Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization

Colin Unger[†♠] Zhihao Jia[‡♭♠] Wei Wu[*◇] Sina Lin[§] Mandeep Baines[♭]
Carlos Efrain Quintero Narvaez[♭] Vinay Ramakrishnaiah[*] Nirmal Prajapati[*]
Pat McCormick[*] Jamaludin Mohd-Yusof[*] Xi Luo[♯] Dheevatsa Mudigere[♭]
Jongsoo Park[♭] Misha Smelyanskiy[♭] Alex Aiken[†]

Stanford University[†] *Carnegie Mellon University*[‡] *Los Alamos National Lab*[*]
NVIDIA[◇] *Microsoft*[§] *Meta*[♭] *SLAC National Accelerator Laboratory*[♯]

[Paper link]

Presented by Andrzej Szablewski
20/11/2024

Motivation

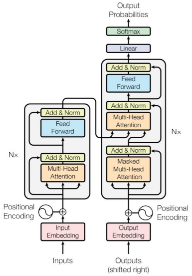
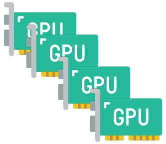


Figure 1: The Transformer - model architecture.



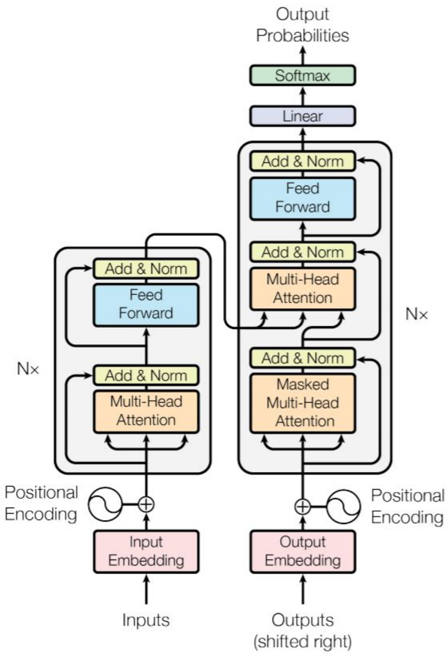
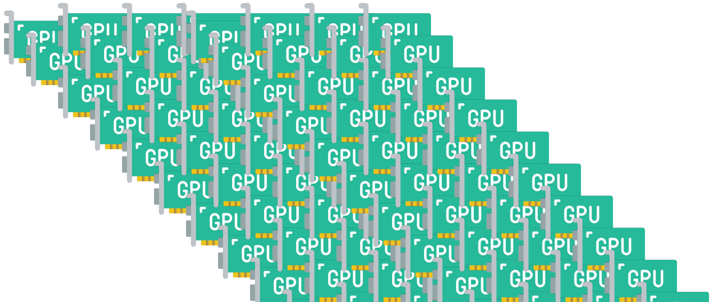


Figure 1: The Transformer - model architecture.



Optimisations

- Algebraic transformations
  - operator fusion
  - operator reordering

- Parallelism, many dimensions!
  - data
  - model
  - spatial
  - reduction
  - pipeline

Algebraic Transformations

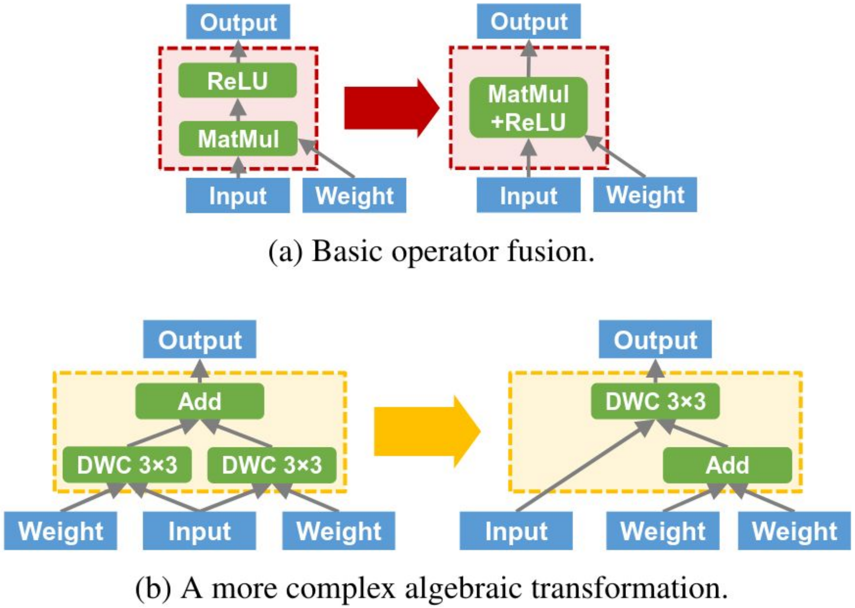
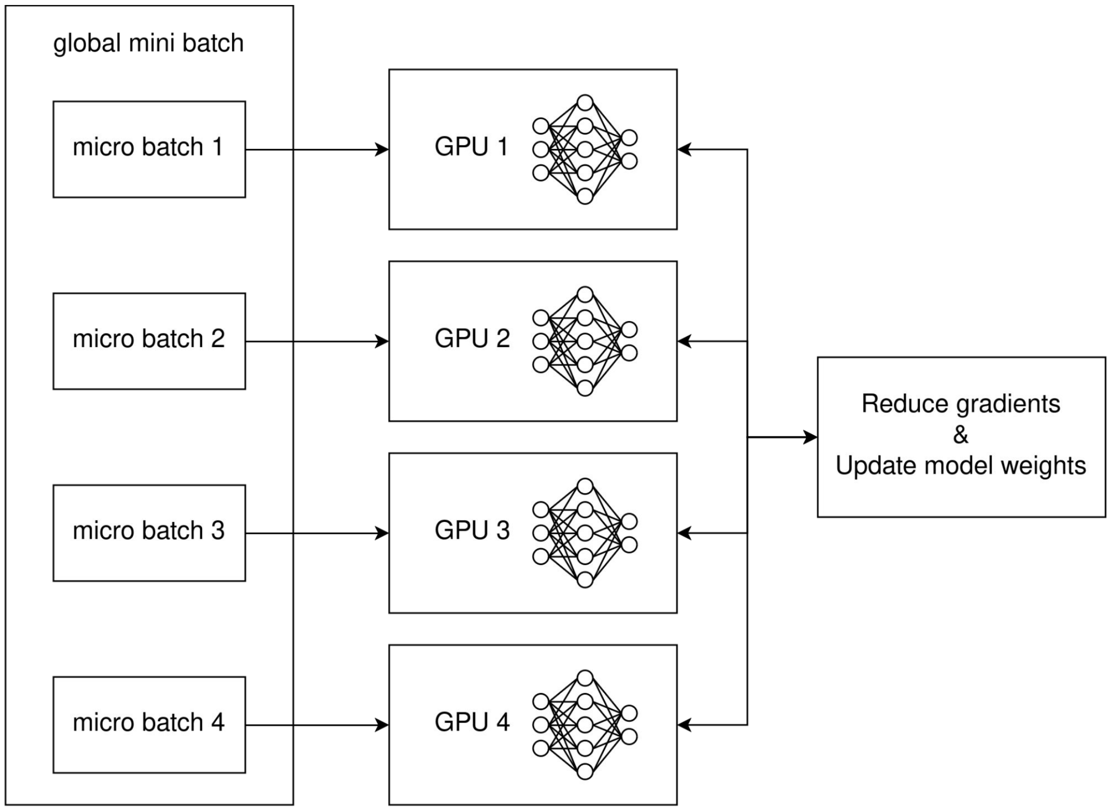


Figure 4: Example algebraic transformations. `DWC` stands for `DepthwiseConv` (i.e., depth-wise separable convolution).

Parallelism: Data parallelism



Cool, how do we apply them together?

- Simple approach:
  - Let’s apply algebraic transformations first
  - Then parallelise!

- The reverse order doesn’t work! (Why? 🤔)

Example: 2-layer MLP

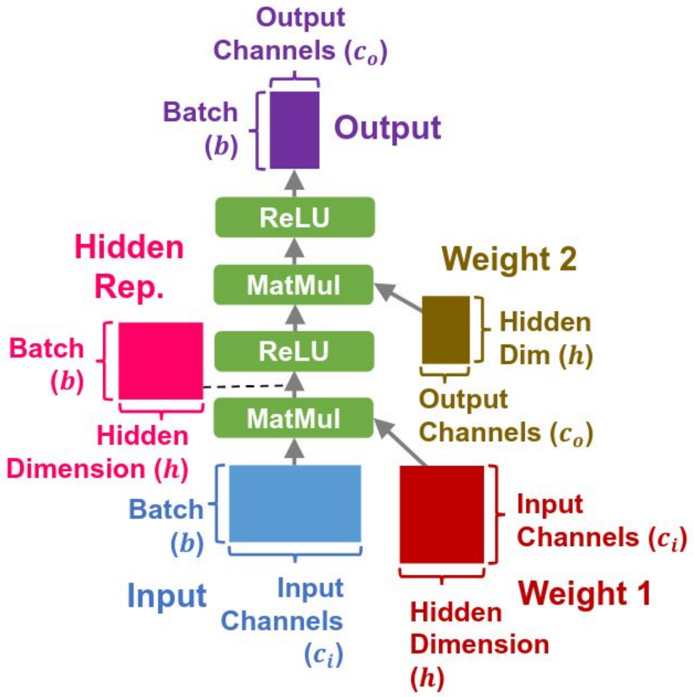


Figure 1: Computation graph for a 2-layer MLP.

Example: 2-layer MLP

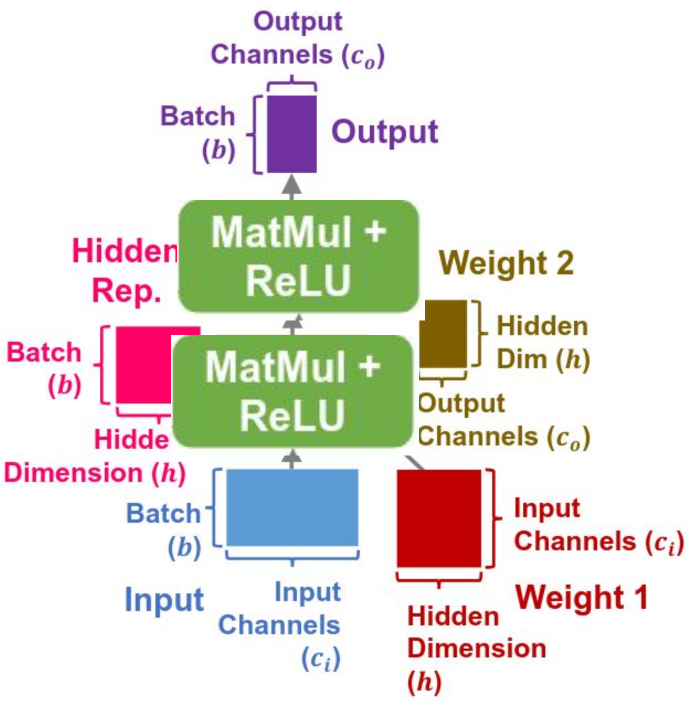


Figure 1: Computation graph for a 2-layer MLP.

Example: 2-layer MLP

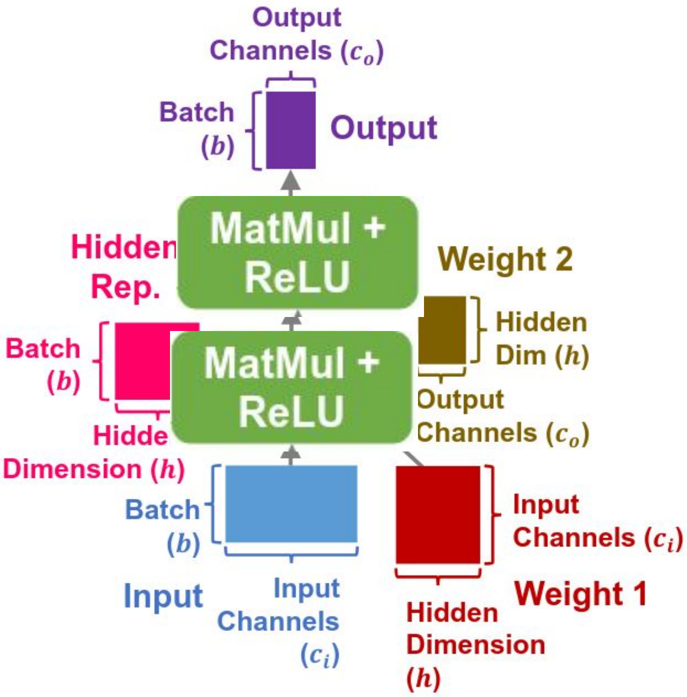


Figure 1: Computation graph for a 2-layer MLI

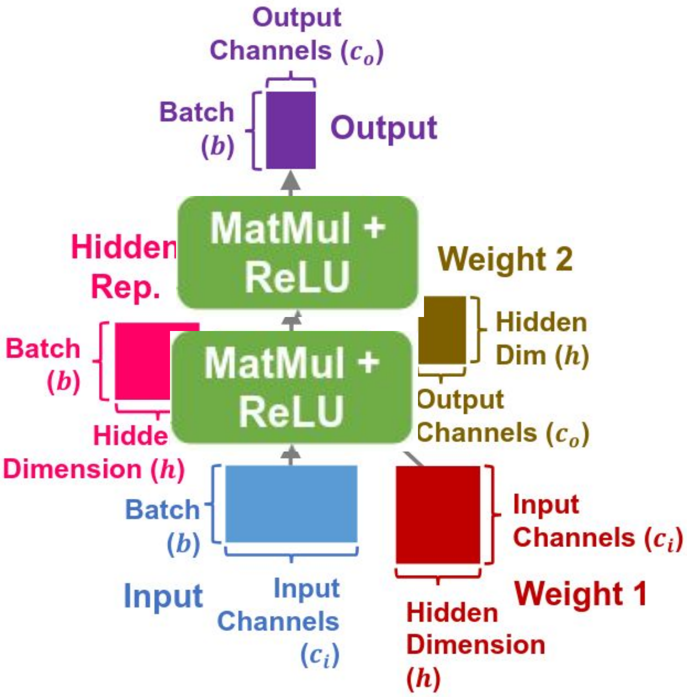


Figure 1: Computation graph for a 2-layer MLP.

Example: 2-layer MLP

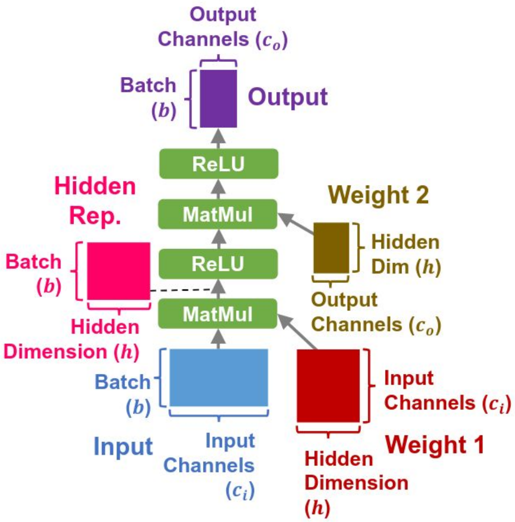
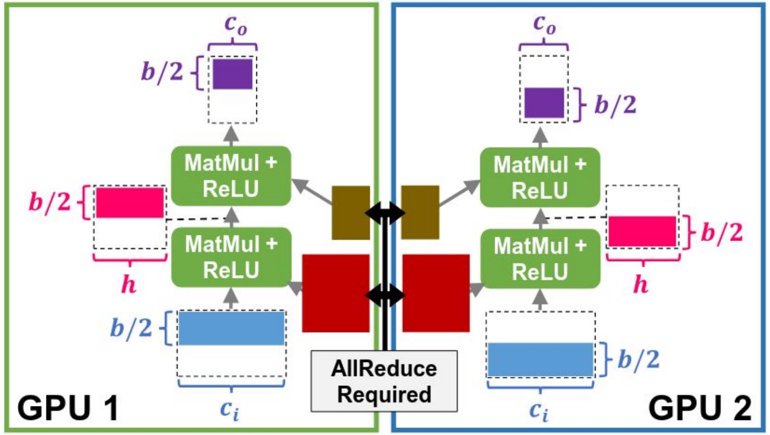


Figure 1: Computation graph for a 2-layer MLP.



(a) Sequential optimization.

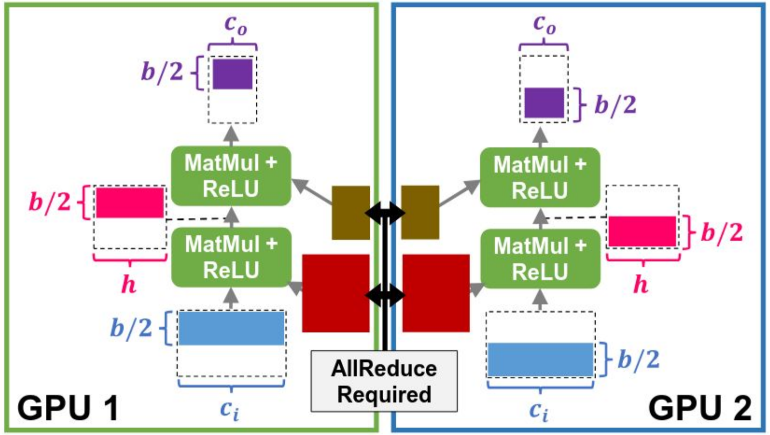
Communication cost?

- Need to communicate gradients of the weights!

$$2(c_i h + h c_o)$$

For MNIST:
813,056 * d

(d is the parameter size)



(a) Sequential optimization.

Example: 2-layer MLP

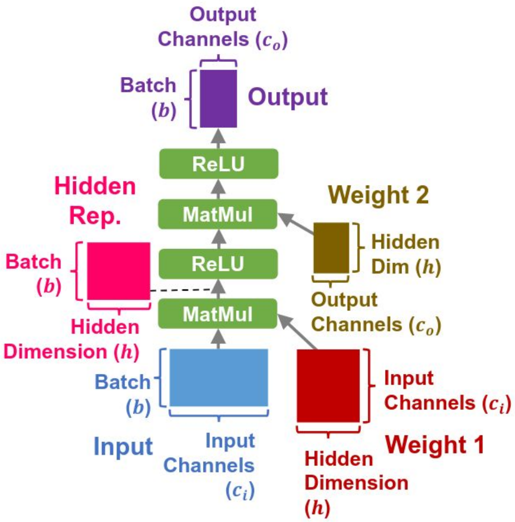
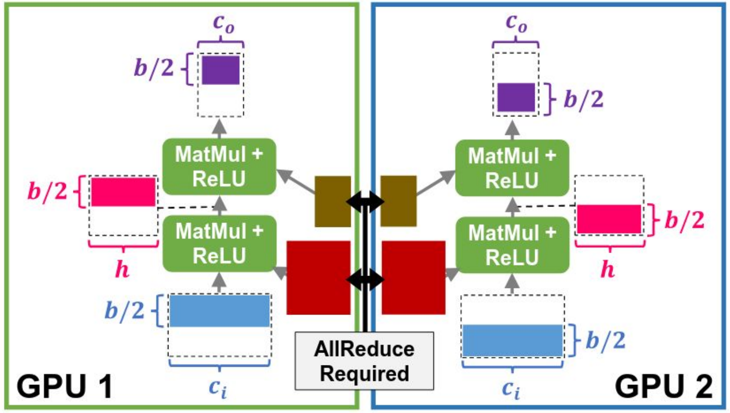
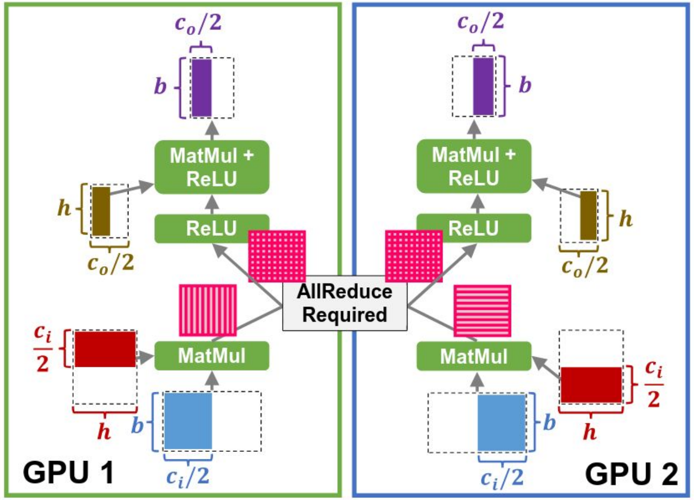


Figure 1: Computation graph for a 2-layer MLP.



(a) Sequential optimization.



(b) Joint optimization.

Communication cost?

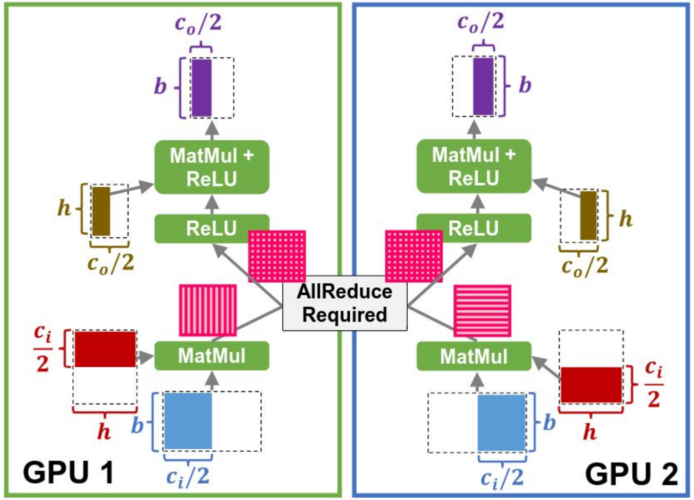
- This time just communicate activations and their gradients!

$$4bh$$

For MNIST:
131,072 * d

(d is the parameter size)

6x less!



(b) Joint optimization.

The approach

- Unified graph representation
    - Parallel Computation Graph
- Transformation generation + verification
- Joint optimisation

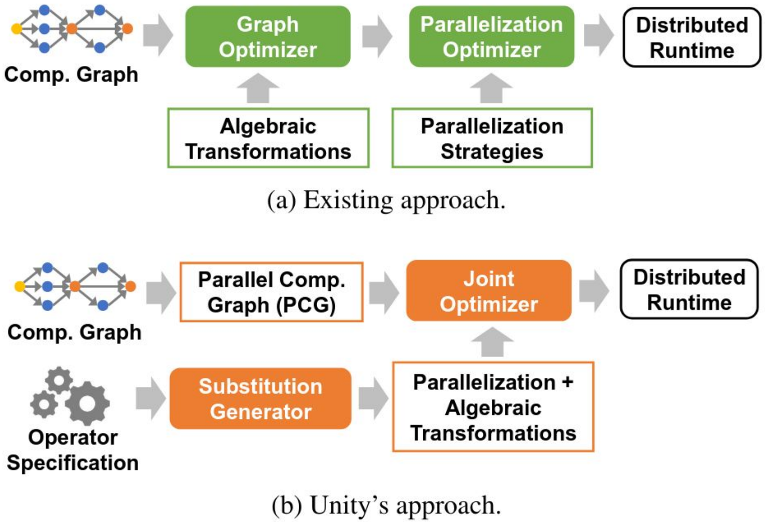
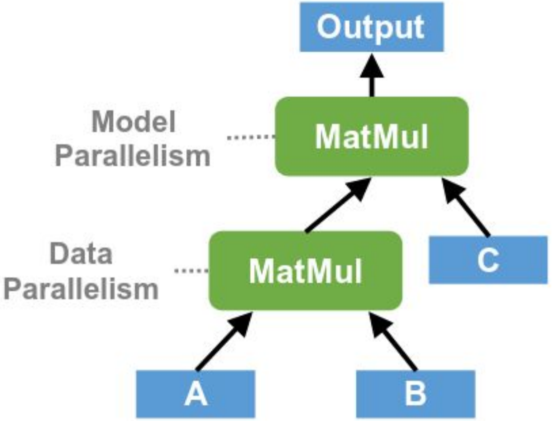


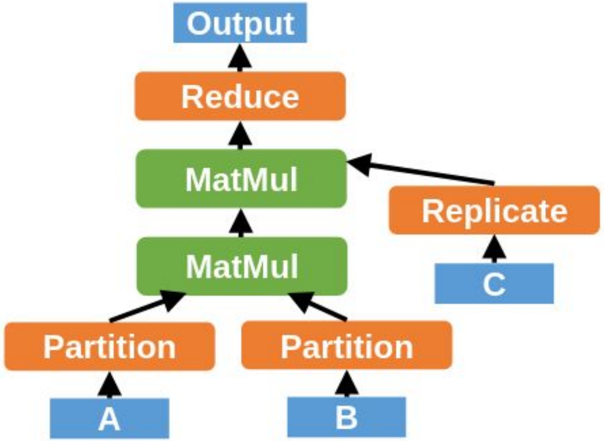
Figure 3: Comparing existing DNN frameworks and Unity.

Why Parallel Computation Graph?

- Evolving an annotated CG may lead to invalid parallelisations
- CG does not capture communication cost directly

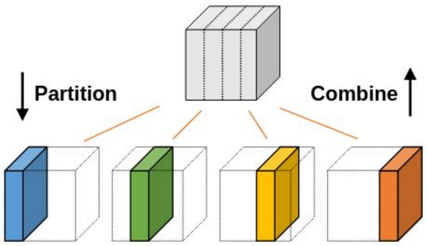


(a) Computation graph.

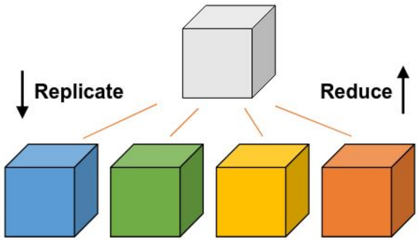


(b) Parallel computation graph.

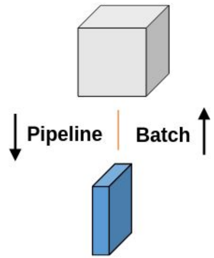
Parallelisation operations



(a) Partition/Combine.

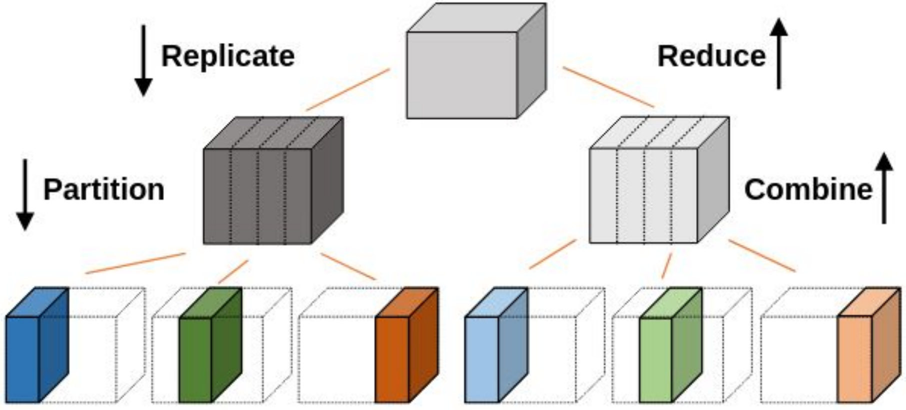


(b) Replicate/Reduce.

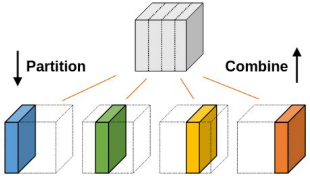


(c) Pipeline/Batch.

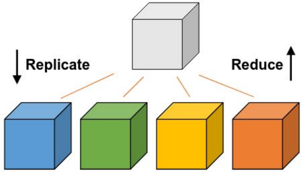
Parallelisation operations



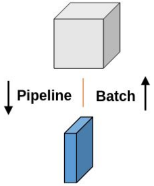
(d) Hybrid Parallelization.



(a) Partition/Combine.

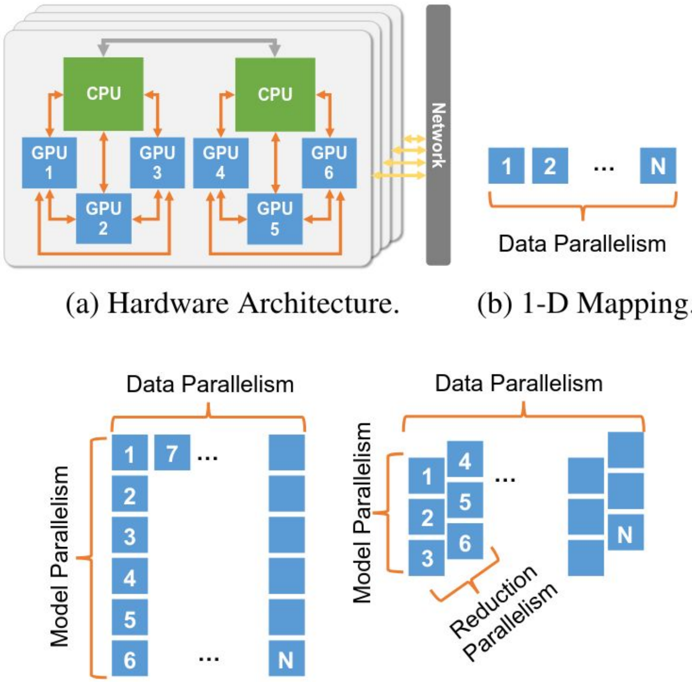


(b) Replicate/Reduce.



(c) Pipeline/Batch.

Machine mappings



(a) Hardware Architecture.
(b) 1-D Mapping.
(c) 2-D Mapping.
(d) 3-D Mapping.

Graph substitutions

- Unity generates all possible substitutions from a provided set of operators.

- Generated substitutions are formally verified!

Joint optimisation

Given:

- a PCG,
- a set of operator-level machine mappings,
- a set of PCG substitutions,

Find:

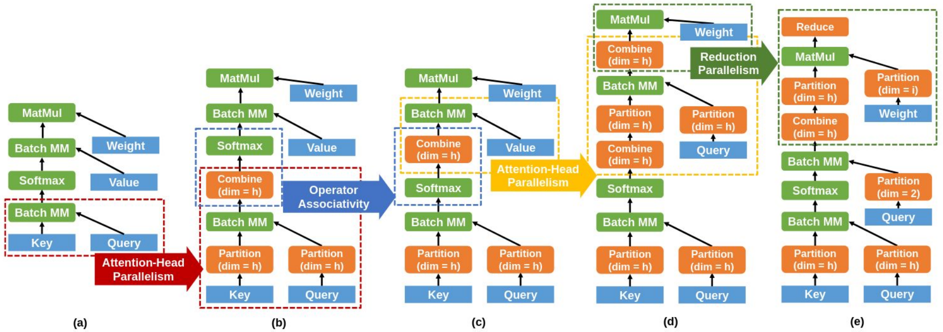
- a sequence of PCG substitutions
- and a machine mapping for the PCG,

minimising the per-iteration training time.

Joint optimisation

Three-level search:

1. Graph splitting
2. Substitution selection
3. Optimised machine mappings



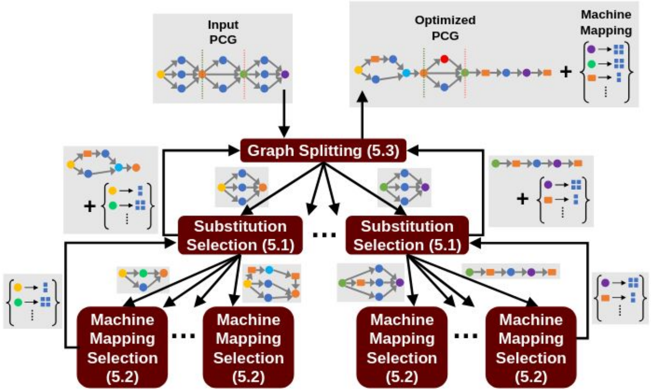
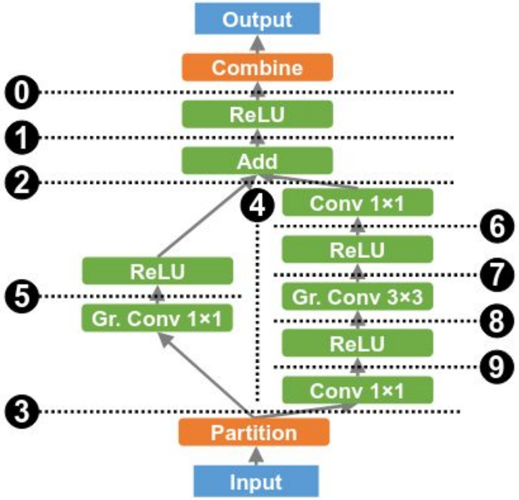


Figure 12: High-level depiction of Unity's hierarchical search.



Evaluation

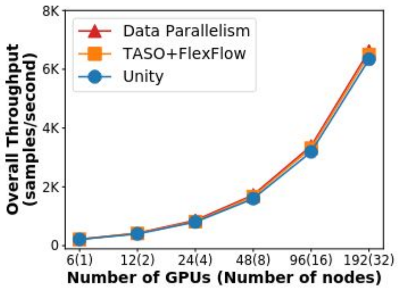
7 real-world neural architectures

- language models, image classifiers, etc.

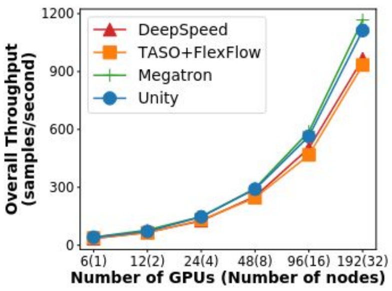
Unity outperforms other optimisation methods!

- search-based
- expert-crafted (e.g. DeepSpeed)

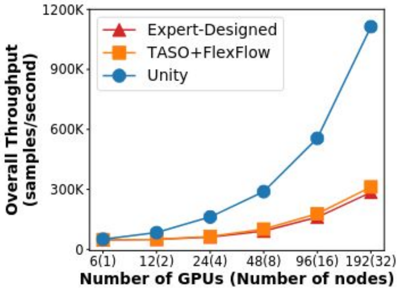
Evaluation



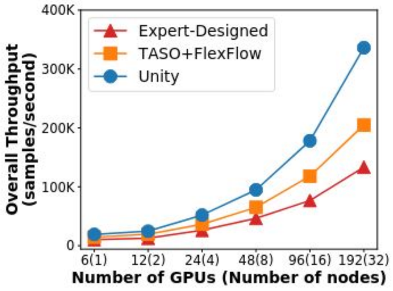
(a) ResNeXt-50.



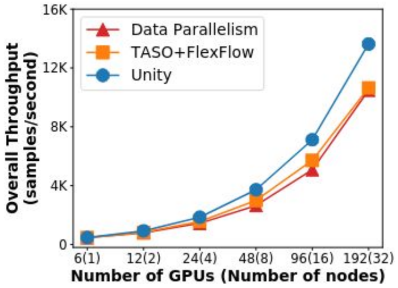
(b) BERT-Large.



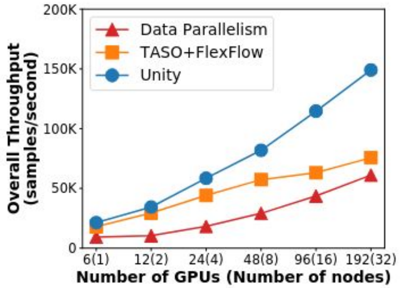
(c) DLRM.



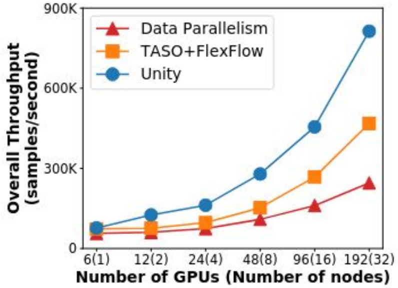
(d) CANDLE-Uno.



(e) Inception-v3.



(f) MLP.

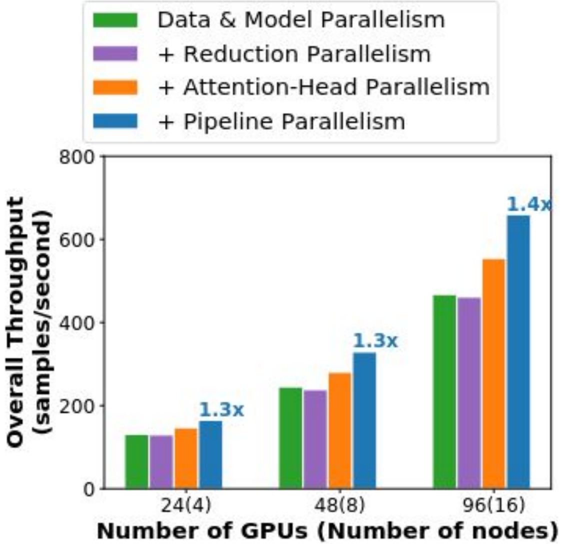


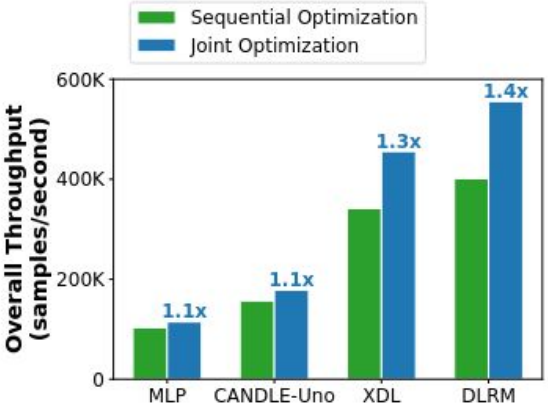
(g) XDL.

Evaluation

Hybrid strategies and operator-specific dimensions are critical!

Improvement up to 1.4x





Evaluation

Search time (🤔)

Table 2: Search algorithm ablation study. “Scaled” numbers are relative to the 2 GPU time with all optimizations enabled.

	All		w/o Split		w/o Cache+Split	
	Time	Scaled	Time	Scaled	Time	Scaled
6 GPUs (1 nodes)	57s	1×	4m 01s	4.3×	37m 01s	38.5×
12 GPUs (2 nodes)	1m 47s	1.9×	11m 15s	16.8×	> 1h	n/a
24 GPUs (4 nodes)	3m 00s	3.1×	> 1h	n/a	> 1h	n/a
48 GPUs (8 nodes)	5m 55s	6.1×	> 1h	n/a	> 1h	n/a

Limitations and Future Work

- NN architectures which do not exhibit nice parallelisable structure
- Other types of optimisations
- No reasoning about the memory usage!
- Simple cost model
- Simple support for pipeline parallelism (no of interleaving pipeline-parallel and non pipeline-parallel operators.)

Thank you!