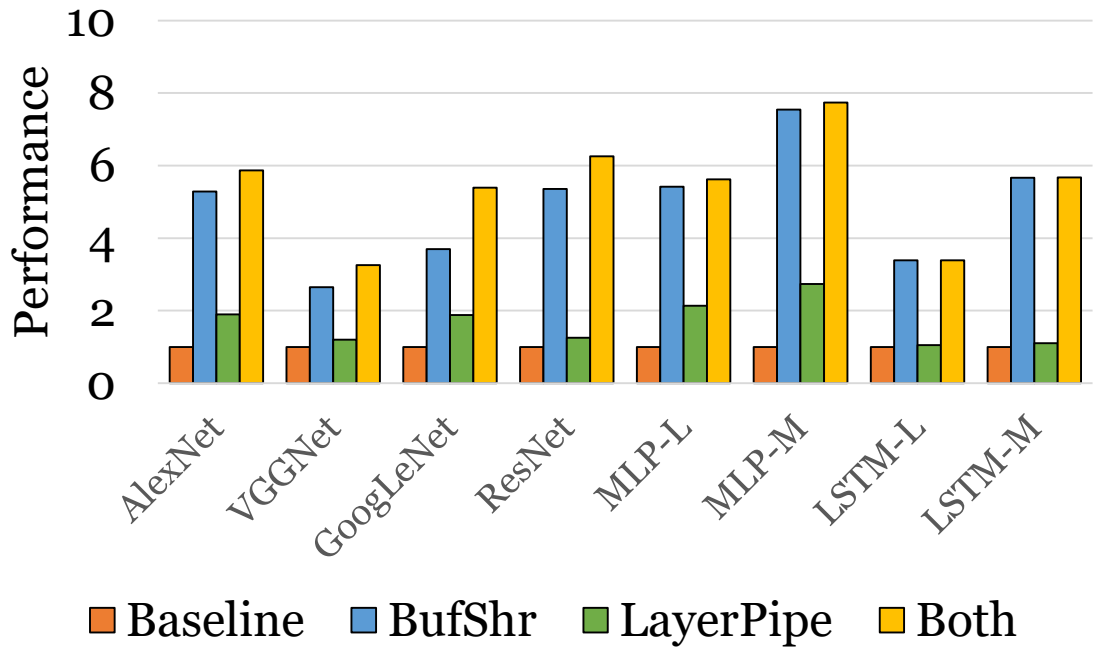
With 256 engines (200 mm², 16 × 16 tiled)

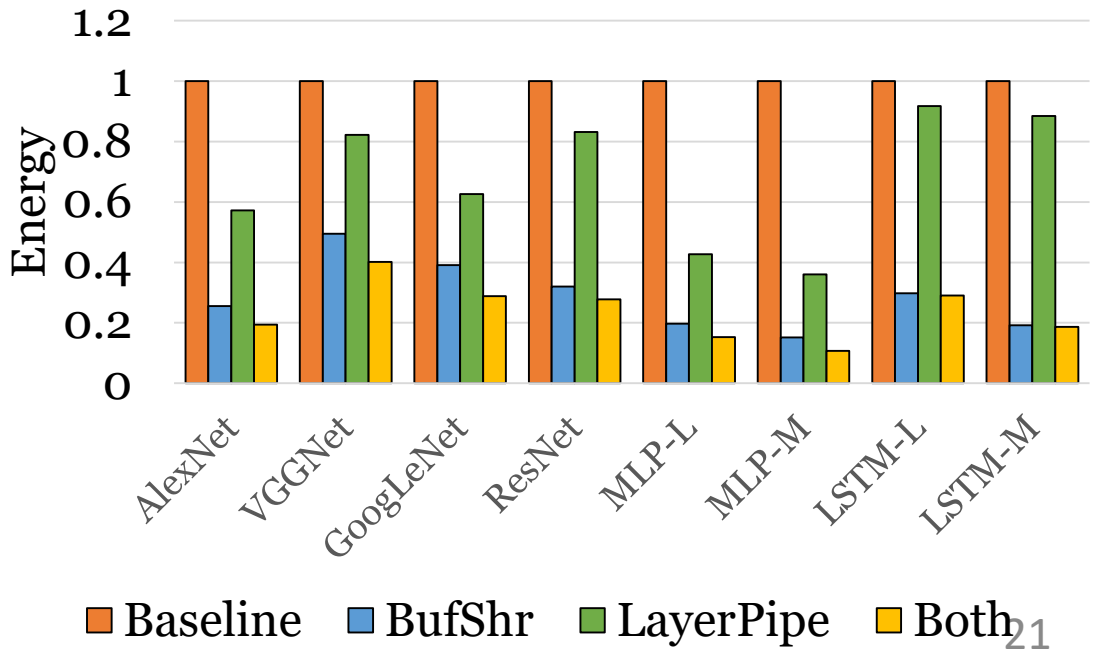
Overall Evaluation

Buffer Sharing + Layer Pipelining

- With 256 engines (200 mm², 16 × 16 tiled)

	BufShr	LayerPipe	Both
Performance Improve	4.6x	1.6x	5.2x
Energy Saving	72%	35%	78%

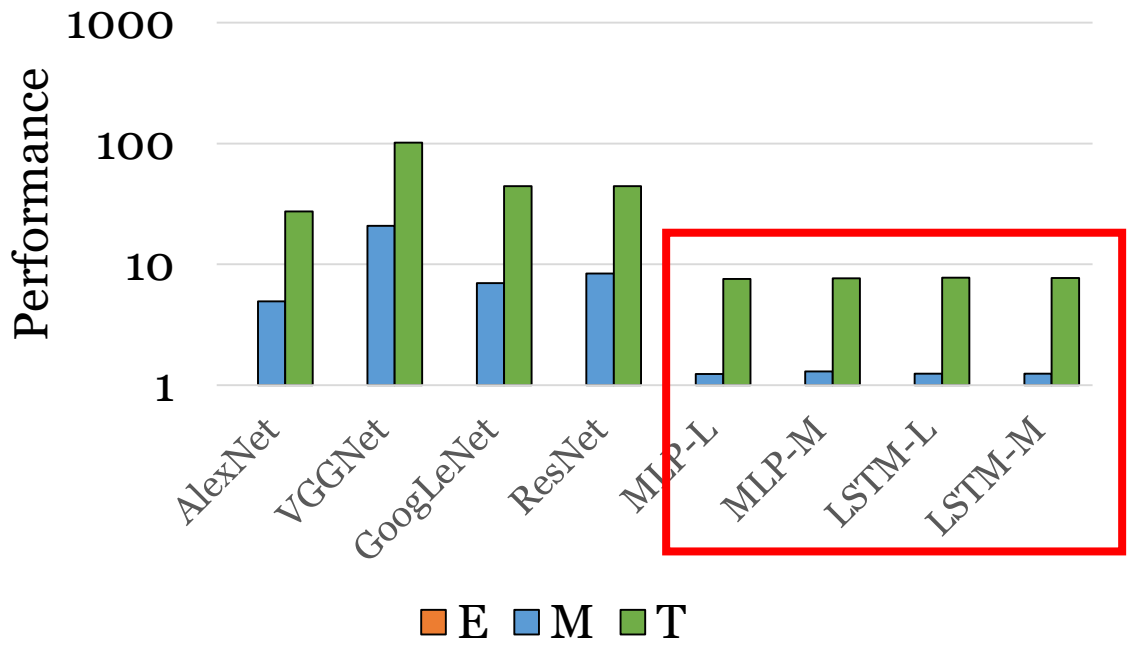


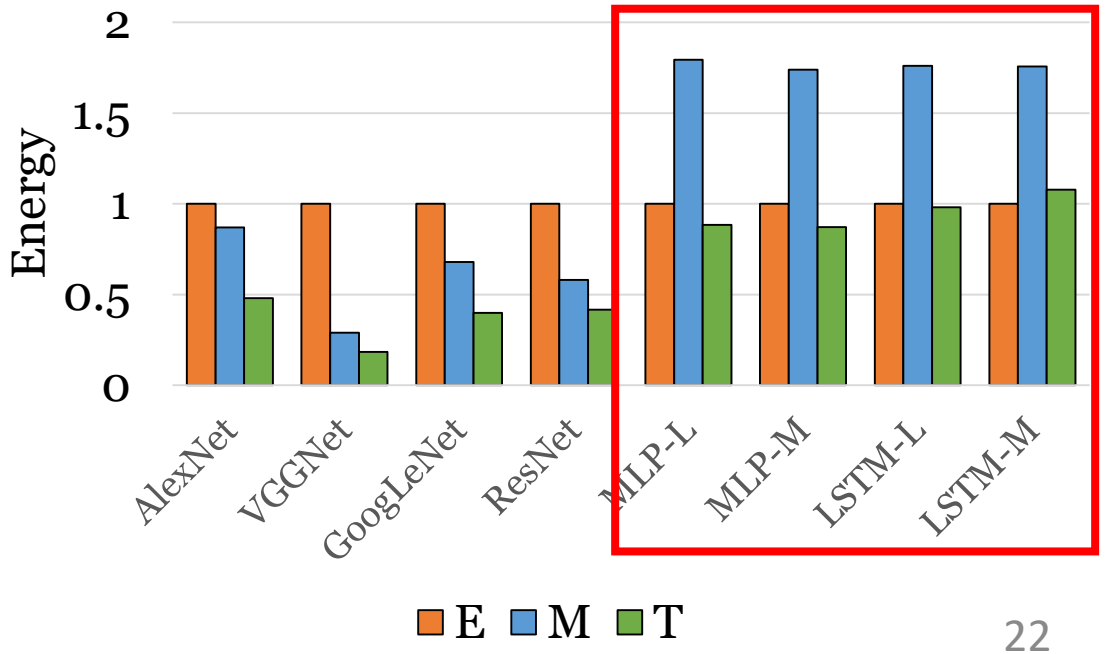


Monolithic vs. Tiled

- With same resources (16384 PEs and 8 MB SRAM)

	Monolithic	Tiled
Performance Improve	3.3x	19.3x
Energy Saving	<1%	43%





Summary

- Effectively and efficiently scale NN acceleration
  - Coarse-grained parallelism on tiled architecture
  - 6x better than using a monolithic engine

- Dataflow optimizations
  - Intra-layer buffer sharing
  - Inter-layer pipelining

Thanks!

Questions?