Tofu: Parallelizing Deep Learning Systems with Automatic Tiling

Minjie Wang
State-of-art DL system is based on dataflow

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

Training: NVIDIA® Tesla® K80 synthetic data (1,8,16,32, and 64)

Valid batch size = 64 * 64 = 4096

* 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
MP is hard to program
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.

• DP and MP can be combined in a single layer
  – DistBelief (Dean, 2012)
  – Impossible to program with manual distributed strategy!
Tofu automatically distributes DL training

- User Program
- Semantic Dataflow Graph
- Parallel Execution Graph
- Execution

Automatic Conversion

Distributed Strategy with least communication

Tofu
Challenges

• What are the different ways to distribute each tensor operator?
• What is the globally optimal way of distribution
  – that minimizes communication?
Different ways of distributing matrix multiplication

Batch size: 300

- Activation Matrix (lower layer) is row-partitioned
- Weight Matrix is replicated
- Activation 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
- Activation Matrix (higher layer) is column-partitioned
- Model Parallelism
Operators can have different strategies

- Different matrix multiplications may choose different strategies.
Operators can have different strategies

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

Mathematical expressions and diagrams illustrating the concept of no communication in matrix multiplication operations.
Operators can have 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.
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.
- **Tofu** finds optimal strategy for deep learning in polynomial time:
  - “Layer-by-layer” propagations → graph with long diameter.
  - Use dynamic programming algorithm to find optimal strategy.
Combined strategies for one operator

Activation Matrix (lower layer) \times Weight Matrix = Activation Matrix (higher layer)

Batch size: 300
Combined strategy is sometimes better

• Fully-connected layer of 500 neurons with batch size 300.
• One combined strategy on 16 GPUs:
  – Model parallelism into 4 groups of GPUs (each group has 4 GPUs).
  – Data parallelism within each group.
  – Saves >33.3% communications than DP and MP.
Find combined strategies

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

\[ \delta_{total} = \delta_1 + 2\delta_2 \]

Step 1: Partition to two groups
Step 2: Apply the algorithm again on one of the group
Step 3: Apply the same strategy to the other group due to symmetry.
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)

Under submission. Contact wmjlyjemaine@gmail.com for more details.
Communication Overhead Evaluation

• Per batch running time of a 4-layer MLP for DP and Tofu.
• Hidden layer size: 8192; Batch size: 512
Real Deep Neural Networks Evaluation

- Experimental setup: 1 machine, 8 cards.

Figure 13: VGG throughput speedup.
Tofu’s tiling for VGG-19 on 8 GPUs

Data 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.
Q & A
YOU SHALL NOT PASS
One-cut Tiling Algorithm

• Given a dataflow graph $G$, find $T_{\text{min}}: M_G \mapsto \{R,C,r\}$ 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:

\[ XW_0W_1...W_n = Y \]
\[ dX = YW_n^TW_{n-1}^T...W_0^T \]
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:

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

  \[ DP\ equation\ (l \geq 1): \]

  \[ g_l(\tau_l) = \min_{\tau_{l-1}} \{ \text{level\_cost}_l(\tau_{l-1}, \tau_l) + g_{l-1}(\tau_{l-1}) \} \]
Which One is Better?

ToyNet Configuration

✓ 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!
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.

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Single GPU</th>
<th>Single GPU w/ Tofu partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.31s</td>
<td>0.19s</td>
</tr>
<tr>
<td>1024</td>
<td>0.56s</td>
<td>0.39s</td>
</tr>
<tr>
<td>2048</td>
<td>1.13s</td>
<td>0.73s</td>
</tr>
</tbody>
</table>
Efficiency
- Fast GPU kernels
- Parallelism
- Fast interconnections

Portability
- Low memory consumption
- Multi-language support

Flexibility
- Flexible interface
- Debug & visualization
Construct Parallel Execution Graph

• Three-phase computation
Construct Parallel Execution Graph

- Dataflow graph for tiling conversion.