Beyond Data and Model Parallelism for Deep Neural Networks
(FlexFlow)
Ron Davies
AU 21
• Introduction and motivation
• Solution summary
• Execution optimizer
• Performance
• Critiques
What is data parallelism?

What is model parallelism?
Data parallelism creates replicas of a full network on multiple devices with a subset of training data.

Benefits?
- Parallelism

Weaknesses?
- Inefficient for operations with large numbers of parameters

Model parallelism assigns disjoint portions of a network to each device.

Benefits?
- Reduced communication for weight synchronization etc.,

Weaknesses?
- Limited parallelism for operations
Is there something better?

**Hybrid parallelism**

How should we pick when to utilize each form of parallelization?
Alternative parallelization strategies:

- **Domain expert design:**
  - Use experience and prior knowledge to pick when to use data or model parallelism

- **OptCNN:**
  - Parallelizes DNNs by finding ways to exploit each operation

- **REINFORCE**
  - Utilizes reinforcement learning to learn efficient operation assignments for model parallelism

![Diagram showing example parallelization configurations for 1D convolution. Dashed lines show partitioning the tensor.](image)
• Introduction and review
• Solution summary
• Execution optimizer
• Performance
• Critiques
FlexFlow

- Operator Graph (input)
- Device topology (input)
- Optimizer
  MCMC search
  Execution simulator
- Distributed runtime
  This is the parallelization execution plan
Operator Graph

- They define a configuration for each operation
- Configurations partition an operation into tasks
- Each operation has parallelizable tasks
Device Topology

• The hardware setup is abstracted as a graph
• Edges represent connections
• Hardware used for connections are indicated (i.e. NVLink or PCI-e)
• Introduction and motivation
• Solution summary
• Execution optimizer
• Performance
• Critiques
Optimizer

- Limited budget loop
- Execution simulator
  - Creates a task graph for a strategy
  - Computes runtime of the strategy
- Search algorithm
  - Random and set start points
  - Suggests new parallelization strategies

Figure 2: FlexFlow overview.
Execution Simulator

• Assumptions:
  • Predictable with low variance independent of tensor contents
  • Full communication bandwidth utilization
  • Devices use FIFO
  • No delays on a device between receiving tensors and the end of a previous task

• Task graph:
  • Directed graph indicating dependencies
  • Communication edges are represented with a communication task
Execution Simulator

• Execution time:
  • The estimated time is an average of multiple runs of a task on a device (assumption 1)
  • This estimation will be used to compare strategies later
  • Communication tasks are given an estimation based on full bandwidth utilization (assumption 2)
Execution Simulator

• **Full Simulation:**
  • Global priority queue for all tasks
  • Queue based on dependencies
  • Rebuild the entire graph each time a new strategy is proposed

• **Delta Simulation:**
  • Only re-simulates the affected portions of the strategy
  • Updates tasks and dependencies in the task graph before enqueuing
  • Uses a Bellman-Ford-like approach to propagate updates to subsequent tasks
Execution Simulator

• Delta example: reduce the parallelization of $o_3$ and then update the graph

Figure 5: Simulating an example parallelization strategy. The tasks' `exeTime` and `device` are shown on the top of each column. In Figure 5c and 5d, the word "r" and "s" indicate the `readyTime` and `startTime` of each task, respectively, and the dashed edges represent the `nextTask`.
MCMC Search

- NP-Hard search space
- MCMC generates a sequence of points starting from a set point in a search space
- Search: stops with time budget or no improvement

  - An operation's configuration in the current strategy is selected at random and replace by a random configuration
  - Starting points/strategies include expert-design or data parallelism as well as randomly generated strategies
• Introduction and motivation
• Solution summary
• Execution optimizer
• Performance
• Critiques
Performance

• Cannot use PyTorch or TensorFlow because they only support paralelization of operations at the batch dimension through data parallelism

• FlexFlow is implemented in Legion

• FlexFlow performance on SOTA benchmarks are equivalent
Performance

• Tested on 2 different cluster styles
• Outperformed or found identical strategies as comparable frameworks
Performance

• Greater or equivalent throughput for every model
• Expert-designed strategies
  • CNNs: data parallelism for convolutions and pooling, model parallelism for densely-connected layers

Figure 7: Per-iteration training performance on six DNN benchmarks. Numbers in parenthesis are the number of compute nodes used in the experiments. The dash lines show the ideal training throughput.
Performance

- FlexFlow for Inception-v3 on a single node (4 GPUs) reduces execution time by 12% per iteration
- Also reduces parameter synchronization costs (communications) by 75%

Figure 13: The best strategy for parallelizing the Inception-v3 model on 4 P100 GPUs. For each operation, the vertical and horizontal dimensions indicate parallelism in the batch and channel dimension, respectively. Each GPU is denoted by a color. This strategy reduces the per-iteration execution time by 12% compared to data parallelism.
Simulator Accuracy

- If it is not accurate, then the solutions may be further from optimal
- Relative difference of 0-30% for operations

Figure 11: Comparison between the simulated and actual execution time for different DNNs and device topologies.
• Introduction and motivation
• Solution summary
• Execution optimizer
• Performance
• Critiques
Critiques

Can the operation simulations be more accurate?

More elaboration on the strategies that have been revealed

Jia et al., Beyond Data and Model Parallelism for Deep Neural Networks (FlexFlow)