UNIVERSITY OF CAMBRIDGE

Beyond Data and Model Parallelism for Deep Neural Networks

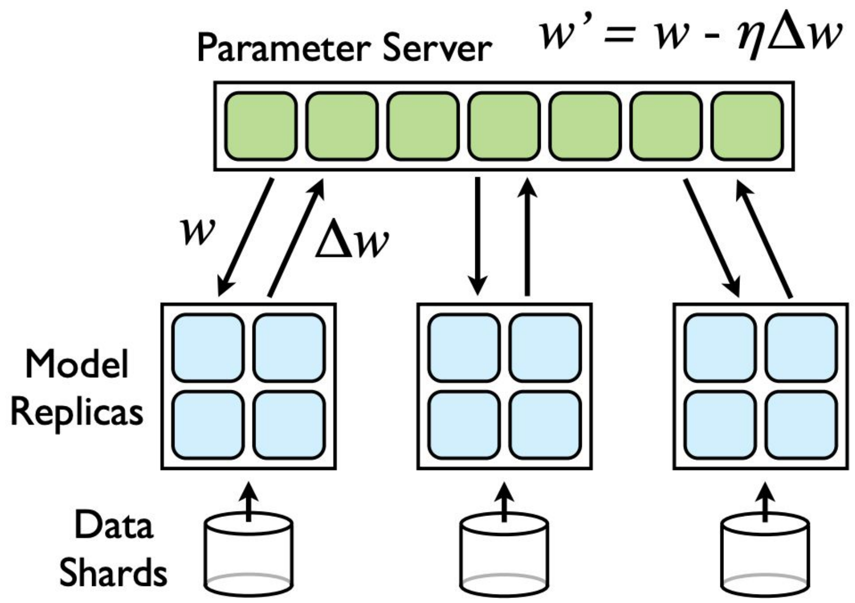
Zhihao Jia, Matei Zaharia and Alex Aiken

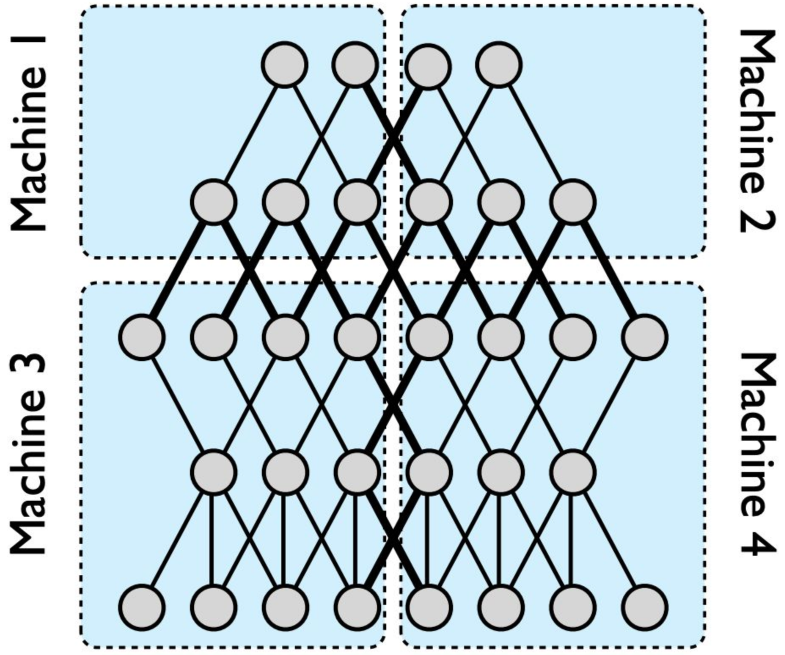
Cristian (cb2015@cam.ac.uk)

Content

Types of parallelism

TensorFlow, PyTorch, Caffe2 are mainly based on **data** and **model** parallelism.





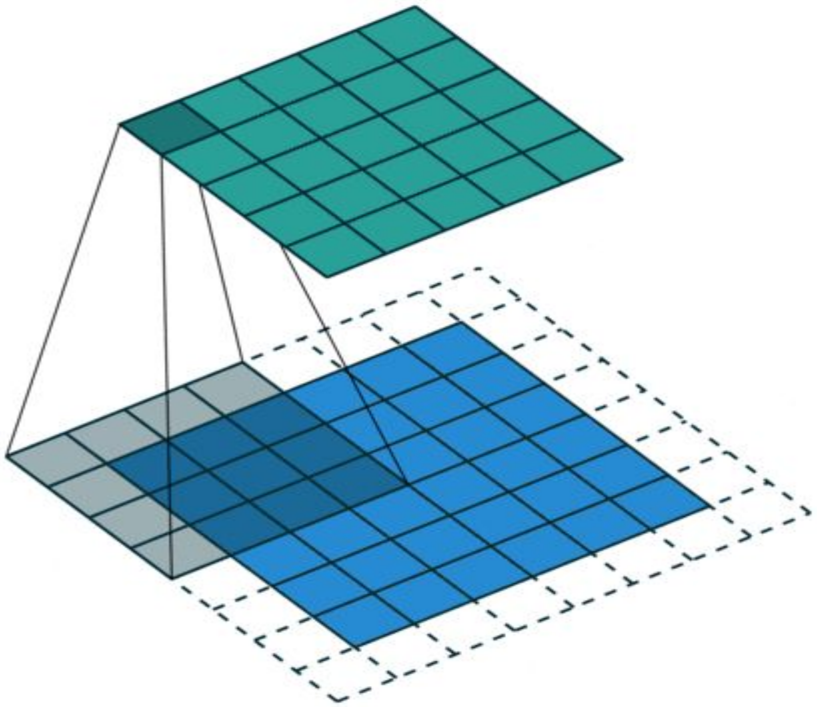
Data parallelism

Model parallelism

Images from **Large Scale Distributed Deep Networks** (Dean et al., 2012)

Types of parallelism

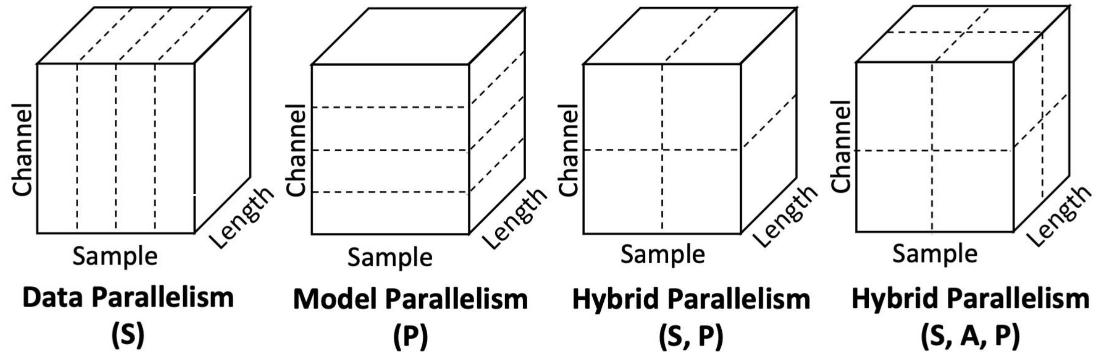
Something deep learning frameworks don't exploit is **operation level parallelism.** The convolution operation can be distributed along the channel or spatial dimensions.



The SOAP space

An obvious idea is to combine all types of parallelisation. However, one has to know first all the dimensions which can be parallelised in a Deep Neural Network.

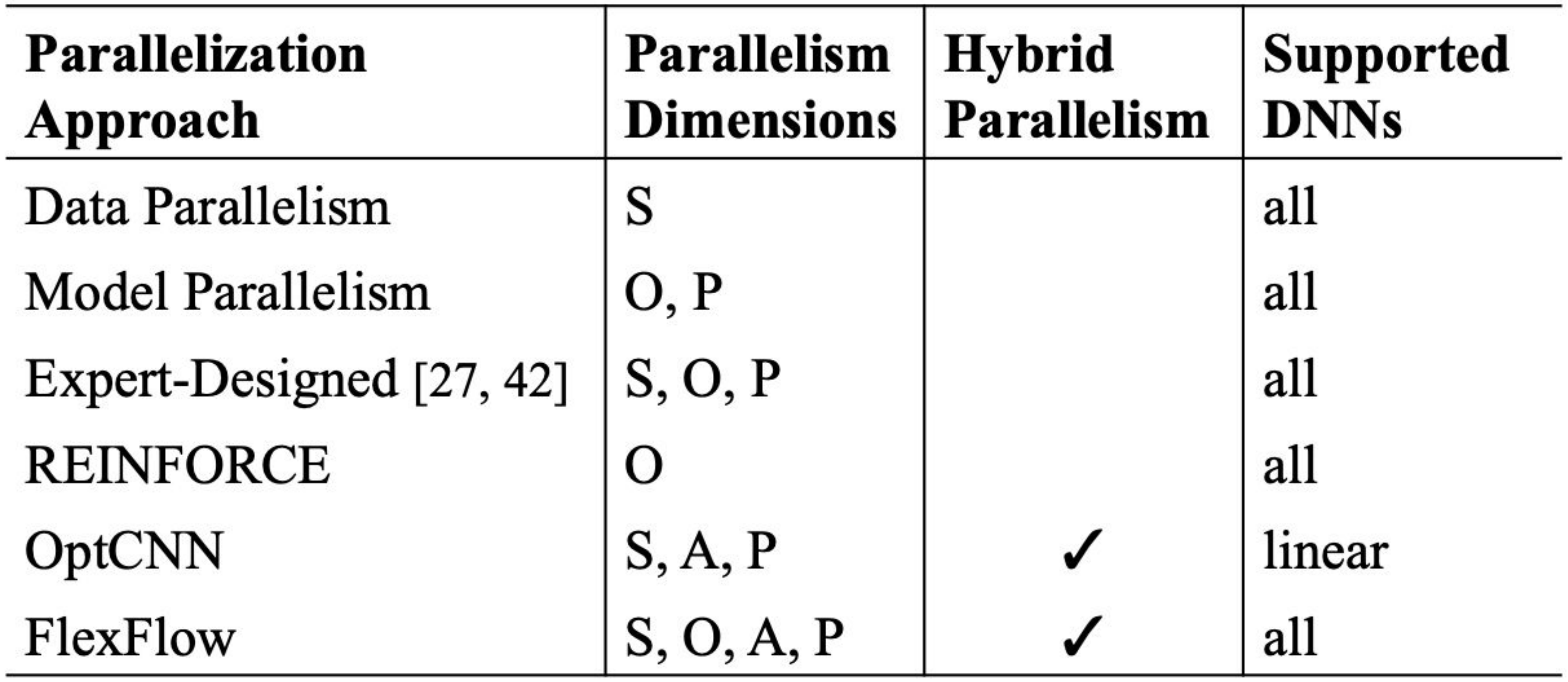
Sample-**O**peration-**A**ttribute-**P**arameter



The figure describes how a single operation can be parallelised across the SAP dimensions. But multiple operations can be executed in parallel if they do not depend on each other, hence the O dimension.

The SOAP space

How does the SOAP space fit with existing parallelization approaches?



Parallelization Approach	Parallelism Dimensions	Hybrid Parallelism	Supported DNNs
Data Parallelism	S		all
Model Parallelism	O, P		all
Expert-Designed [27, 42]	S, O, P		all
REINFORCE	O		all
OptCNN	S, A, P	✓	linear
FlexFlow	S, O, A, P	✓	all

FlexFlow

FlexFlow takes as input a graph of all the operations in the neural network and the topology of the network of devices the neural network will run on.

The **execution optimiser** searches for the best parallelisation strategy of the operations by using a simulation of the strategies run by the **execution simulator**.

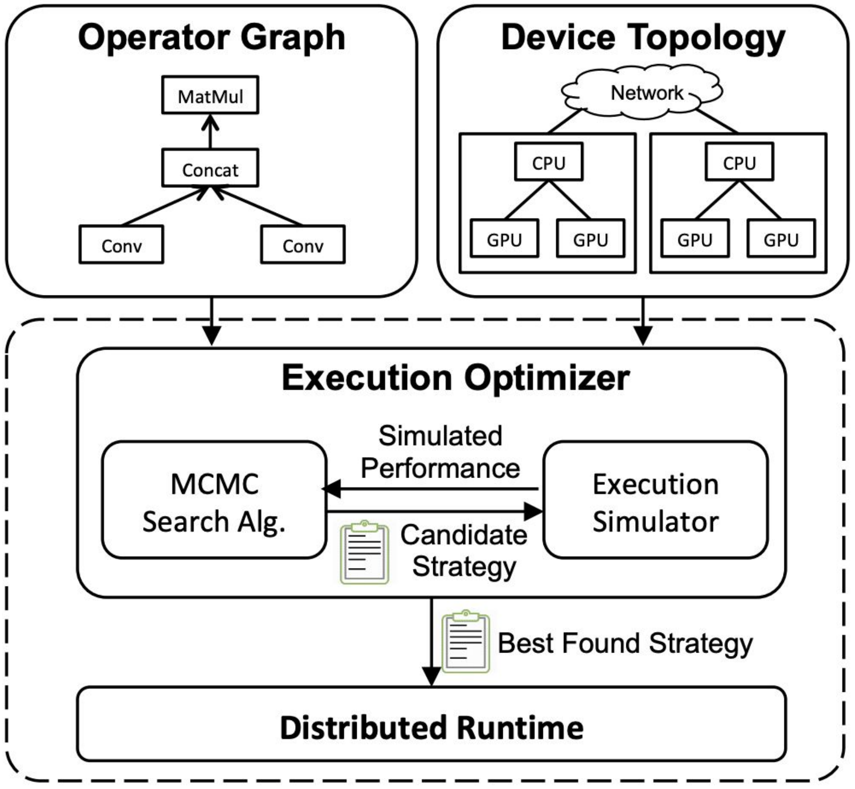


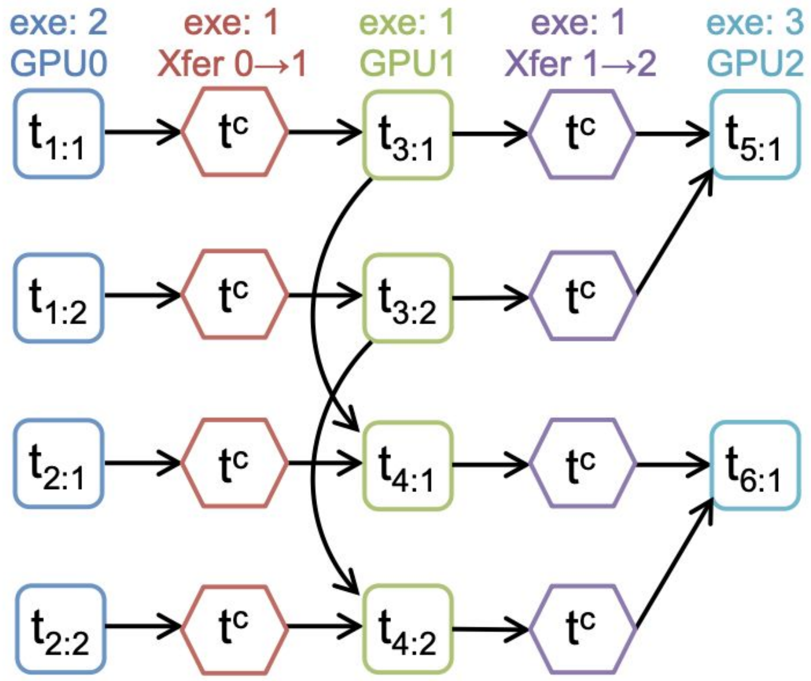
Figure 2: FlexFlow overview.

Execution Simulator: The Task Graph

Each operation o[i] in the operations graph has a configuration c[i] that describes how to split the output tensor in multiple tasks t[i][1],...,t[i][|c[i]|]. The execution simulator puts all these tasks together to create a **task graph** using the (o[i], o[j]) links from the input operation graph.

Nodes represent either **normal tasks** (square) or data **transfer tasks** (hexagon). Edges represent dependencies between tasks.

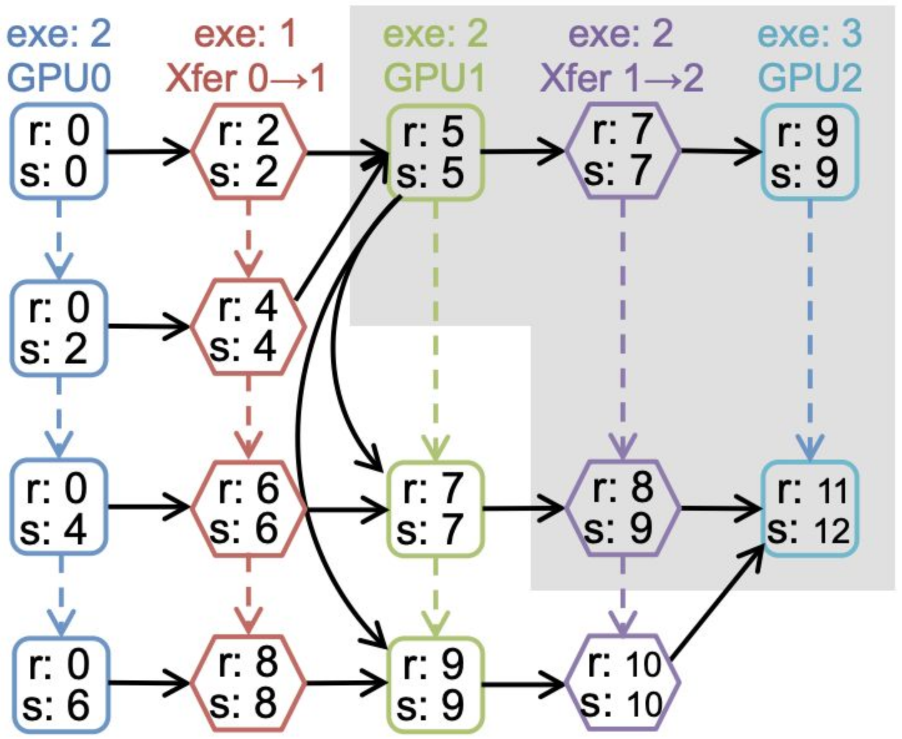
Transfer tasks are added if the tasks are executed by different devices.



Execution Simulator: The Delta Simulation Algorithm

Alternative approaches such as REINFORCE perform an actual execution of the operations to estimate the running time. However, this is expensive and FlexFlow simulates the execution of the task graph.

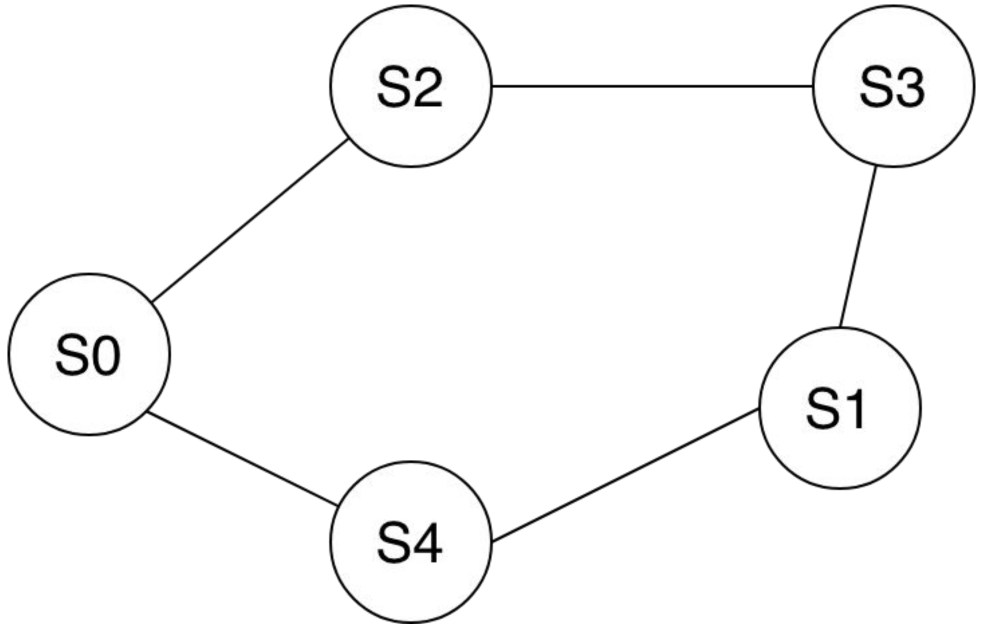
During the search procedure, the optimiser moves from one strategy to another by changing a single configuration. To avoid simulating everything again on the new graph, FlexFlow runs **Bellman-Ford** starting with a queue initialised with the new tasks to process only those tasks affected by the change.



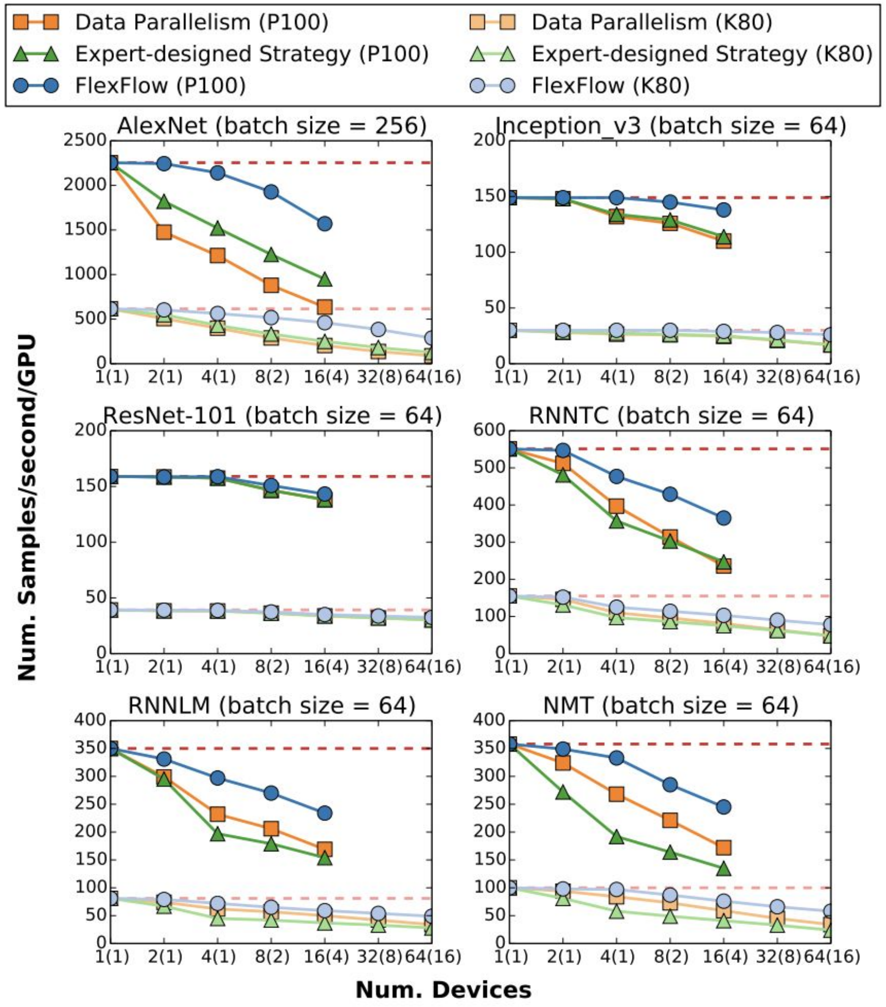
Execution Optimiser and MCMC

Finding the optimal assignment of tasks to devices is an NP-hard problem. As usual, an approximation method is the way to go. Flex flow uses the Metropolis-Hastings algorithm by assigning an execution time dependent distribution to the possible strategies:

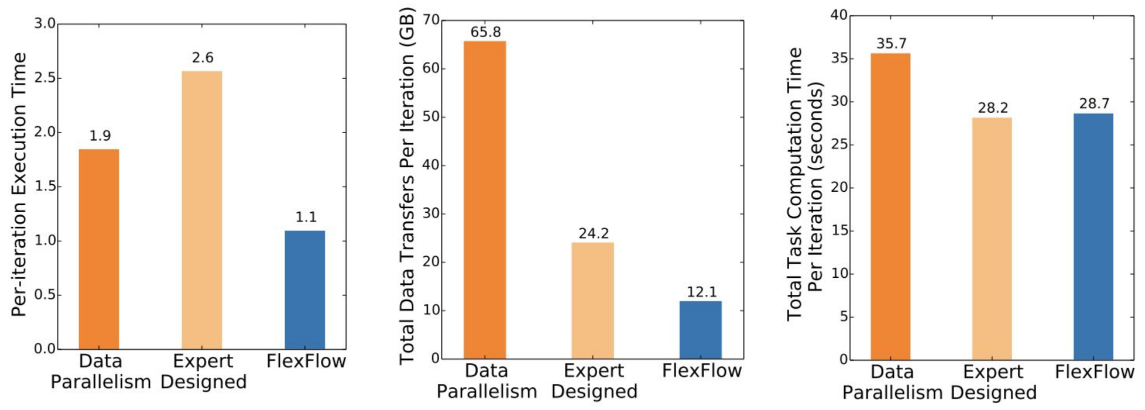
$$p(S) \propto \exp(-\beta \cdot \mathrm{cost}(S))$$



FlexFlow Evaluation: Samples / second / GPU



FlexFlow Evaluation: NMT Parallelization performance



(a) Per-iteration execution time.

(b) Overall data transfers per iteration.

(c) Overall task computation time per iteration.

Figure 8: Parallelization performance for the NMT model on 64 K80 GPUs (16 nodes). FlexFlow reduces per-iteration execution time by 1.7-2.4× and data transfers by 2-5.5× compared to other approaches. FlexFlow achieves similar overall task computation time as expert-designed strategy, which is 20% fewer than data parallelism.

UNIVERSITY OF CAMBRIDGE

FlexFlow Evaluation: Training curve Inception-V3

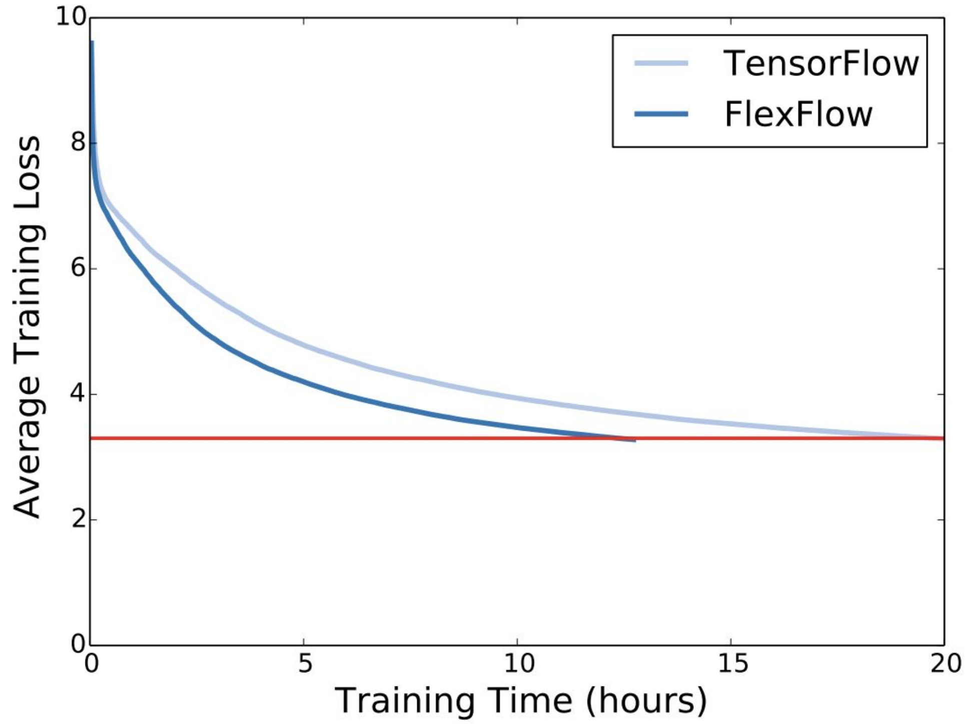
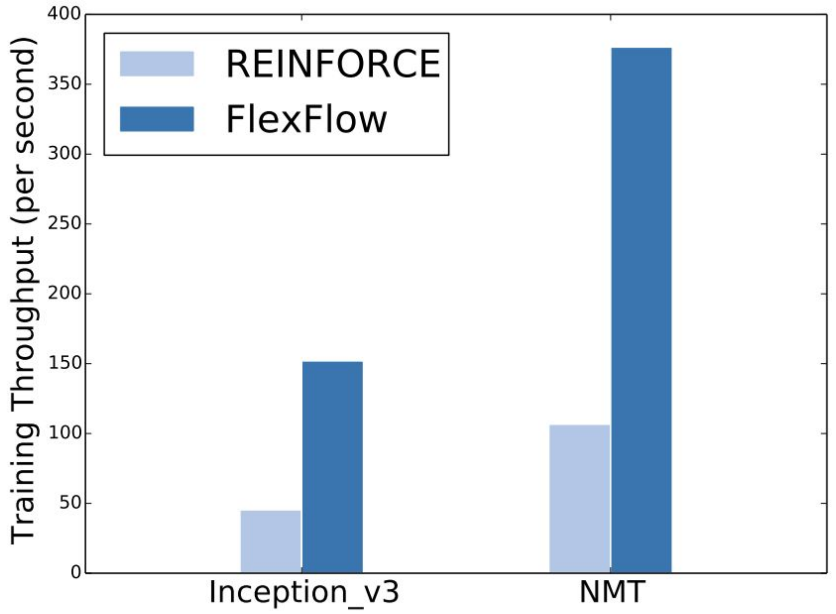
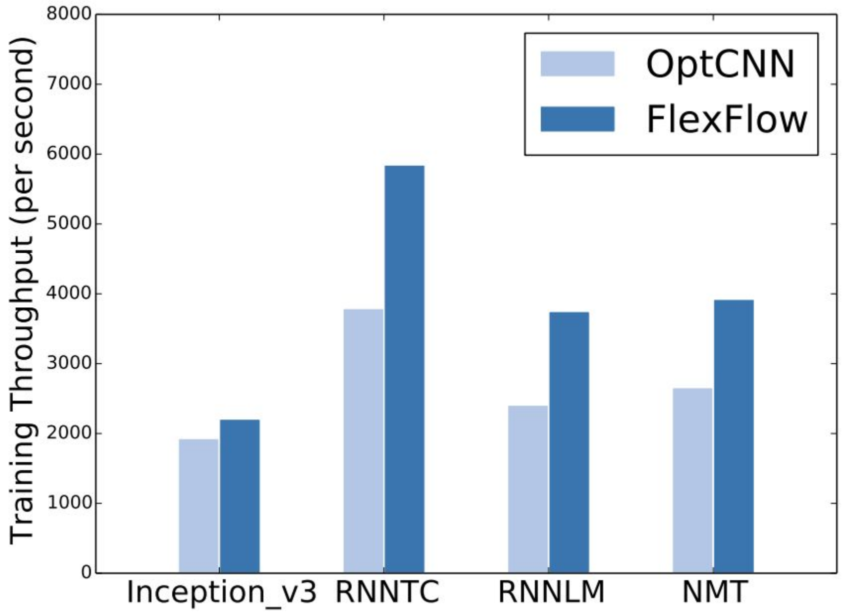


Figure 9: Training curves of Inception-v3 in different systems. The model is trained on 16 P100 GPUs (4 nodes).

FlexFlow Evaluation: Throughput comparison



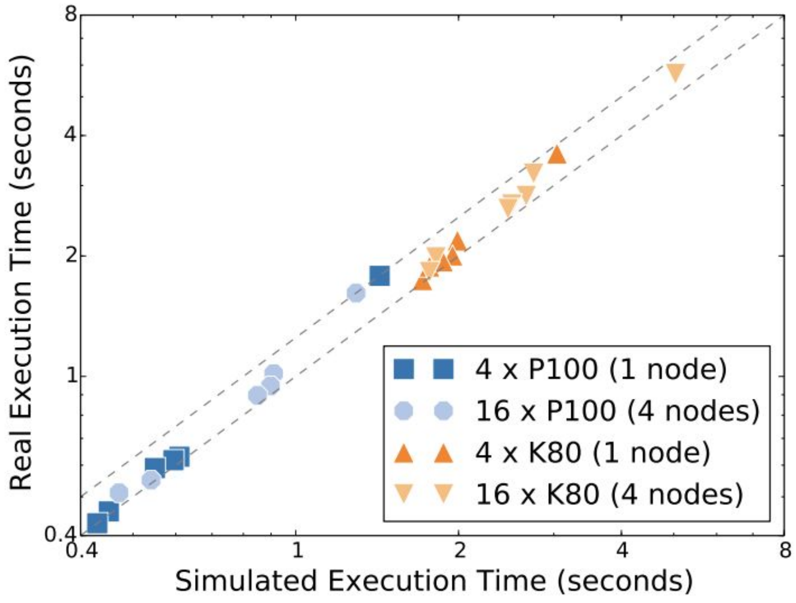
(a) REINFORCE



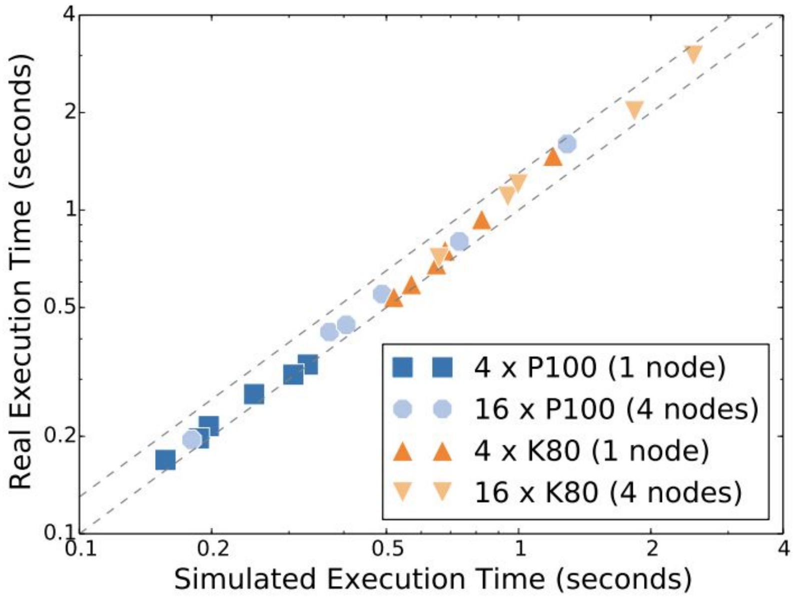
(b) OptCNN

Figure 10: Comparison among the parallelization strategies found by different automated frameworks.

FlexFlow Evaluation: Simulation accuracy



(a) Inception-v3



(b) NMT

Figure 11: Comparison between the simulated and actual execution time for different DNNs and device topologies.

UNIVERSITY OF CAMBRIDGE

Critique

The Good	The Bad
● Hybrid and granular optimisation ● Portable (just works on any device topology) ● Great user experience: just program the model and don’t worry about optimisation ● Easy way to insert expert knowledge	● The simulation algorithm is based on 4 assumptions. They do not hold for some ML algorithms. ● Assumption 2 (bandwidth can be fully utilised) might not hold in data center scenarios or from a certain cluster size in general.

Future work

- Some of the assumptions might be relaxed or even eliminated by combining simulation and execution. Simulation gives a very good insight on what is worth spending time on executing.
- Ability to configure the balance between time and the quality of the found strategy.

The End

Thank you!
Questions?