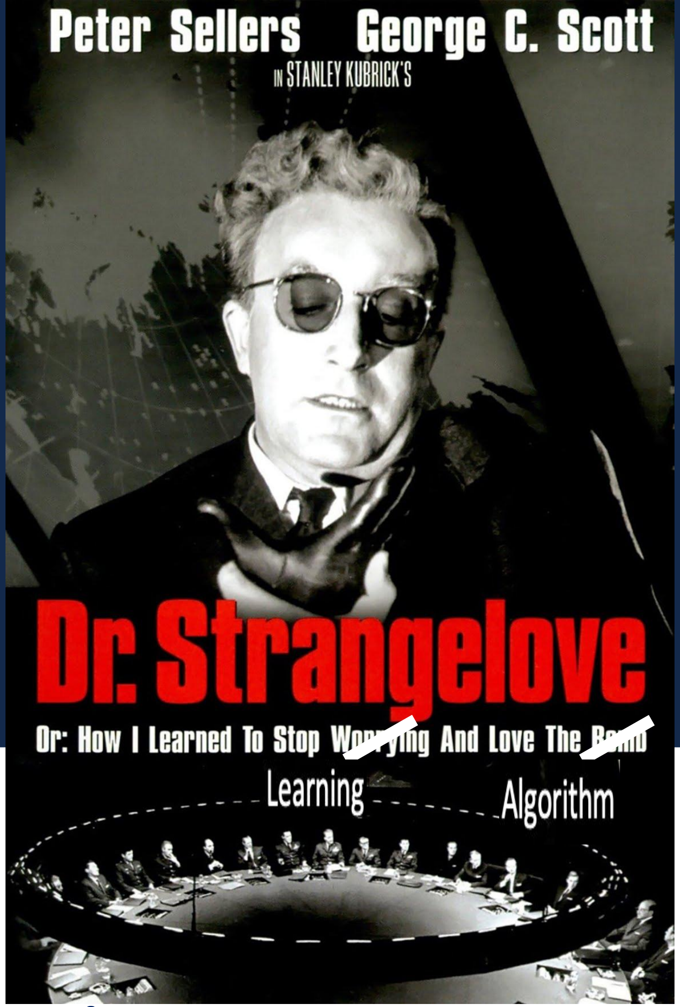
Hammer or Gavel?



Indranil Gupta

Professor, CS, University of Illinois at Urbana Champaign

(Joint work with students Beomyeol Jeon[†], Linda Cai[*], Chirag Shetty[†], Pallavi Srivastava[◊], Jintao Jiang[‡], Xiaolan Ke[†], Yitao Meng[†], Cong Xie[**]

[†]UIUC, [*]Princeton University, [◊]Microsoft, [‡]UCLA, [**]ByteDance

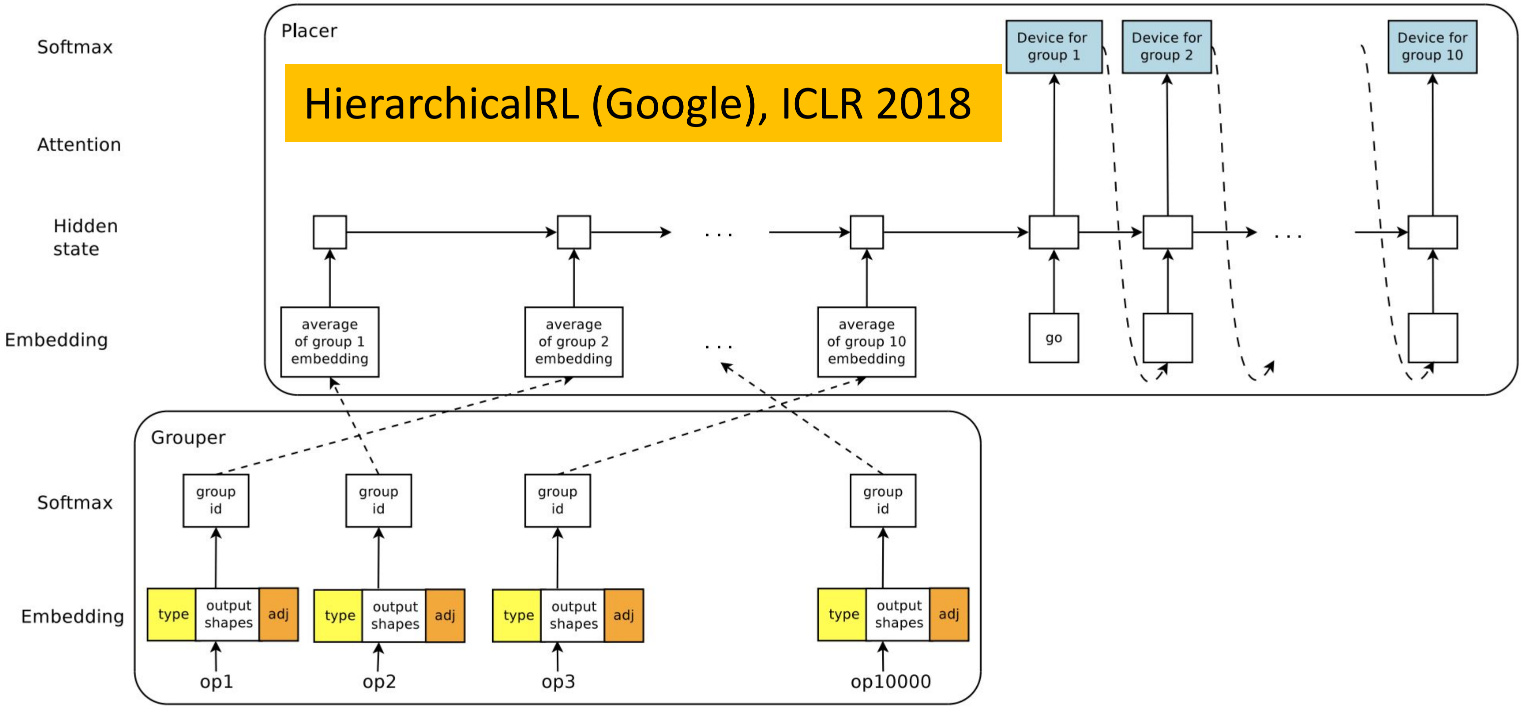
Applications of DNNs to Systems Problems

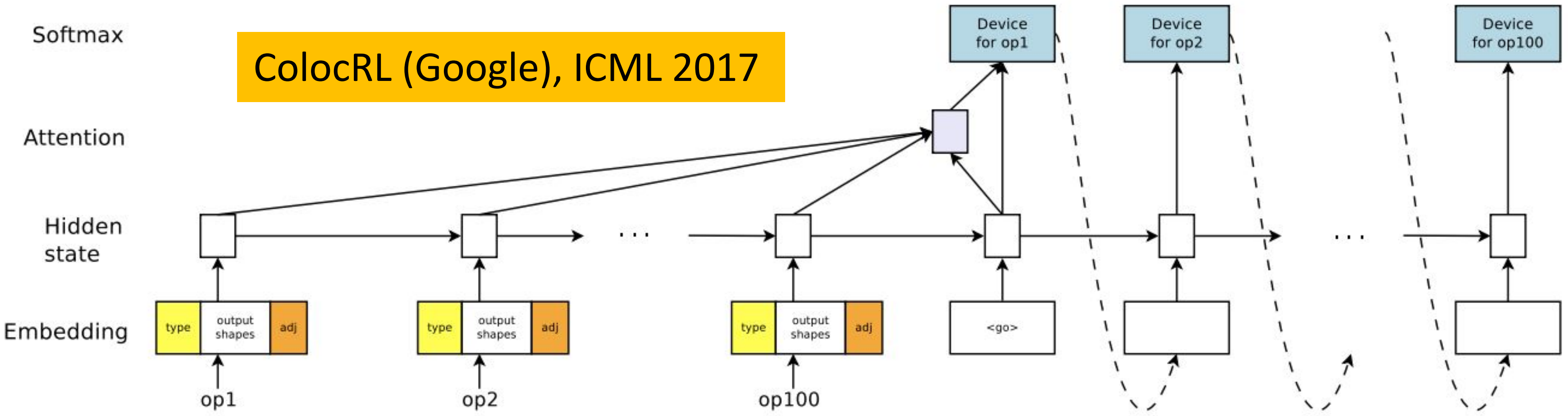
- *"When you have a hammer, everything looks like a nail" – Abraham Maslow/Abraham Kaplan /Mark Twain*

Some Real Reasons Companies Throw DNNs at a Problem

- Simple models often work well
- They have the datacenter resources
- They have the data
- You don't have to be (that) creative and design an algorithm
- Many (not all) in industry production teams…prefer not to read papers
- "Something will work!"
- And yet…
  - Anecdotal evidence: "DNNs have high accuracy in papers, but if you get 25% accuracy from them in real life, consider yourself lucky!"
  - Dangerous if applying DNNs to solving systems problems!
- Nevertheless, DNNs are useful for a variety of data processing applications
- Applying DNNs to systems problems needs a rethink

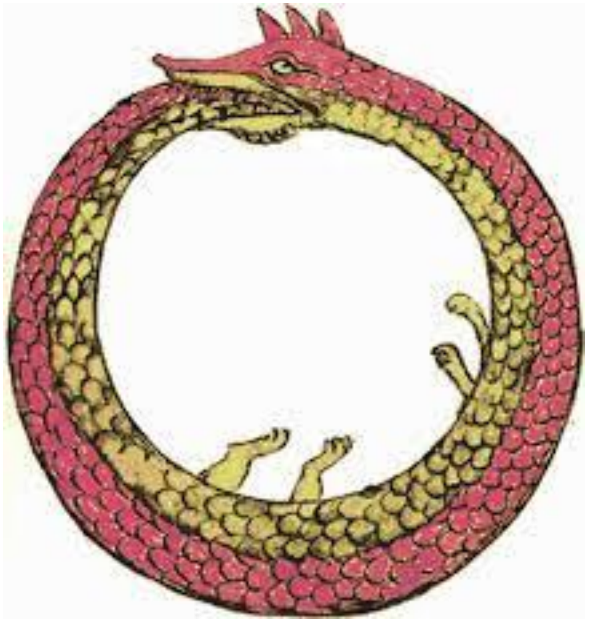
Two "Simple" DNN Models



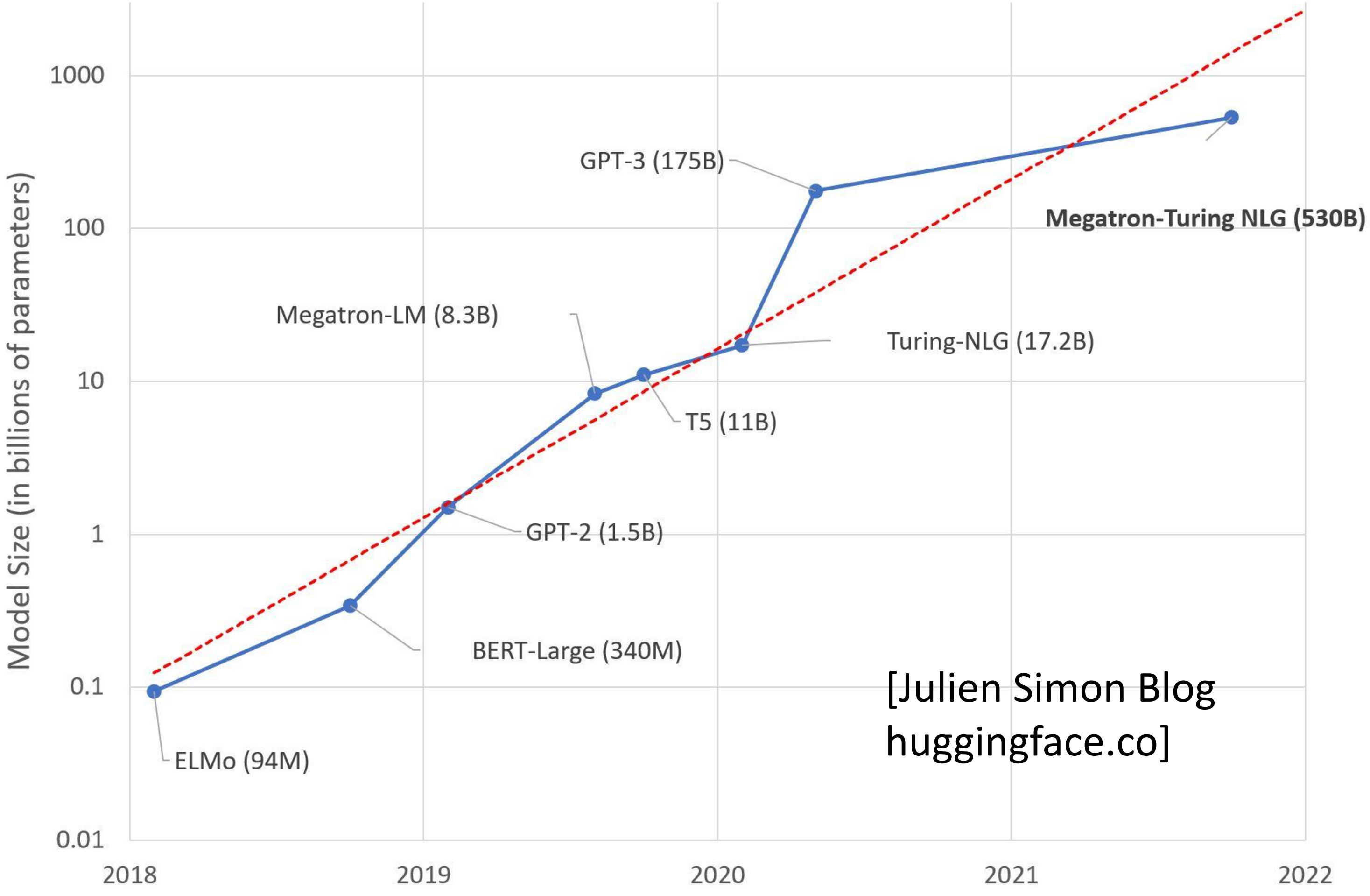


Case Study

- Our project on the problem of "**Model Parallelism of DNNs**"
- **Key Problem: Model Parallelism == Split a DNN model graph (for training) across multiple devices (GPUs) to satisfy memory constraints**

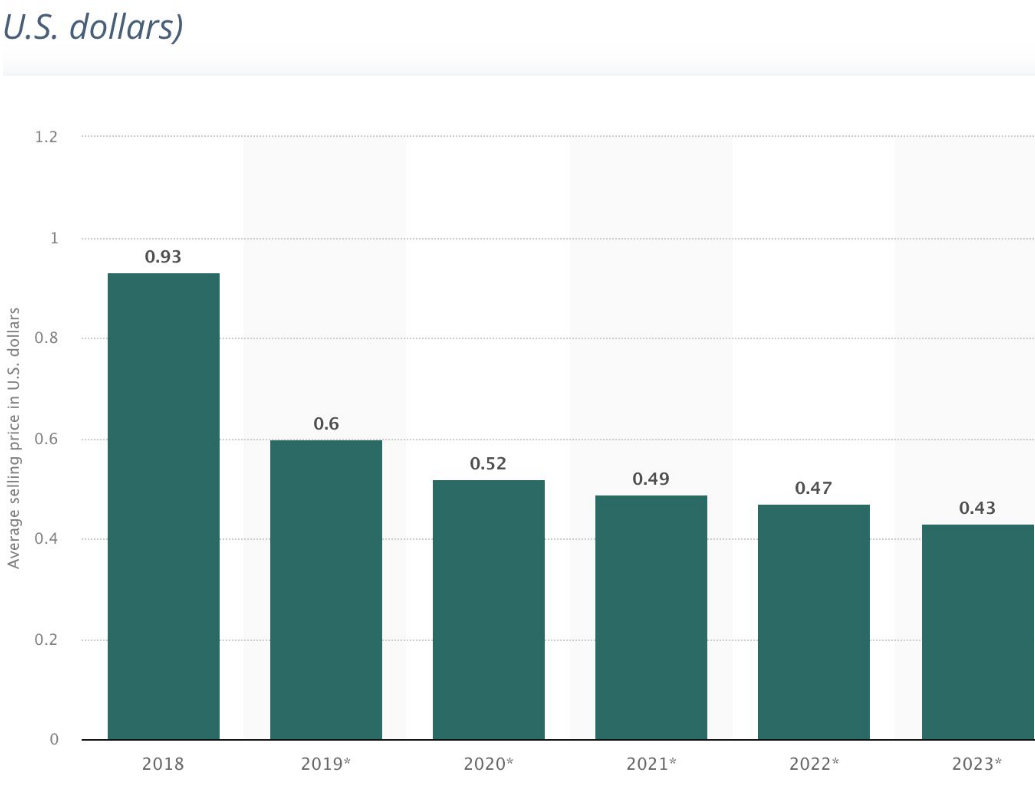


Why? ML Model sizes outpacing memory



[Julien Simon Blog
huggingface.co]

Average selling price of 1Gb equivalent DRAM units



[statista.com]

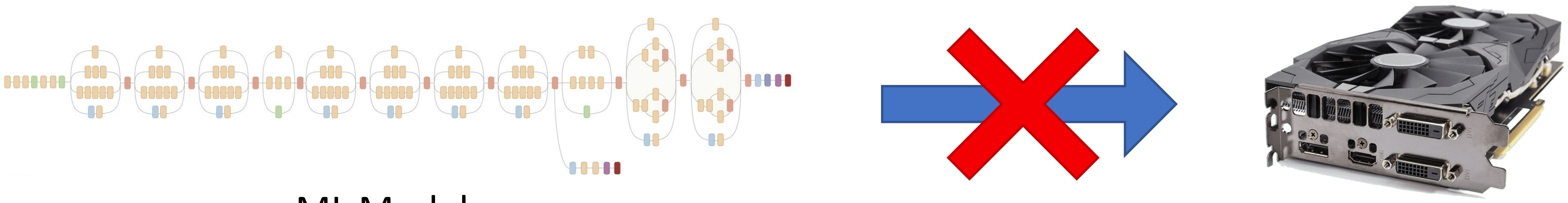
Why Model Parallelism?

Key Problem: Model Parallelism == Split a model graph (for training) across devices (GPUs) to satisfy memory constraints

- Even 32GB GPU **insufficient** for > 1.3 B parameters

GPU	P4	M60	K80	P100	T4	AWS Graviton	V100	A10G	A100
Memory	8 GB	8 GB	12 GB	12/16 GB	16 GB	16 GB	16/32 GB	24 GB	40 GB

- GPUs used in AWS, Google Cloud, and Azure



ML Model Graph

- ML training on *memory-constrained* devices

Approach 1/3: DIY (By hand)

- *Expert-designed* Approach
  - Developer does placement manually
  - E.g., Google Neural Machine Translation (GNMT), Inception-v3

☹️ High placement time: Require **domain knowledge** and **significant manual efforts**

☺️ Low step times (of placed model)

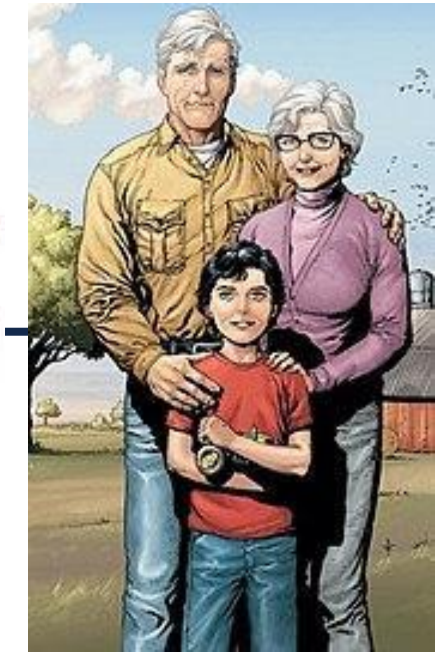
Approach 2/3: DNNs to the Rescue!...?

- ☺️ *Learning-based* Approaches to Placement
  - Use Reinforcement learning (RL)
  - ColocRL [ICML 2017] (Google)
  - HierarchicalRL [ICLR 2018] (Google)
  - Placeto [NeurIPS 2019] (MIT)
- ☺️ Low **step times** (of placed models) comparable to expert placements
- ☹️ Require very long **placement time** to place ML models (2 hours ~ 3 days)
  - Using Placeto on NMT models took 68.67 hours (2.86 days).
- ☹ Require **re-training** on different ML models and varying environment

Approach 3/3: Magic!

- Transformative!
- Radical!!
- Unthinkable in decades before!!!
- ☺️
- Our new approach: **Baechi** [SoCC 2020]
  - *("Baechi" = Korean word for placement)*

Baechi’s “radical” idea…
Old-fashioned Algorithms



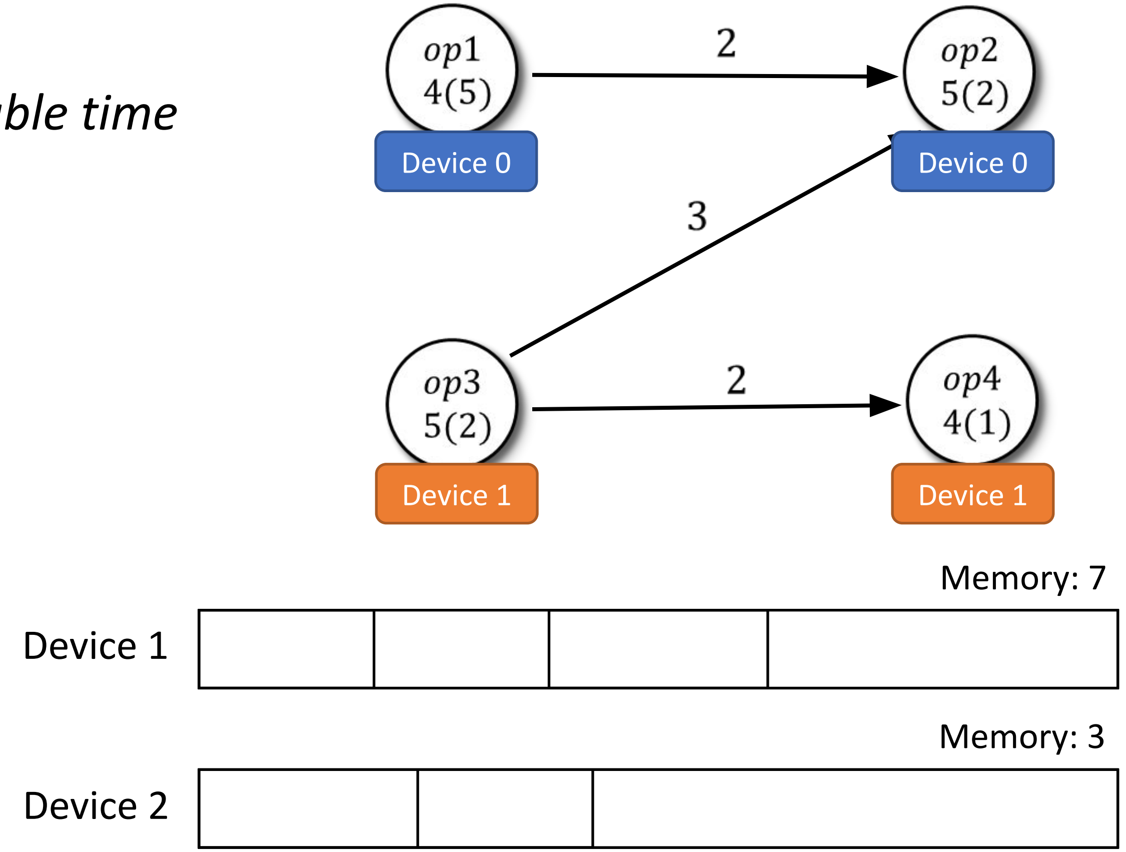
- DNN = Just a task precedence/dependency graph

- Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. **1989**. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. SIAM Jornal on Computing 18, 2 (1989), 244–257.
  - **ETF Placement Algorithm: Earliest Task First**
- Claire Hanen and Alix Munier. **1995**. An Approximation Algorithm for Scheduling Dependent Tasks on m Processors with Small Communication Delays. In 4th INRIA/IEEE Symposium on Emerging Technologies and Factory Automation (ETFA '95), Vol. 1. IEEE, 167–189.
  - **SCT Placement Algorithm: Small Communication Time**

Classical ETF

- Earliest Task First (ETF)
  - Schedule an operator with *earliest schedulable time* on its corresponding device *first*
  - Assumes *Infinite* memory
- In example: *op_number*

  compute time (memory)



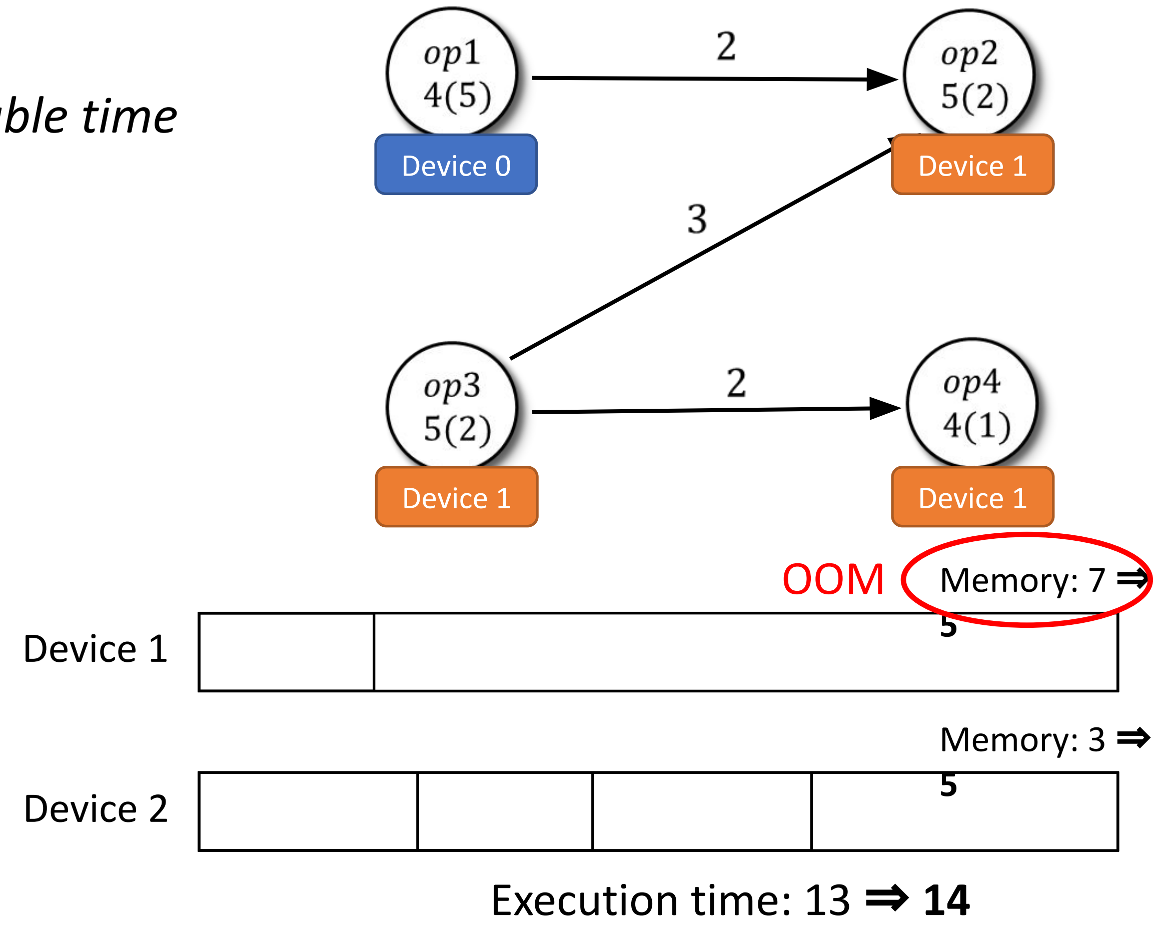
Execution time: 13

Hwang et al. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. SIAM Journal on Computing, 18(2)

1. m-ETF: Baechi's Memory constrained ETF

- Earliest Task First (ETF)
  - Schedule an operator with *earliest schedulable time* on its corresponding device *first*
  - Assumes *Infinite* memory

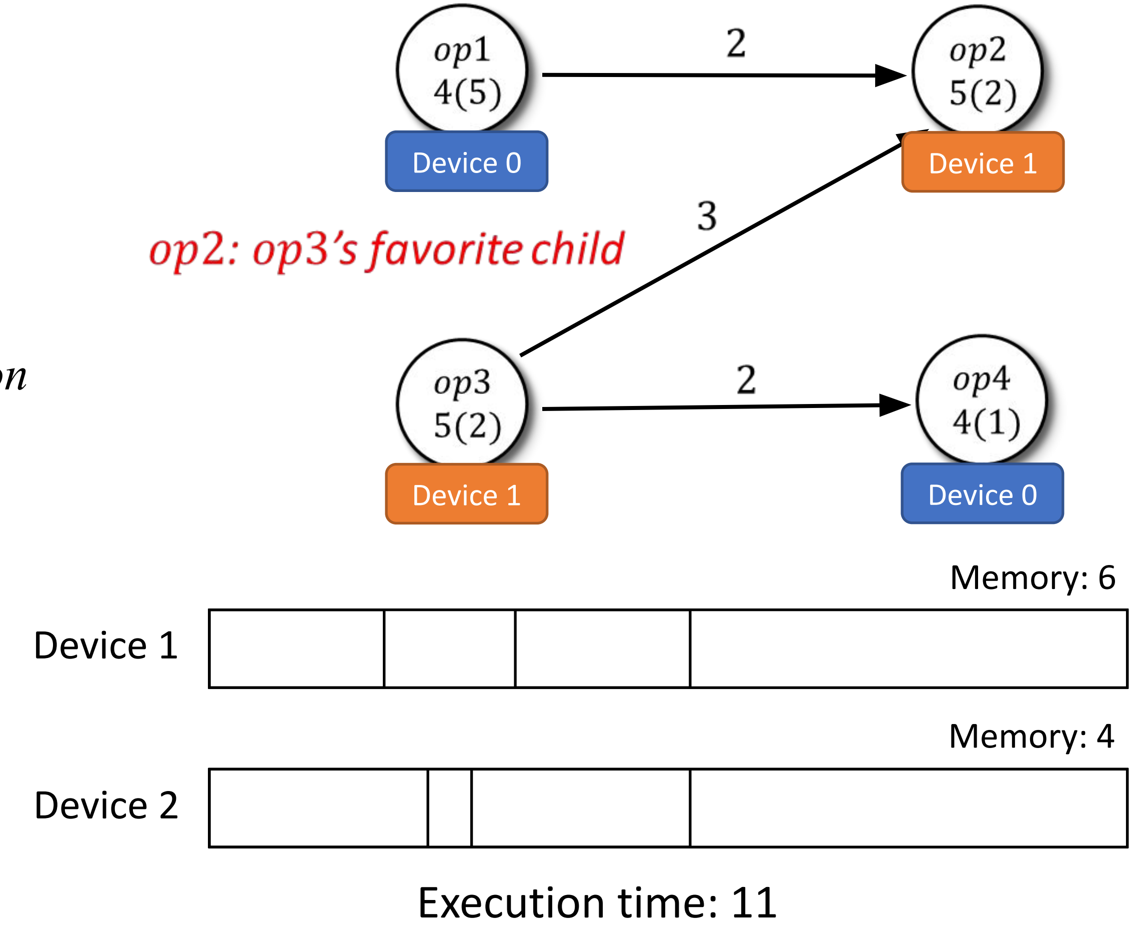
- **Our modified version:** ***m-ETF***
  - What if device memory limit is **5**?
  - *Exclude* devices with *insufficient* memory from placement



Classical SCT

- Small Communication Time (SCT)
  - Find operator's *favorite child* (via ILP) and schedule it on the *same* device as parent

Theorem (Old). *SCT's execution time has a* ***constant*** *approximation ratio with respect to the optimal execution time*.*



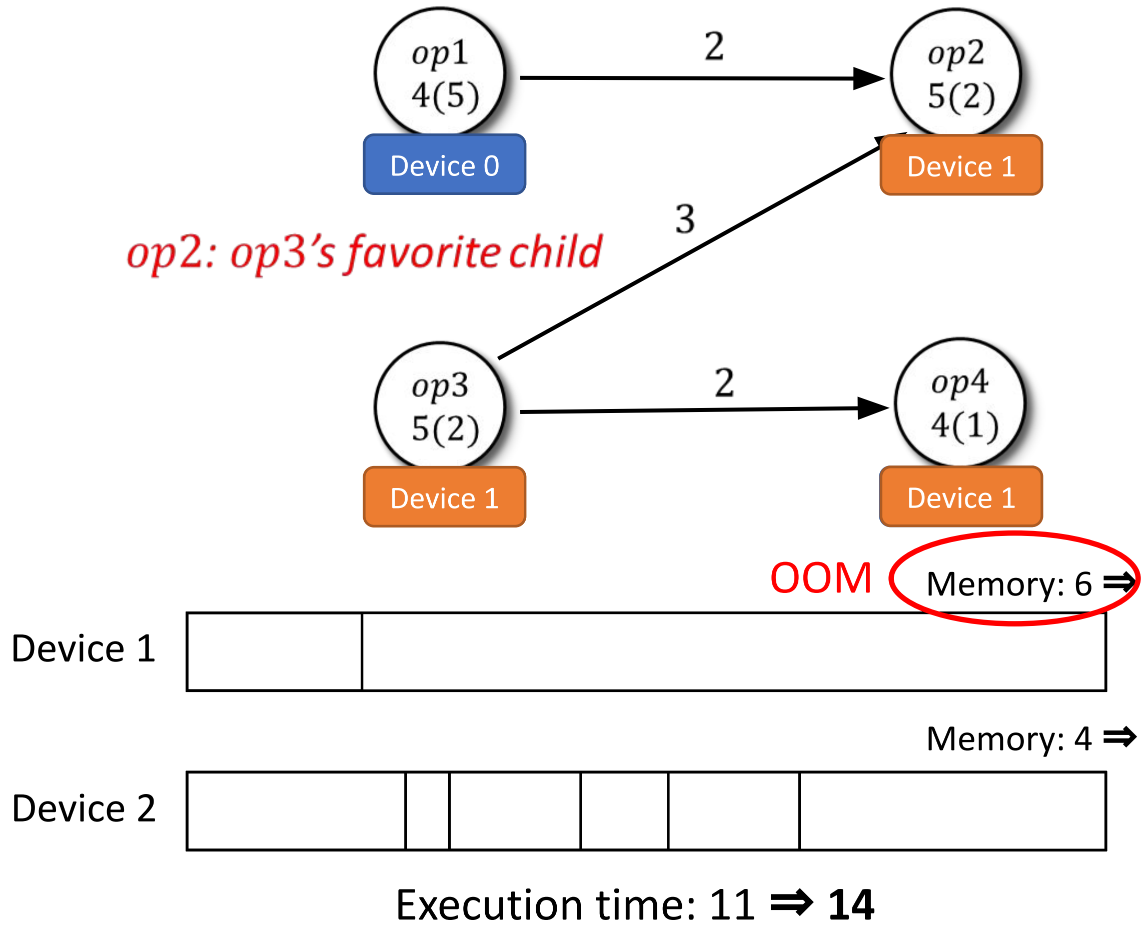
Hanen and Munier. An Approximation Algorithm for Scheduling Dependent Tasks on m Processors with Small Communication Delays. ETFA '95

* Conditions apply

2. m-SCT: Baechi's Memory constrained SCT

- Small Communication Time (SCT)
  - Find operator's *favorite child* (via ILP) and schedule it on the *same* device as parent
- **Our modified version:** ***m-SCT***
  - Determine favorite child via *relaxed ILP* (integer values in [0,1], and later round up/down). Solved by interior point method.
  - *Exclude* devices with *insufficient* memory from placement
  - Each device memory limit is **5**

Theorem (New). *m-SCT's execution time has a* ***constant approximation ratio*** *with respect to the optimal execution time*.*

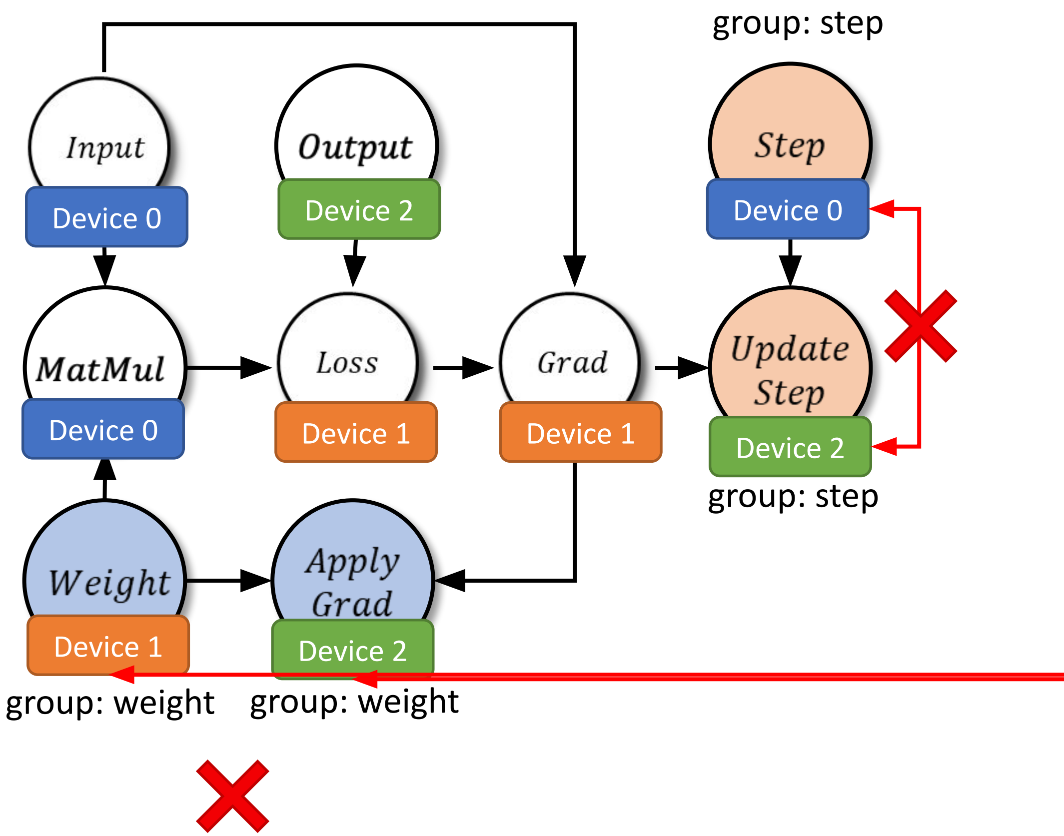


From an Algorithm to a System

- We can prove that these m-SCT(m-ETF) algorithms are within a constant factor of optimal.
- (Believe it or not, this was the easy part.)
- We implemented them into TensorFlow (1.12). Alas:
  - Generated placement results were **infeasible**
  - Performance was **awful**

Challenges #1: TensorFlow Colocation Constraints

- TensorFlow *requires* some operators to be *colocated*

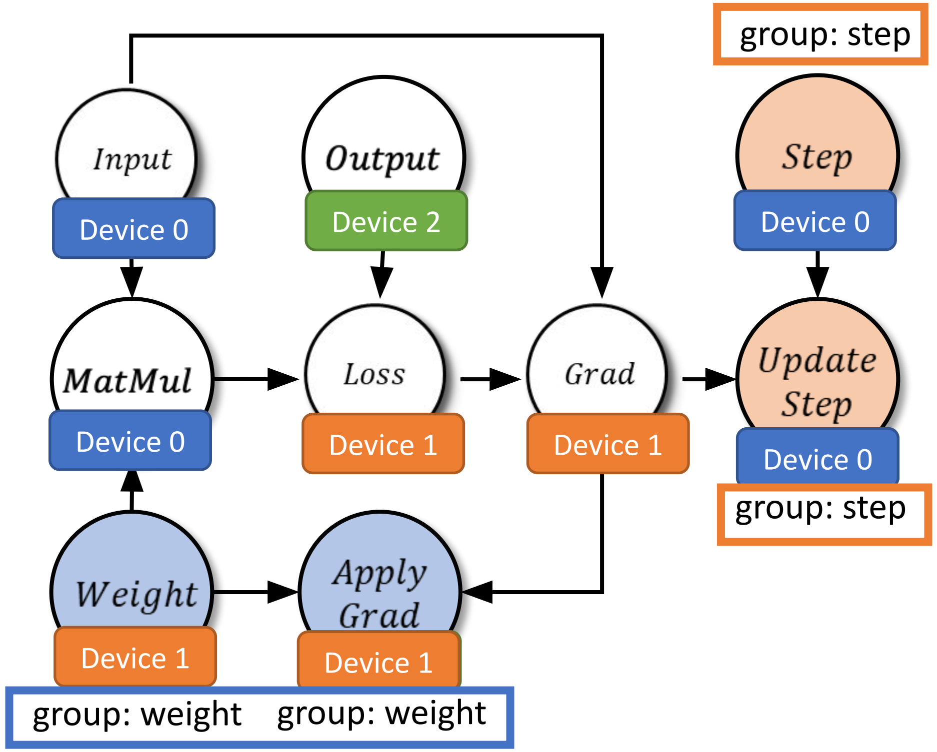


Challenges #1: TensorFlow Colocation Constraints

- TensorFlow *requires* some operators to be *colocated*

➪ Tried *post-adjust placement*
  - Fix *colocation-unaware* placement to satisfy the colocation constraints (tried 3 different ways)
  - **Inconsistent** performance gain

➪ *Co-adjust placement*
  - Consider colocations *while* creating schedule
  - 1^{st} operator in a group placed ⇒ other ops in the group placed on the same device

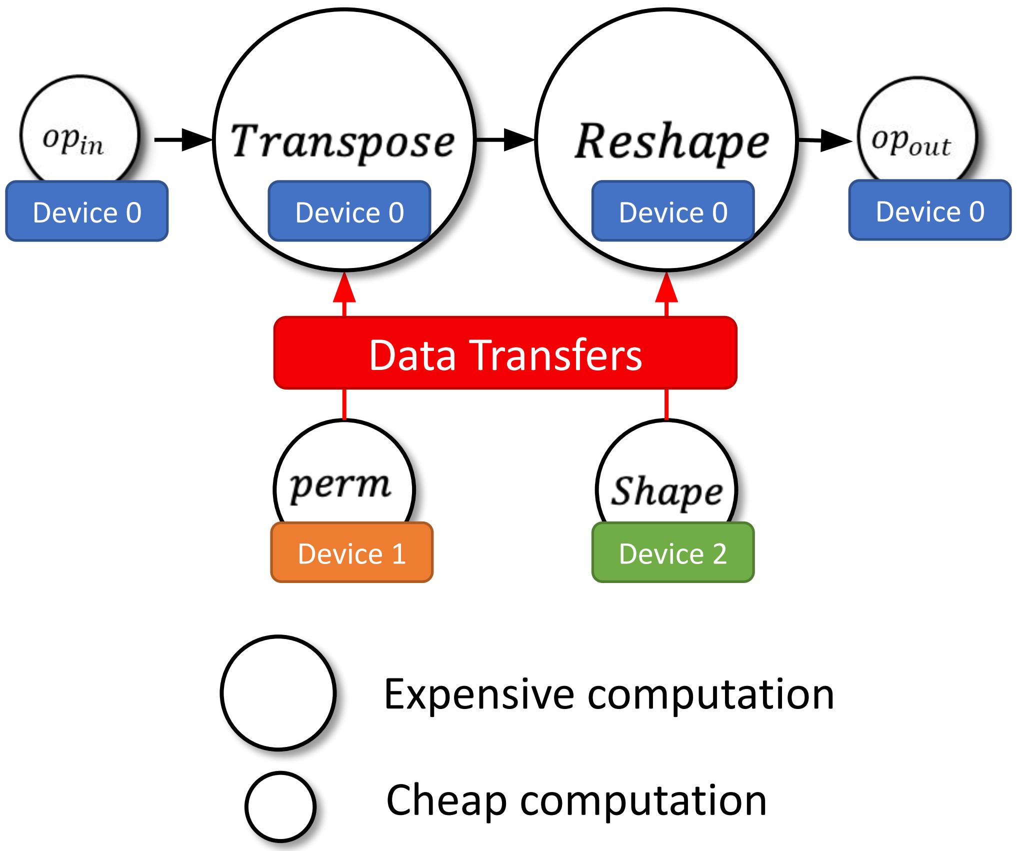


Challenge #2: Communication Blowup

- Splitting an ML model graph

  ⇒ Communication ↑

  ⇒ Step time ↑

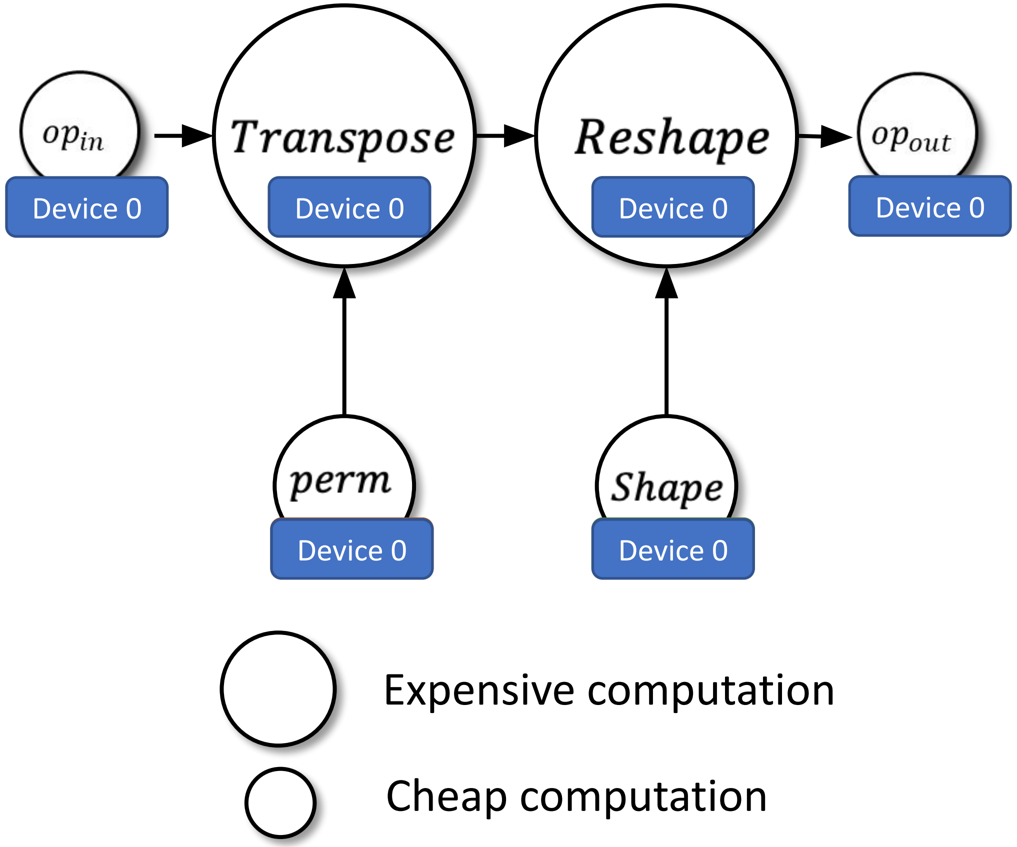


Challenge #2: Communication Blowup

- Splitting an ML model graph
  - ⇒ Communication ↑
  - ⇒ Step time ↑

➪ Operator *Co-placement*

- Operator's output is *only* used by its successor ⇒ Place them *together*
- Place respectively-matched forward and backward operators *together*

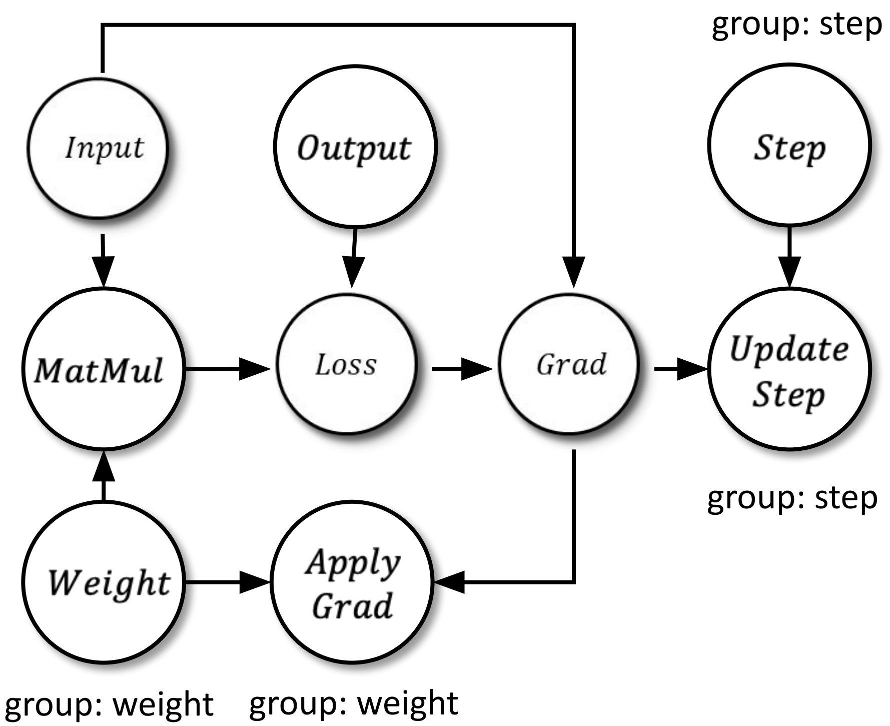


Challenge #3: Massive Number of Operators

- Number of operators ↑ ⟹ Placement time ↑
- E.g., 4-layer GNMT
  - 22,340 operators ⟹ 7-minute placement time

➪ *Operator Fusion*

- **Fuse** operators that are *directly connected* and *in the same co-placement group*

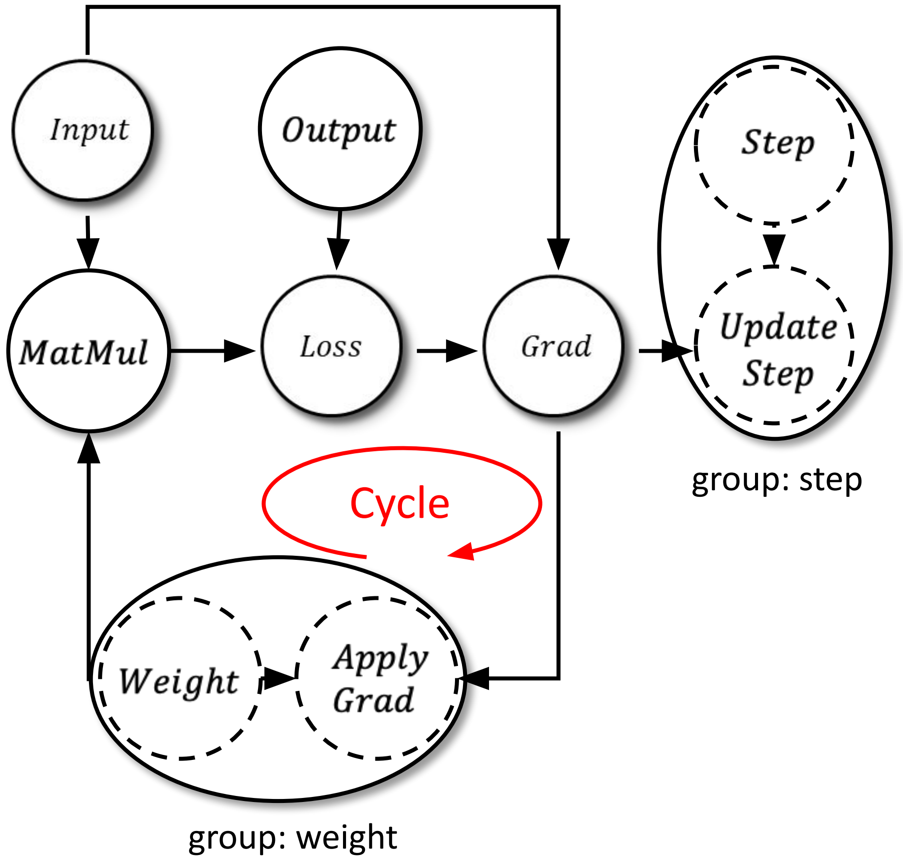


Challenge #3: Massive Number of Operators

- Number of operators ↑ ⟹ Placement time ↑
- E.g., 4-layer GNMT
  - 22,340 operators ⟹ 7-minute placement time

➪ *Operator Fusion*

- **Fuse** operators that are *directly connected* and *in the same co-placement group*
- May introduce cycles
  - Checking **all** cycles – Expensive, Not scalable
  - ***Conservative, local*** and so ***scalable*** heuristic

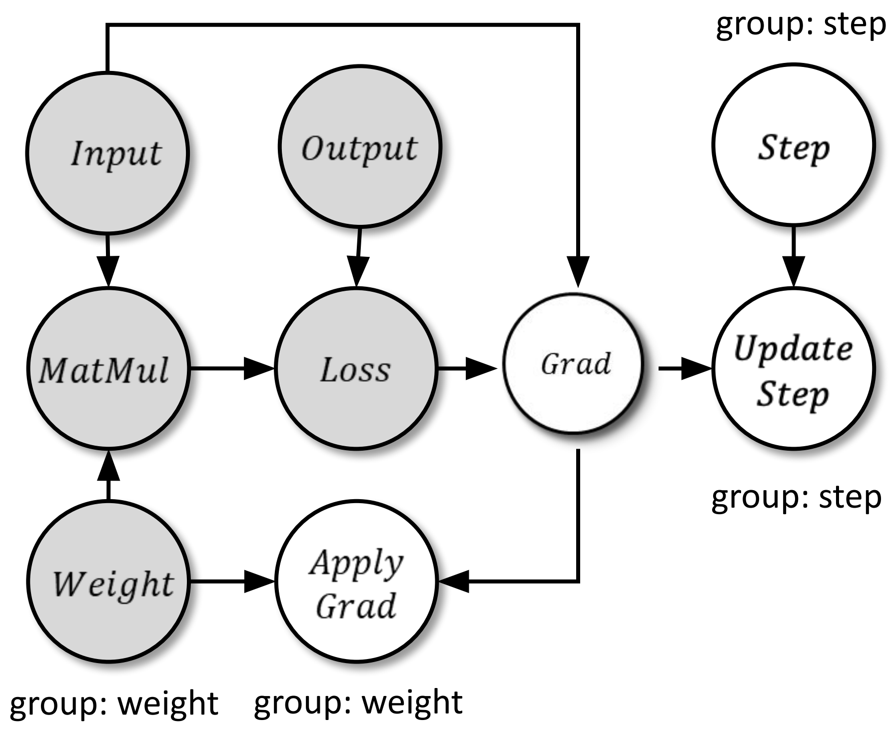


Challenge #3: Massive Number of Operators

➪ *Forward-Operator-based Placement*

- Place ops by *only* considering forward ops
  - Place backward ops as their corresponding forward ops on the same device

- 4-layer GNMT
  - # operators: 22,340 $\Longrightarrow$ **706**
  - Placement time: 7 minutes $\Longrightarrow$ **1.2 seconds**

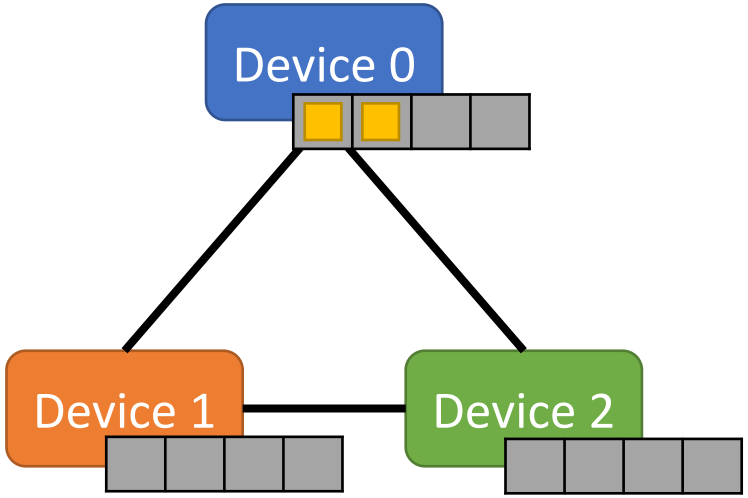


Challenge #4: Different Network Architecture

- m-SCT and m-ETF assume *parallel communication*

- Environment with a constrained network
  - Only *sequential communication* is supported
  - E.g., Indirect GPU-to-GPU communication

➪ *Sequential Communication Support*
  - Introduce device communication queues
  - Baechi planner automatically adds queuing time
  - Support computation-communication overlap
  - *Cache* received data to avoid duplicate transfers



Baechi-PT: Integrating Baechi into PyTorch

- That was for Baechi □ TensorFlow
- We also integrated Baechi □ PyTorch
- Challenges
  - PyTorch has *modules* (unlike TensorFlow's operators which are fine-grained)
  - PyTorch Developers need to specify communication programmatically

- Baechi-PT integration addresses this by
  1. **Co-placement** of subgraphs of modules that are common design patterns in the model
  2. **Annotating tensors** for backpropagation
  3. **An automated wrapper-based communication protocol** (leverages **CUDA streams** for both computation and communication)

How Long Does It Take to Place? (Placement Time)

- 4 NVIDIA RTX 2080 GPUs (8GB) with shared communication
- Baechi-TensorFlow

Model	HierarchicalRL [34]	Placeto [2]	Baechi (m-SCT)
Inception-V3	11 hrs 50 mins	1 hr 49 mins	1-10 seconds
NMT (GNMT)	1 day 21 hrs 14 mins	2 days 20 hrs 40 mins	1.2-48 seconds

Inception-V3:
654×–42.6K× Speedup over RL

GNMT:
3392×–206K× Speedup over RL

(Excludes profiling time, which was 10-12 s for the entire model)

How Fast Are Placed Models (Step Times)?

	Model	Batch Size	Single GPU	Expert	m-TOPO	m-ETF	m-SCT	Speedup over Single GPU: m-ETF	Speedup over Single GPU: m-SCT	Speedup over Expert (4 GPUs): m-ETF	Speedup over Expert (4 GPUs): m-SCT
TensorFlow	Inception-V3	32	0.269	0.269	0.286	0.269	0.269	0.00% (1 GPU Expert)			
		64	0.491	0.491	0.521	0.491	0.491	0.00% (1 GPU Expert)			
	GNMT (length: 40)	128	0.251	0.214	0.265	0.224	0.212	12.1%	18.4%	-4.5%	0.9%
		256	0.474	0.376	0.481	0.354	0.369	33.9%	28.5%	6.2%	1.9%
	GNMT (length: 50)	128	0.319	0.259	0.348	0.264	0.267	20.9%	19.5%	-1.9%	-3.0%
		256	0.618	0.484	0.609	0.502	0.516	23.1%	19.8%	-3.6%	-6.2%
PyTorch	Inception-V3	32	0.240	0.240	0.274	0.241	0.241	0.00% (1 GPU Expert)			
		64	0.461	0.461	0.537	0.465	0.462	0.00% (1 GPU Expert)			
	Transformer (length: 50)	64	0.249	0.257	0.262	0.242	0.244	2.9%	2.0%	6.2%	5.3%
		128	0.465	0.462	0.466	0.451	0.453	3.0%	2.6%	2.4%	2.0%

m-TOPO:
up to 34% higher than expert

m-ETF
-4.5% to **6.2%** speedup

m-SCT
-6.2% to **5.3%** speedup

How Effective are the Optimizations?

- m-SCT in Baechi-TensorFlow
- All optimizations applied

Model	Un-Optimized			Optimized		
	Num. Ops	Placement (seconds)	Step (seconds)	Num. Ops	Placement (seconds)	Step (seconds)
Inception-V3	6884	68.0	0.302	17	0.9	0.269
GNMT (length: 40)	18050	275.1	0.580	542	1.2	0.212
GNMT (length: 50)	22340	406.1	0.793	706	2.4	0.267

Number of Operators:
96.8%–99.8% Reduction

Placement times:
75.6×–229.3× Speedup

Step times:
1.1×–3.0× Speedup

Baechi-Parallel and -Inspired Takeaways

- Other Algorithmic Model Parallelism Approaches
  - *(concurrent with Baechi, though standalone)* Jakub Tarnawski, Amar Phanishayee, Nikhil Devanur, Divya Mahajan, and Fanny Nina Paravecino, “Efficient Algorithms for Device Placement of DNN Graph Operators,” NeurIPS 2020.
  - *(inspired by Baechi)* Ubaid Ullah Hafeez, Xiao Sun, Anshul Gandhi, and Zhenhua Liu. 2021, Towards Optimal Placement and Scheduling of DNN Operations with Pesto, Middleware 2021
  - Other related works that came afterward: Megatron-LM and Terapipe (for Transformers), Pipedream (Pipeline parallelism), Alpa & Unity (combining different parallelisms)

Lessons Learned: Using RL vs. Using Algorithms


- RL approaches need retraining when one changes the setup (scale), devices, or model
- Algorithmic approaches are more generalizable
- Designing good algorithms can be hard, but sometimes easier than one expects! (especially if one reads the literature)
- Adapting an algorithm into a system is non-trivial.
  - Baechi TensorFlow took 1-2 person years, led by one determined graduate student
  - Baechi PyTorch was tough (0.6 person years), faster because of our previous TF experience
- But the rewards are worth it!
- Creativity and Determination >> massive resources at FAANG companies

Lessons Learned (2): Hammer vs. Gavel

- **Not indiscriminate**: Using the RL/DNN Hammer for systems problems should be **selective**
- **Parallel Design**: DNN development and Algorithm development should occur in parallel, rather than “either or”
- **Concurrent at Run-time**: Many scenarios where algo can give you a quick solution, and DNN can help customize it (or vice versa!)
- **Papers using DNN** must explicitly explain
  - Alternative algorithmic designs explored
  - Why those didn’t work
  - Why ML is a good match for this problem
  - Compare DNN to best-known algorithms (or heuristics)

- *“Simplicity is the ultimate sophistication.” – Lao Tzu*