Unifying Data, Model and Hybrid Parallelism in Deep Learning via Tensor Tiling

Minjie Wang, Chien-chin Huang, Jinyang Li

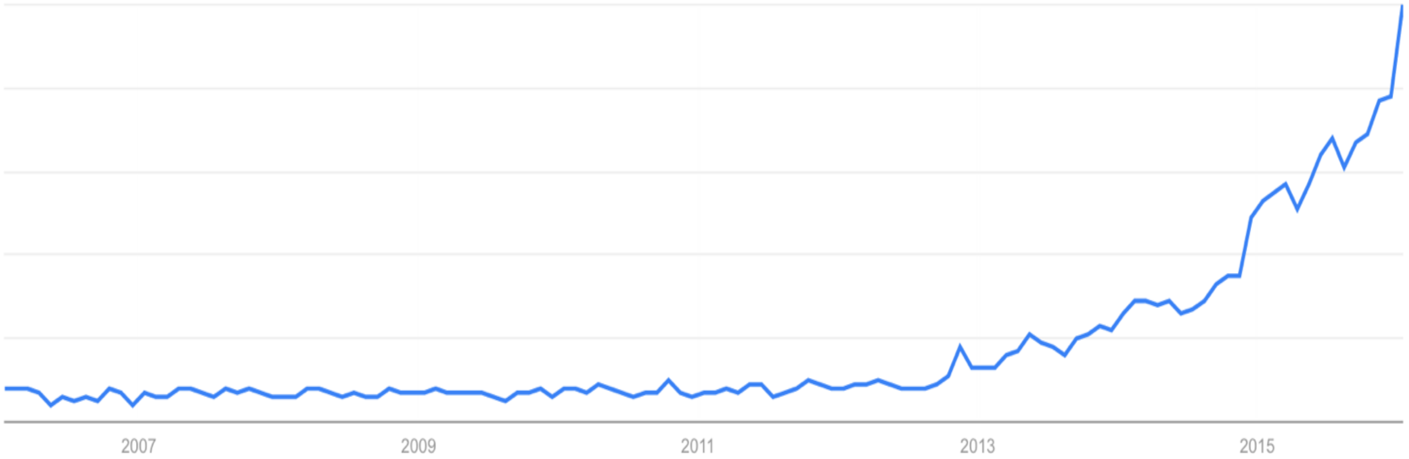
[ARXIV18]

Adapted From Minjie Wang

Outline

- Background & Motivation
- Problem Setup & Challenge
- Tofu solution
  - Single Operator
  - Whole Graph
- Experiments

Deep Learning

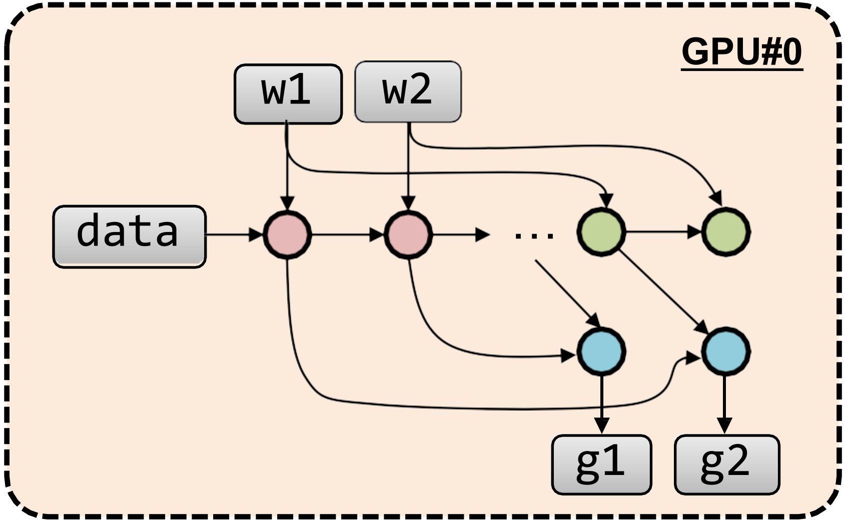


"Deep Learning" trend in the past 10 years

Caffe theano TensorFlow mxnet

State-of-art DL system is based on dataflow

```
import tensorflow as tf
...  # generate data and weight
act1 = tf.matmult(data, w1)
act2 = tf.matmult(act1, w2)
...
grad_act2 = tf.matmult(w3.T, grad_act3)
grad_act1 = tf.matmult(w2.T, grad_act2)
...
grad_w2 = tf.matmult(act1.T, grad_act2)
grad_w1 = tf.matmult(data.T, grad_act1)
...  # update weights using gradients
```



Forward propagation

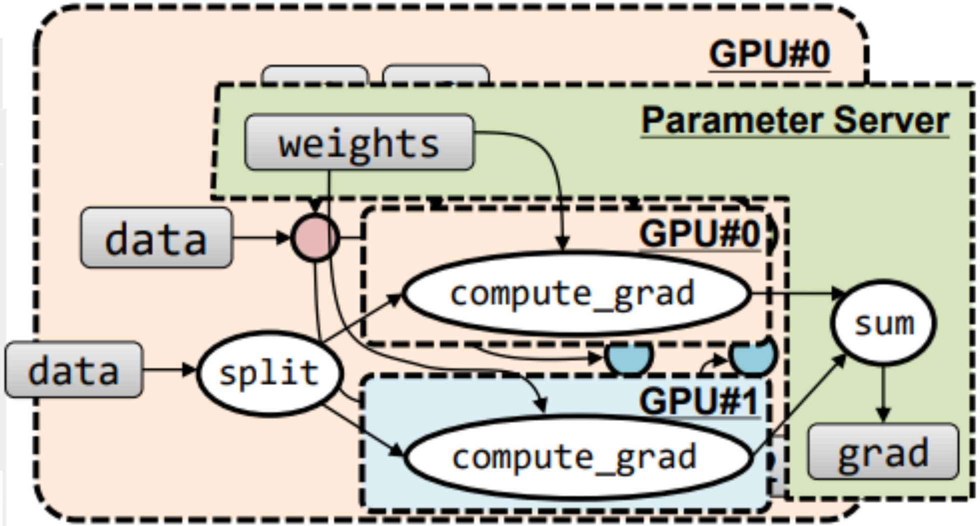
Backward propagation (input gradients)

Backward propagation (weight gradients)

What if I have many GPUs?

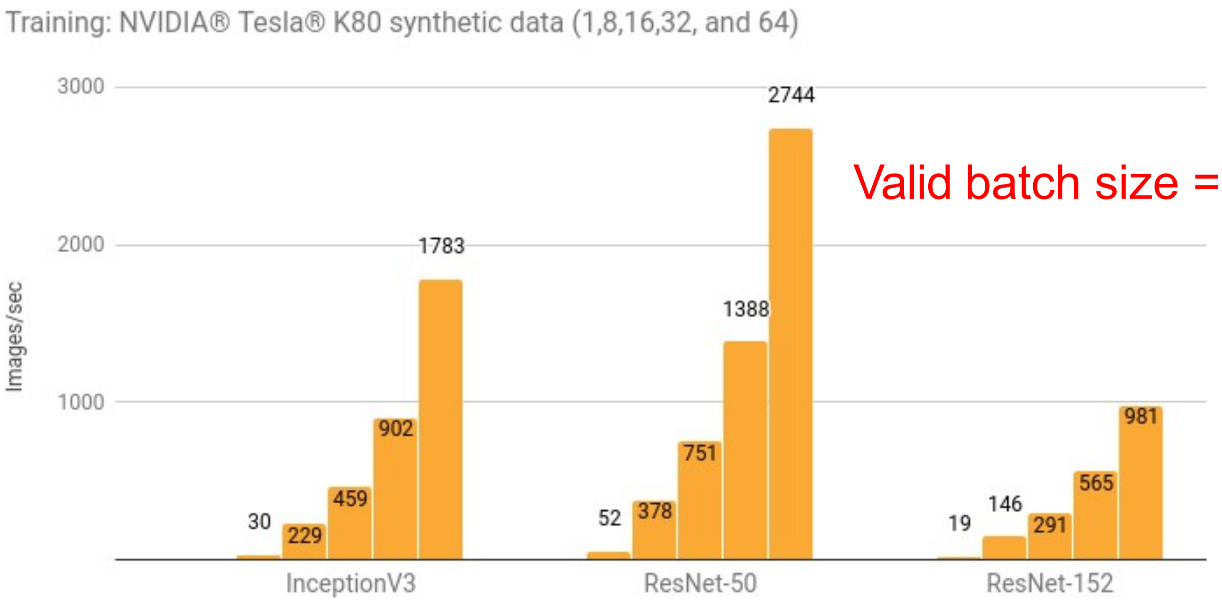
Data parallelism with manual distribution

```python
import tensorflow as tf
...  # generate data and weight
data1, data2 = tf.split(data, axis=0)
with tf.device('/gpu:0'):
    grad1 = compute_grad(data1, weights)
with tf.device('/gpu:1'):
    grad2 = compute_grad(data2, weights)
with tf.device('/ps'):
    grad = aggregate(grad1, grad2)
    ...  # update weights using gradients
```



Manual Distribution & Device assignment

Scalability secret of data parallelism

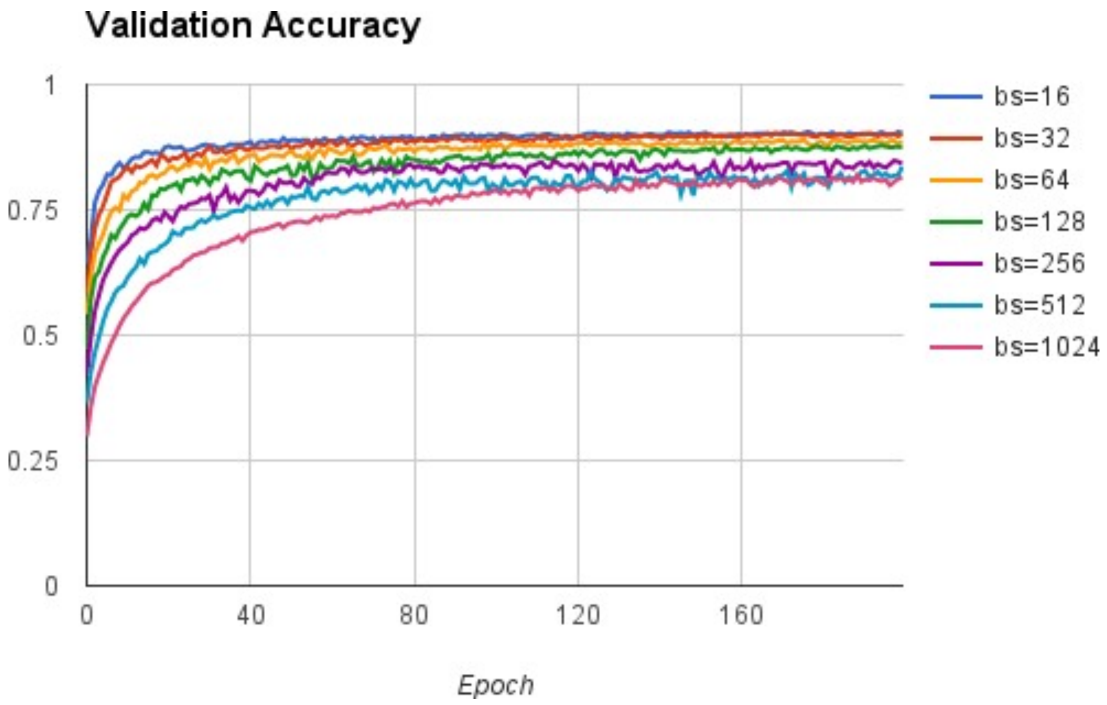


Valid batch size = 64 * 64 = 4096

Options	InceptionV3	ResNet-50	ResNet-152	Alexnet	VGG16
Batch size per GPU	64	64	64	512	64
Optimizer	sgd	sgd	sgd	sgd	sgd

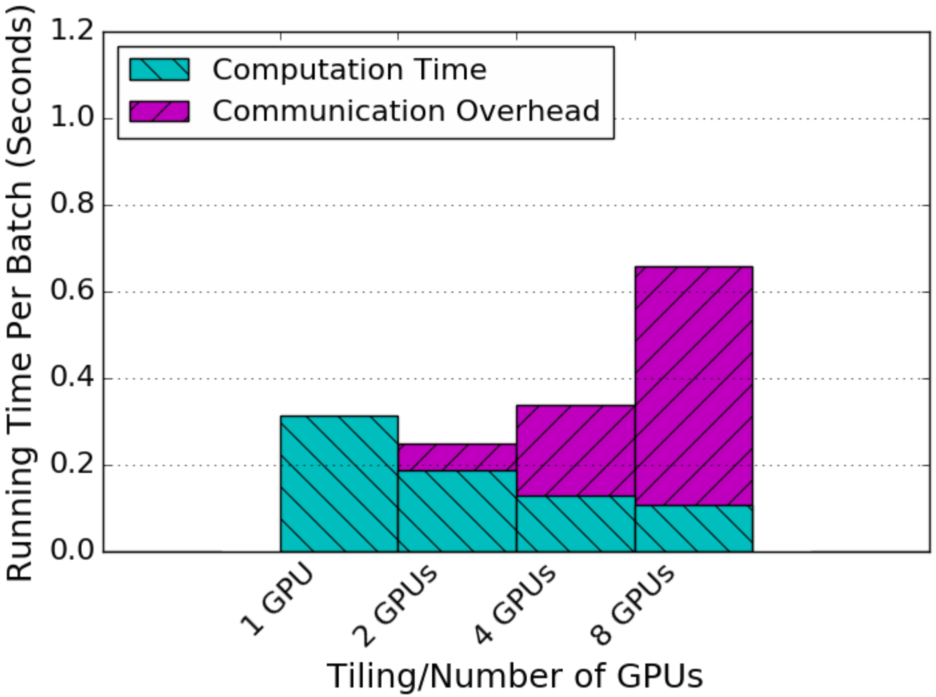
* Numbers from https://www.tensorflow.org/performance/benchmarks

Large batch size harms model accuracy



Inception Network on Cifar-10 dataset

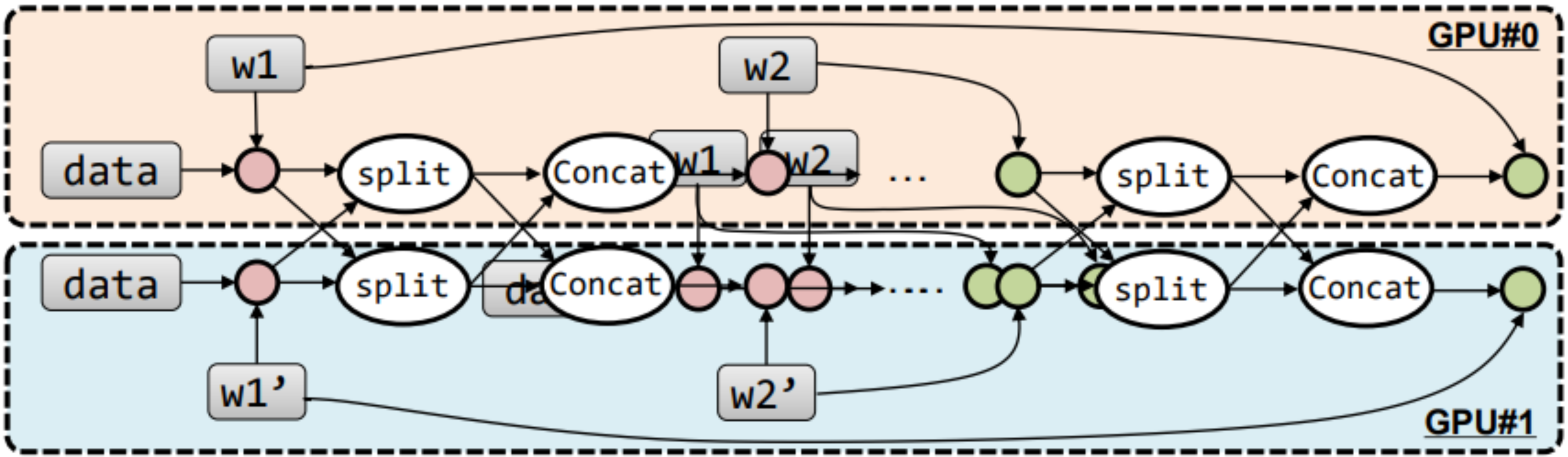
Data parallelism bottlenecked by communication



>80% of the total running time is for communication on 8 cards

5-layer MLP; Hidden Size = 8192; Batch Size = 512

An alternative way: Model Parallelism



Forward propagation

Backward propagation (input gradients)

MP is hard to program

```python
# Original MLP code.
def mlp(data, weights):
  # Forward Propagation.
  fwd = [data]
  for i in xrange(FLAGS.num_layers):
    fwd.append(tf.matmul(fwd[-1], weights[i])) # forward matmult
  # Backward Propagation.
  targets = []
  last = fwd[-1]
  for i in reversed(xrange(FLAGS.num_layers)):
    dw = tf.matmul(fwd[i], last, transpose_a=True) # matmult: grad
    last = tf.matmul(last, w[i], transpose_b=True) # matmult: bp
    # update
    targets.append(dw)
  return targets
```

```python
# Manual Model Parallelism implementation for a MLP network.
def model_par_mlp(data, weights):
  # Partition weights on row.
  w = []
  for i in xrange(FLAGS.num_layers):
    w.append([])
    for j in xrange(FLAGS.num_workers):
      with tf.device('/job:worker/task:%d' % j):
        w[i].append(tf.get_variable(
              name='w%d' % j,
              shape=[slice_size,feature_size],
              trainable=True))
  # Forward Propagation.
  fwd = []
  last = data
  for i in xrange(FLAGS.num_layers):
    with tf.name_scope('fc_ff%d' % i):
      fwd.append(last)
      tmp = []
      for j in xrange(FLAGS.num_workers):
        with tf.device('/job:worker/task:%d' % j):
          y = tf.matmul(last[j], w[i][j])  # forward matmult
          # split the result so we can do balanced reduction.
          tmp.append(tf.split(split_dim=1, num_split=FLAGS.num_workers, value=y))
      # Reduce the result.
      red = []
      for j in xrange(FLAGS.num_workers):
        with tf.device('/job:worker/task:%d' % j):
          red.append(tf.accumulate_n([s[j] for s in tmp]))
      last = red
  # Backward Propagation.
  targets = []
  for i in reversed(xrange(FLAGS.num_layers)):
    with tf.name_scope('fc_bp%d' % i):
      # Concatenate input tensors.
      tmp = []
      for j in xrange(FLAGS.num_workers):
        with tf.device('/job:worker/task:%d' % j):
          tmp.append(tf.concat(concat_dim=1, values=last))
      last = []
      for j in xrange(FLAGS.num_workers):
        with tf.device('/job:worker/task:%d' % j):
          dy = tf.matmul(tmp[j], w[i][j], transpose_b=True) # matmult: bp
          last.append(dy)
          dw = tf.matmul(fwd[i][j], tmp[j], transpose_a=True) # matmult: grad
          targets.append(dw) # update
  return targets
```

Outline

- Background & Motivation
- Problem Setup & Challenge
- Tofu solution
  - Single Operator
  - Whole Graph
- Experiments

What is the best strategy for distribution?

- No one-size-fits-all
  - DP and MP suit different situations (parameter shapes, batch sizes).
  - Different layers might be suited for different strategies **(hybrid parallelism)**.
    - Use data parallelism for convolution layers; use model parallelism for fully-connected layers.

- *Can we find an optimal distributed execution plan?*

Parallelism in Deep Learning

- Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks [ICML18, Stanford & MSR]
- Unifying Data, Model and Hybrid Parallelism in Deep Learning via Tensor Tiling [NYU, May 10]
- PipeDream: Fast and Efficient Pipeline Parallel DNN Training [MSR&Stanford&CMU, June 8]
- Beyond Data and Model Parallelism for Deep Neural Networks [Stanford, July 14]
- GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Google Brain, Nov 20]

Outline

- Background & Motivation
- Problem Setup & Challenge
- Tofu solution
  - Single Operator
  - Whole Graph
- Experiments

Tofu automatically distributes DL training

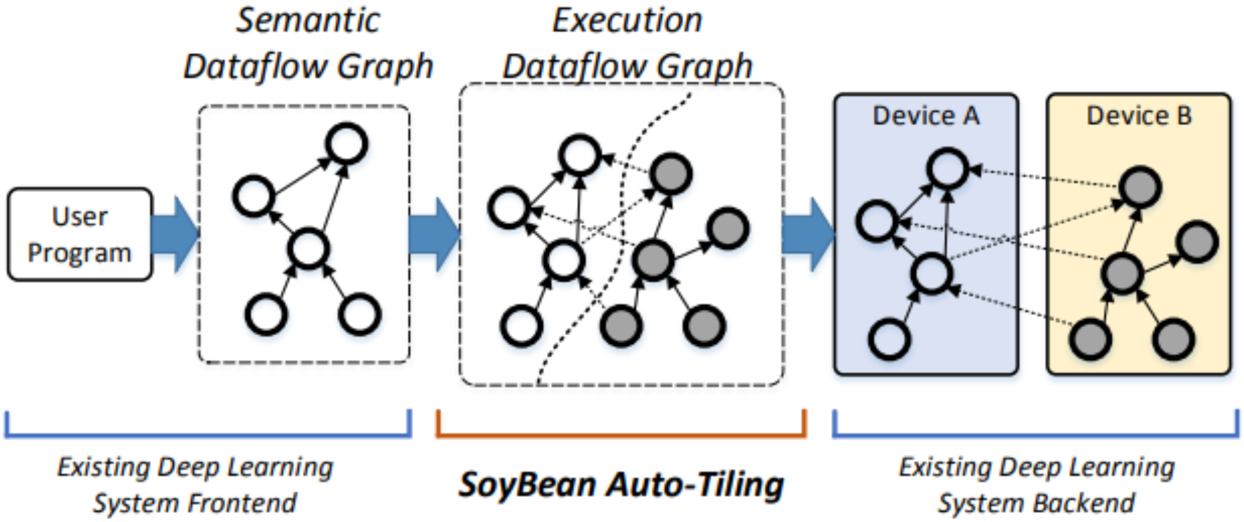


Figure 3: Overview of SoyBean's design.

Tofu

Parallelism unified as tensor tiling:

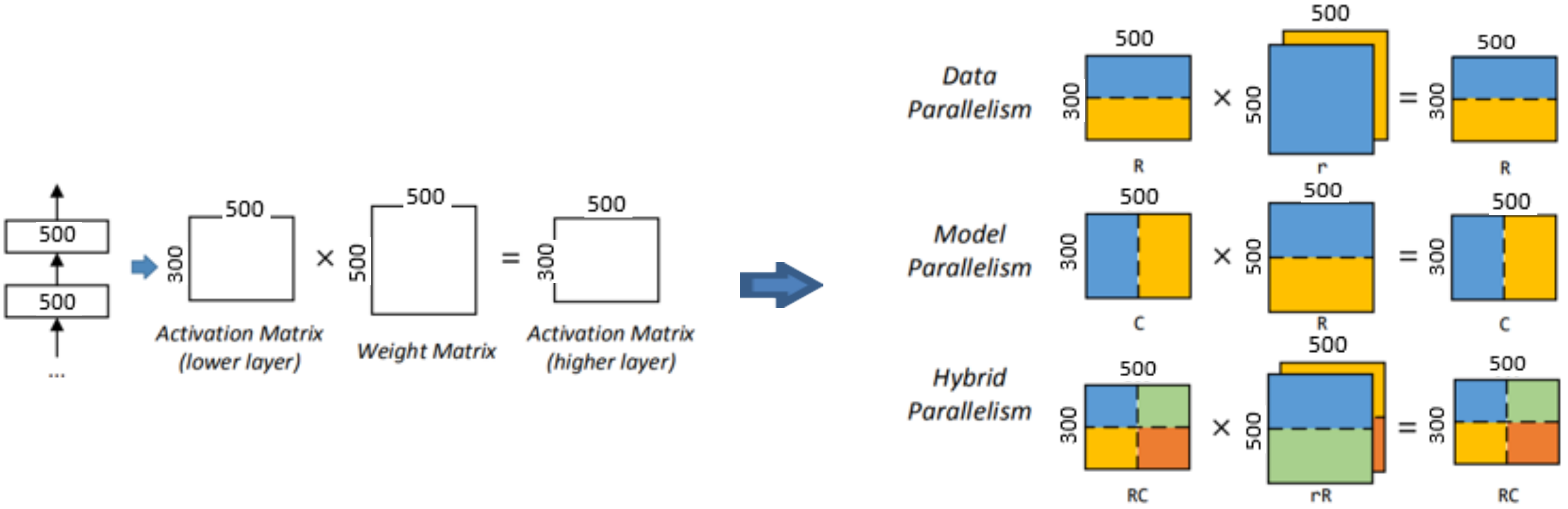


Figure 5: Top: forward propagation of one layer in a MLP model. Bottom: how matrices are tiled in the forward propagation for different parallelisms.

Aligned Tilings

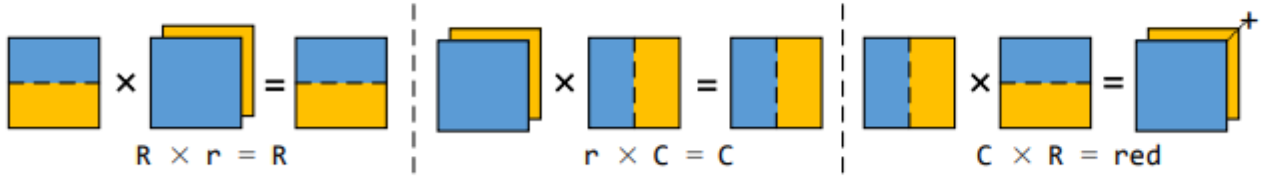
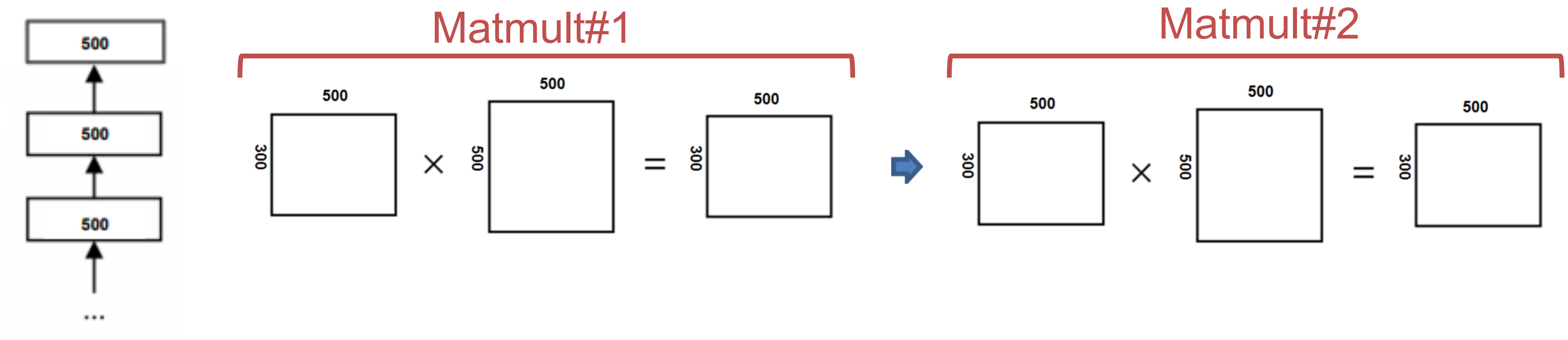


Figure 6: Three forms of aligned tilings for matrix multiplication. The resulting partition of the third form is an intermediate one `red`, and requires an extra reduction.

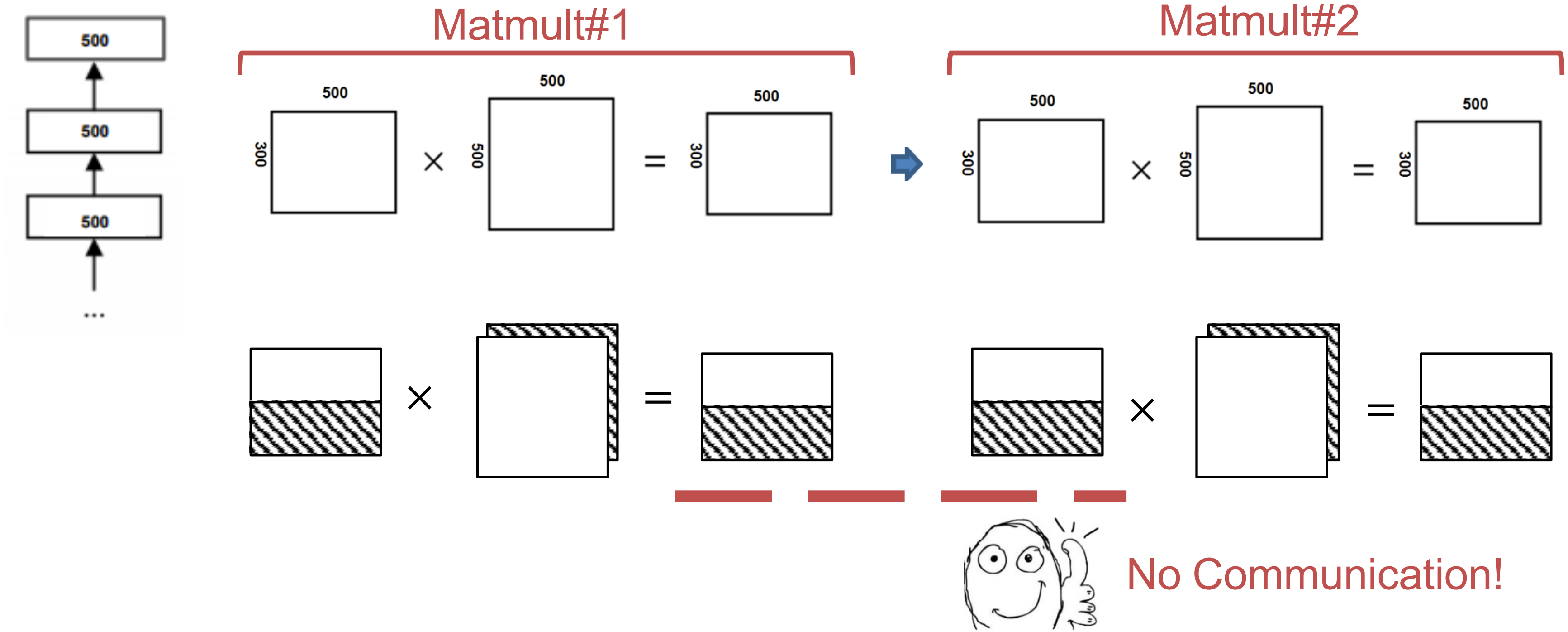
Example of different strategies

- Different matrix multiplications may choose different strategies.



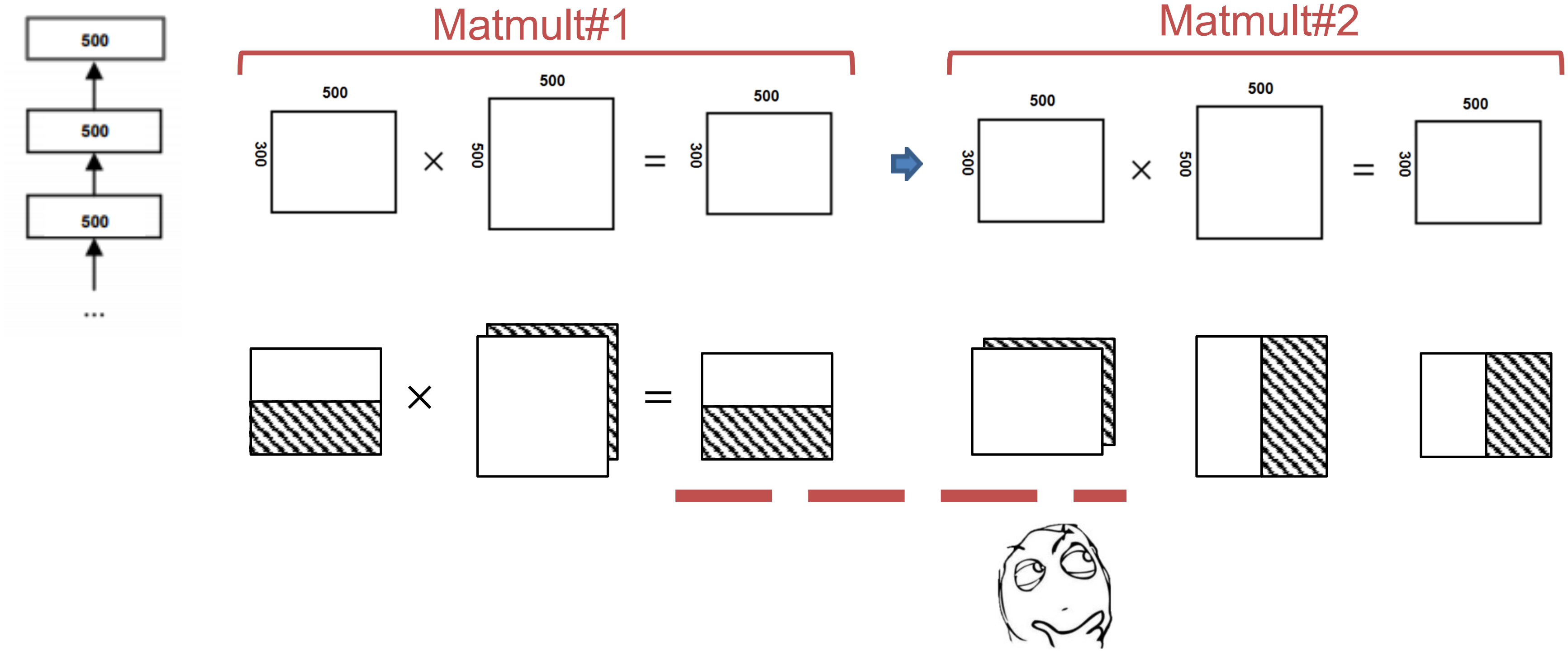
Example of different strategies

- No communication if the output matrix satisfies the input partition.



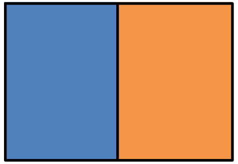
Example of different strategies

- Communication happens when matrices need to be re-partitioned.



Communication Cost

- Communication happens when matrices need to be re-partitioned.
- Communication cost == partition conversion cost.



C

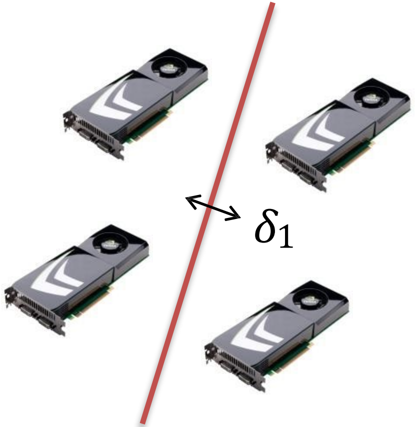
R

Finding optimal strategy with minimal communication

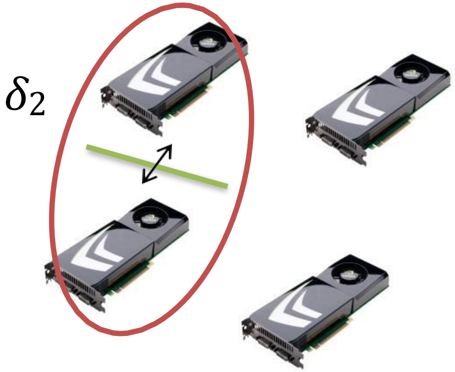
- Each operator has several distribution decisions.
  - DP and MP are one of them.
- Looking at one operator at a time is **not** optimal.
- Finding strategy with minimal communication cost for a general graph is NP-Complete.
- <u>Tofu finds optimal strategy for deep learning in polynomial time</u>:
  - "Layer-by-layer" propagations → graph with long diameter.
  - Use dynamic programming algorithm to find optimal strategy.

Find combined strategies

- Solve the problem recursively.
- Proved to be optimal.

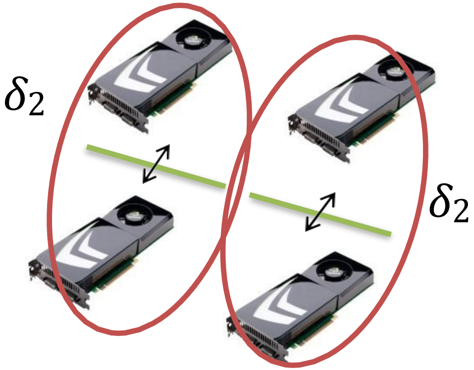


Step 1: Partition to two groups



Step 2: Apply the algorithm again on one of the group

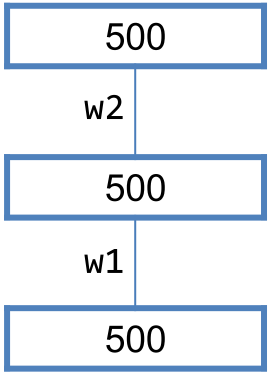
$$\delta_{total} = \delta_1 + 2\delta_2$$



Step 3: Apply the same strategy to the other group due to symmetry.

Which One is Better?

ToyNet Configuration



<u>nGPUs</u>: 16
<u>Batch size</u>: 300

<u>Parameter (gradients) size:</u>
500 * 500 * 2 = 500K
<u>Activation (gradients) size:</u>
500 * 300 * 2 = 300K

✓ Data Parallelism
- 500K * 2 * 4B * 16 = 64MB

✓ Model Parallelism
- 300K * 2 * 4B * 16 = 38.4MB

✓ Hybrid Parallelism
- 4 groups of GPUs, each group has 4 GPUs
- Model Parallelism among groups
  - 300K * 2 * 4B * 4 = 9.6MB
- Data Parallelism within each group
  - 500K / 4 * 2 * 4B * 4 = 4MB
- 9.6MB + 4 * 4MB = 25.6MB
- Save 33.3% communications!

Outline

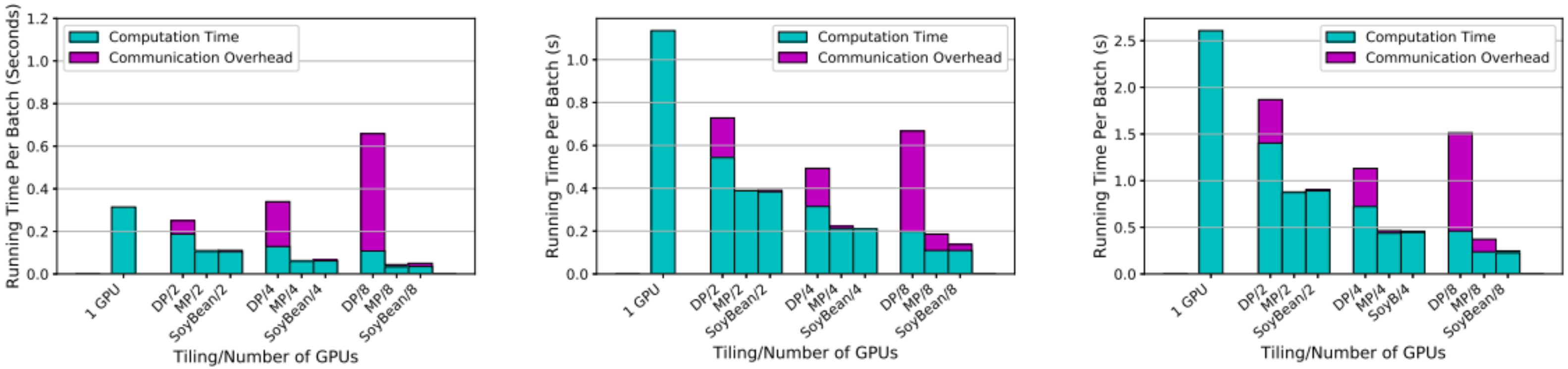
- Background & Motivation
- Problem Setup & Challenge
- Tofu solution
  - Single Operator
  - Whole Graph
- Experiments

Tofu Evaluation Setup

- Implemented in MXNet's NNVM dataflow optimization library.
- Multi-GPU evaluation
  - Amazon p2.8xlarge instance
  - 8 NVIDIA GK210 GPUs (4 K80)
  - 12GB memory per card
  - Connected by PCI-e (160Gbps bandwidth)

Communication Overhead Evaluation

- Per batch running time of a 4-layer MLP for DP, MP and TOFU.
- Hidden layer size: 8K/12K; Batch size: 512/2K



(a) Batch size = 512; Hidden Size = 8K

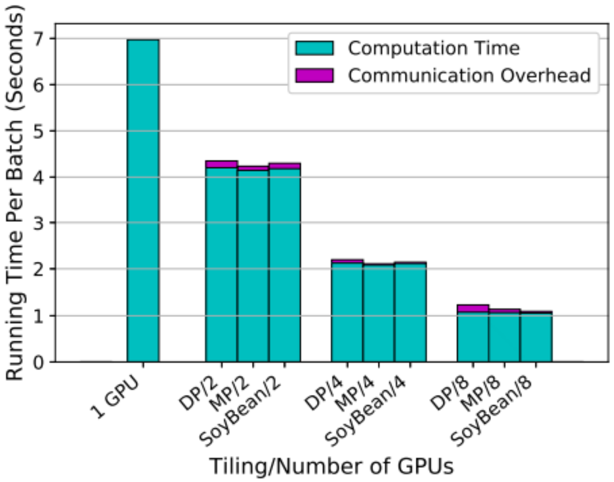
(b) Batch size = 2K; Hidden Size = 8K

(c) Batch size = 2K; Hidden Size = 12K

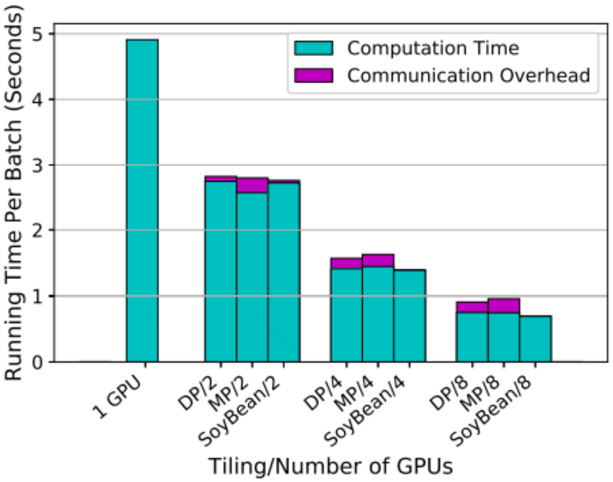
Figure 8: Runtime comparison of a 4-layer MLP for DP, MP, and SoyBean with different batch sizes and hidden sizes.

Communication Overhead Evaluation

- Per batch running time of a 5-layer CNN for DP and TOFU.
- Filter size: 2048/512



(a) Image Size = 6x6, Filter Size = 2048

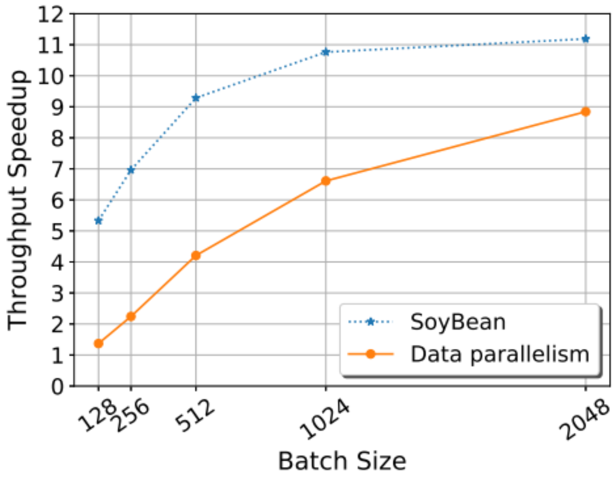


(b) Image Size = 24x24, Filter Size = 512

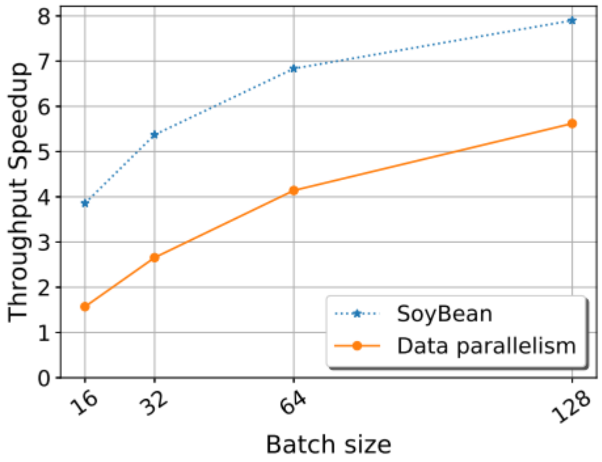
Figure 9: Runtime comparison of training a 5-layer convolutional neural network using DP, MP, and SoyBean. The batch size is 256.

Real Deep Neural Networks Evaluation

- Experimental setup: 1 machine, 8 cards.
- Baseline: 1 card



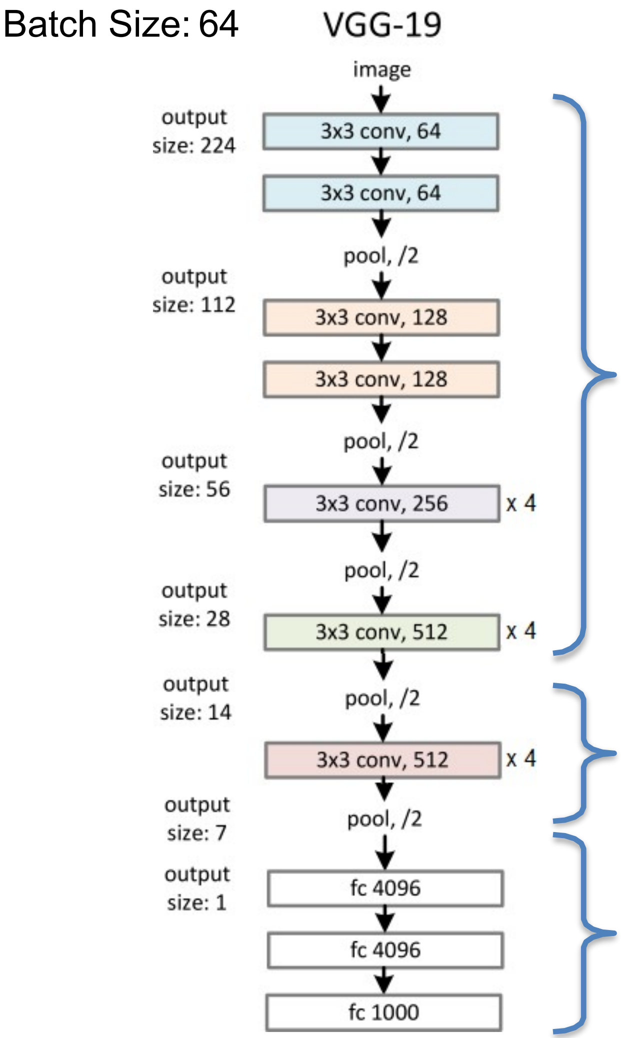
(a) AlexNet throughput speedup.



(b) VGG throughput speedup.

Figure 10: Throughtput comparison of SoyBean and data parallelism on 8 GPUs.

Tofu's tiling for VGG-19 on 8 GPUs



Data Parallelism

Hybrid Parallelism

- 8 GPUs into 4 groups
- Data parallelism among groups
- Model parallelism within each group (tile on channel)

Model Parallelism

- Tile on both row and column for weight matrices

Recap

- Data parallelism suffers from *batch-size-dilemma.*
- Other parallelisms exist but are hard to program.
  - Model parallelism, hybrid parallelism, combined parallelism, etc.
- Tofu automatically parallelizes deep learning training
  - Figure out distributed strategies for each operator.
  - Combine strategies recursively.
  - Proved to have least communication cost.

- Implemented in TVM (we may use ☺)

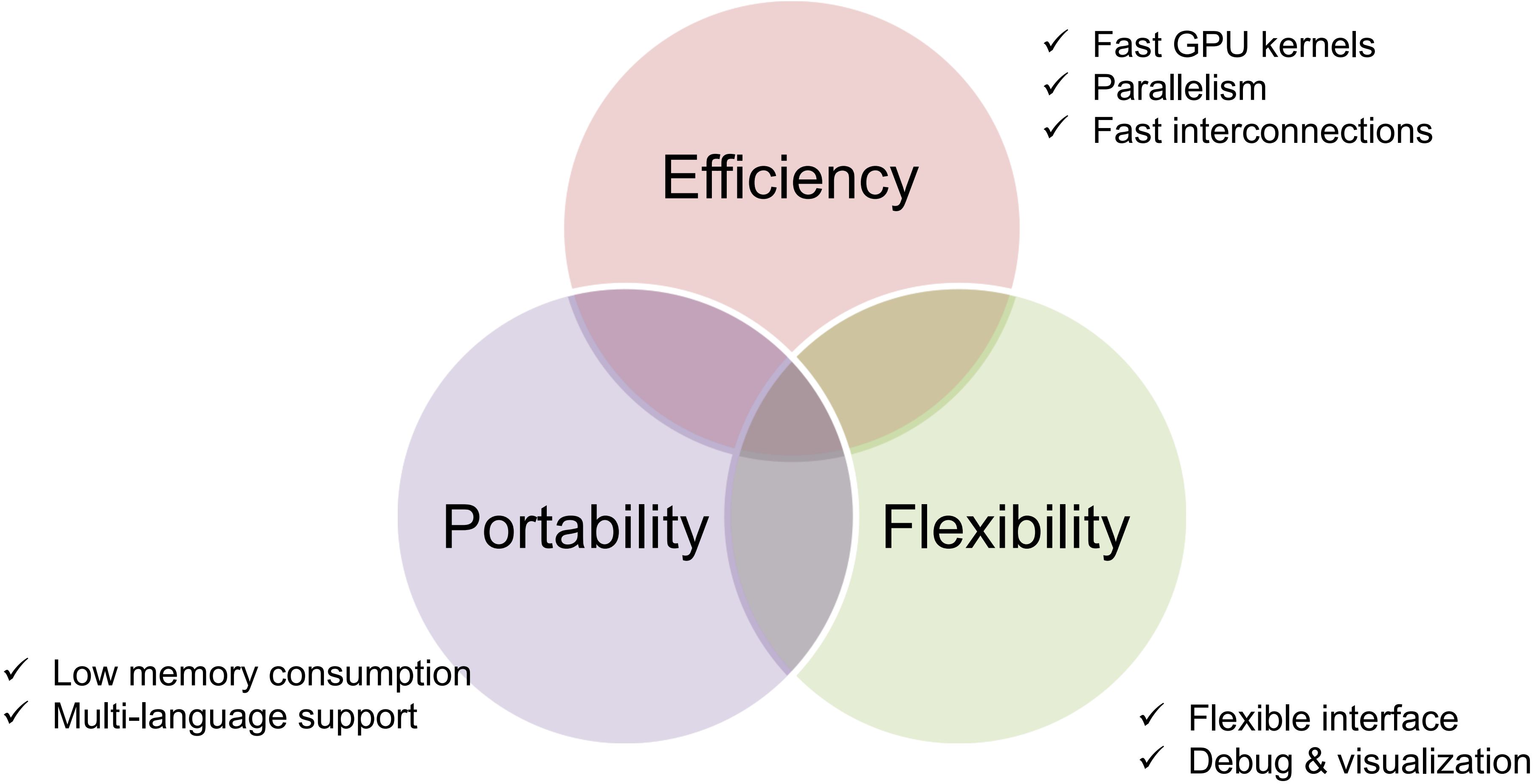
Q & A


YOU SHALL NOT PASS
zipmeme

Single Card Different Tilings

- Per batch running time for a 4-layers MLP network.
- Hidden layer size: 8192
- Partition dataflow to 8 workers but put them on the same GPU.

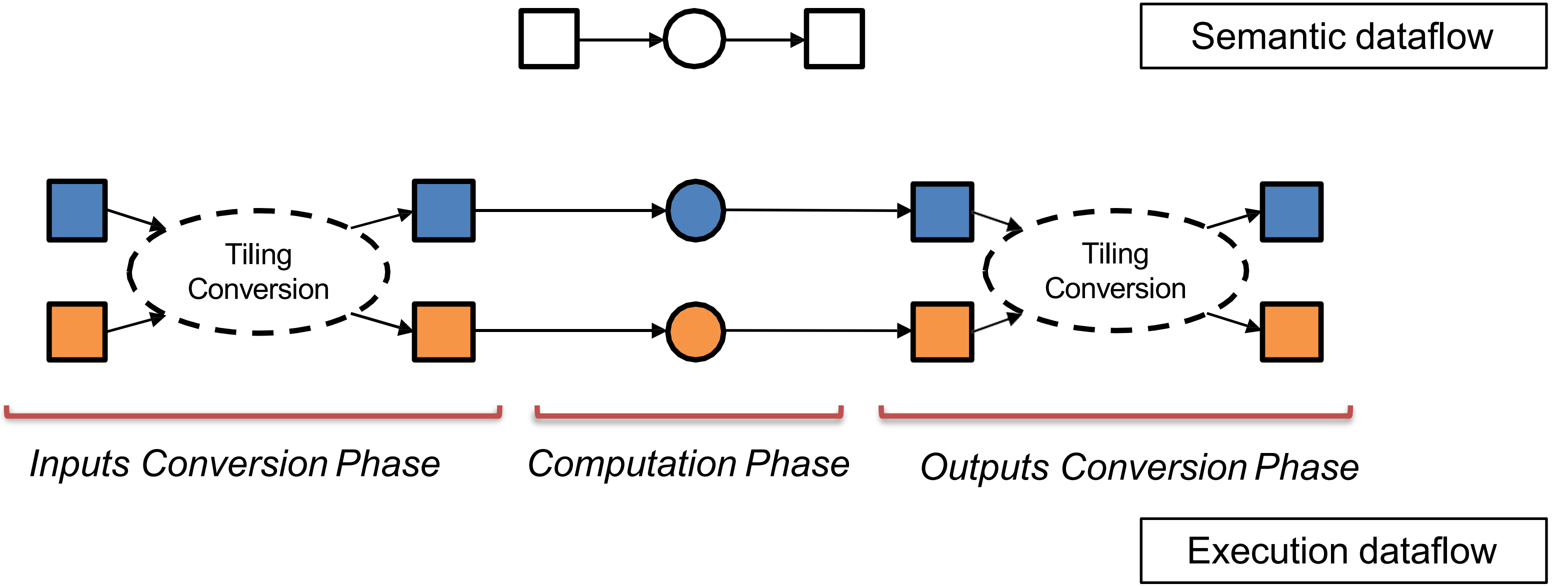
Batch Size	Single GPU	Single GPU w/ Tofu partitions
512	0.31s	0.19s
1024	0.56s	0.39s
2048	1.13s	0.73s



Efficiency
Portability
Flexibility
✓ Fast GPU kernels
✓ Parallelism
✓ Fast interconnections
✓ Low memory consumption
✓ Multi-language support
✓ Flexible interface
✓ Debug & visualization

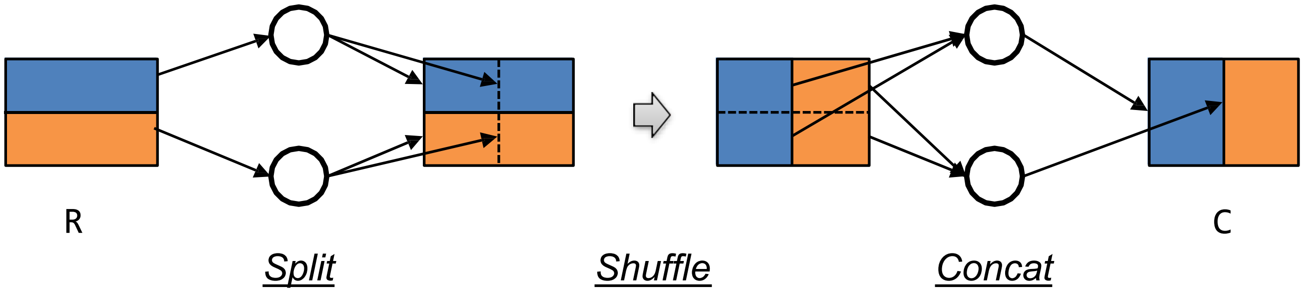
Construct Parallel Execution Graph

- Three-phase computation



Construct Parallel Execution Graph

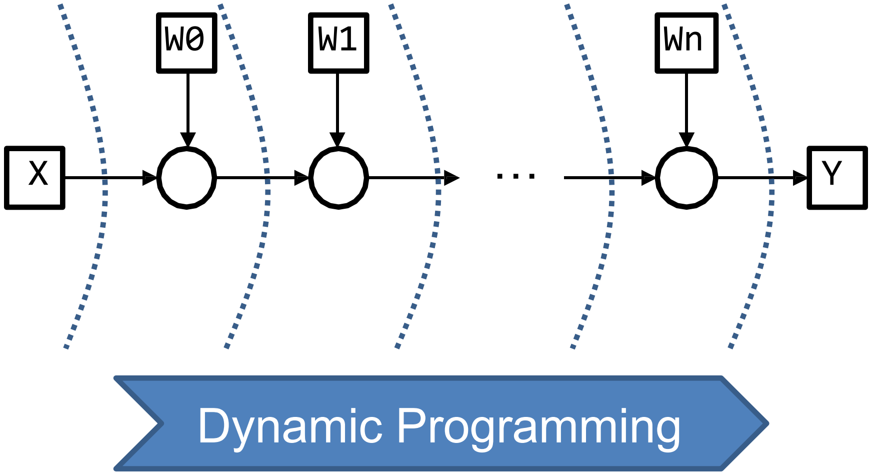
- Dataflow graph for tiling conversion.



One-cut Tiling Algorithm

- Given a dataflow graph G, find $\mathcal{T}_{min}: M_G \mapsto \{\mathrm{R,C,r}\}$ such that the communication cost of *all* matrix multiplications are minimized.
- Case #1:

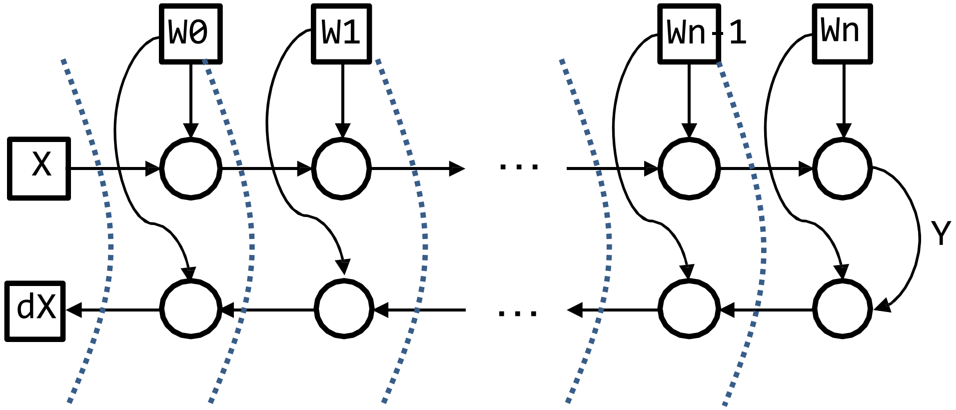
$$XW_0W_1 \ldots W_n = Y$$



One-cut Tiling Algorithm

- Case #2:

$$XW_0W_1 \dots W_n = Y$$
$$dX = YW_n^T W_{n-1}^T \dots W_0^T$$



Dynamic Programming

One-cut Tiling Algorithm

- Organize nodes in the dataflow graph into levels, such that for any node, **all** its neighbors are contained in the adjacent levels.
- BFS is one way to produce such levels.
- Dynamic Programming:

Initial condition:

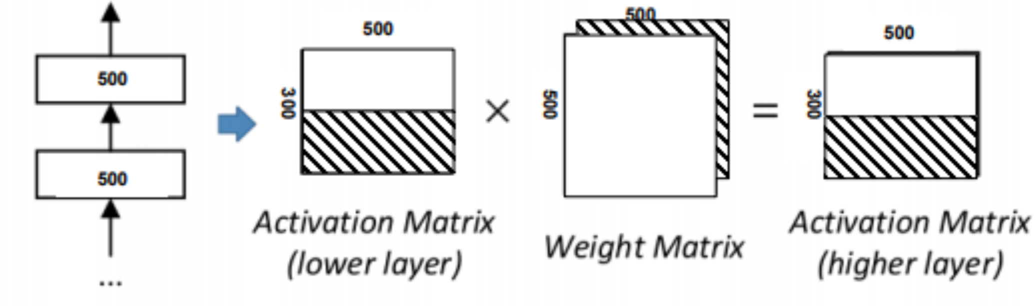
$$g_0(\tau_0) = \text{level\_cost}_0(\phi, \tau_0)$$

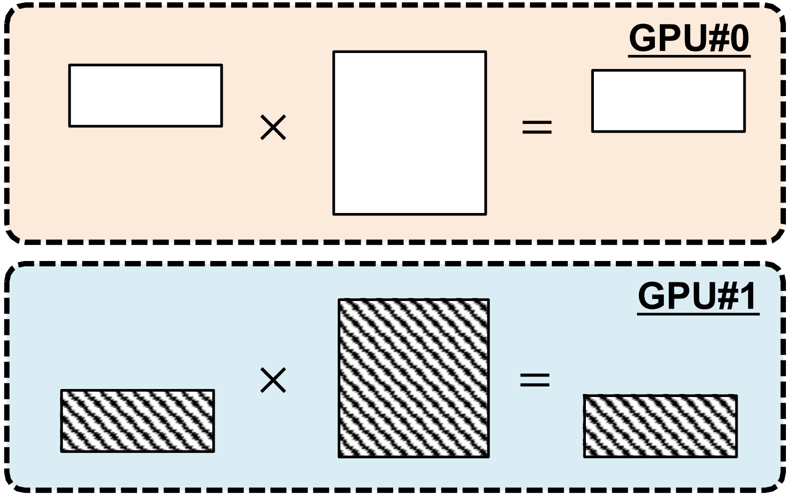
DP equation ($l \geqslant 1$):

$$g_l(\tau_l) = \min_{\tau_{l-1}} \{\text{level\_cost}_l(\tau_{l-1}, \tau_l) + g_{l-1}(\tau_{l-1})\}$$

Different ways of distributing matrix multiplication

Batch size: 300

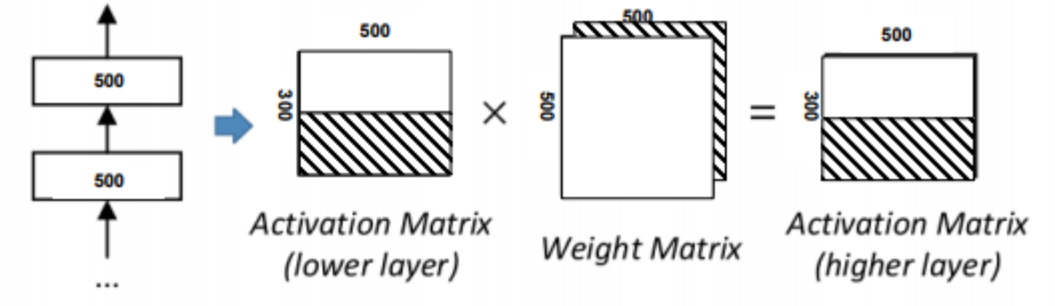


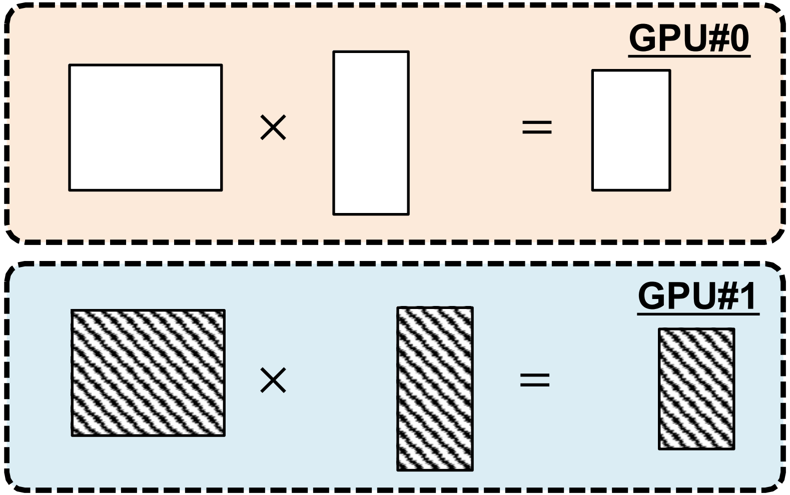


- Activation Matrix (lower layer) is **row-partitioned**
- Weight Matrix is **replicated**
- Acitvation Matrix (higher layer) is **row-partitioned**
- Data parallelism

Different ways of distributing matrix multiplication

Batch size: 300

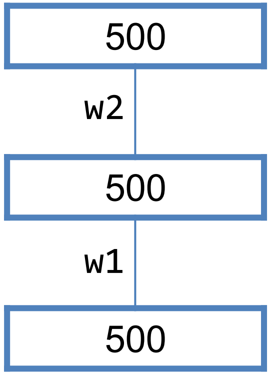




- Activation Matrix (lower layer) is **replicated**
- Weight Matrix is **column-partitioned**
- Acitvation Matrix (higher layer) is **column-partitioned**
- Model Parallelism

Which One is Better?

ToyNet Configuration



nGPUs: 16
Batch size: 300

Parameter (gradients) size:
500 * 500 * 2 = 500K
Activation (gradients) size:
500 * 300 * 2 = 300K

- ✓ Data Parallelism
  - 500K * 2 * 4B * 16 = 64MB
- ✓ Model Parallelism
  - 300K * 2 * 4B * 16 = 38.4MB
- ✓ Hybrid Parallelism?