

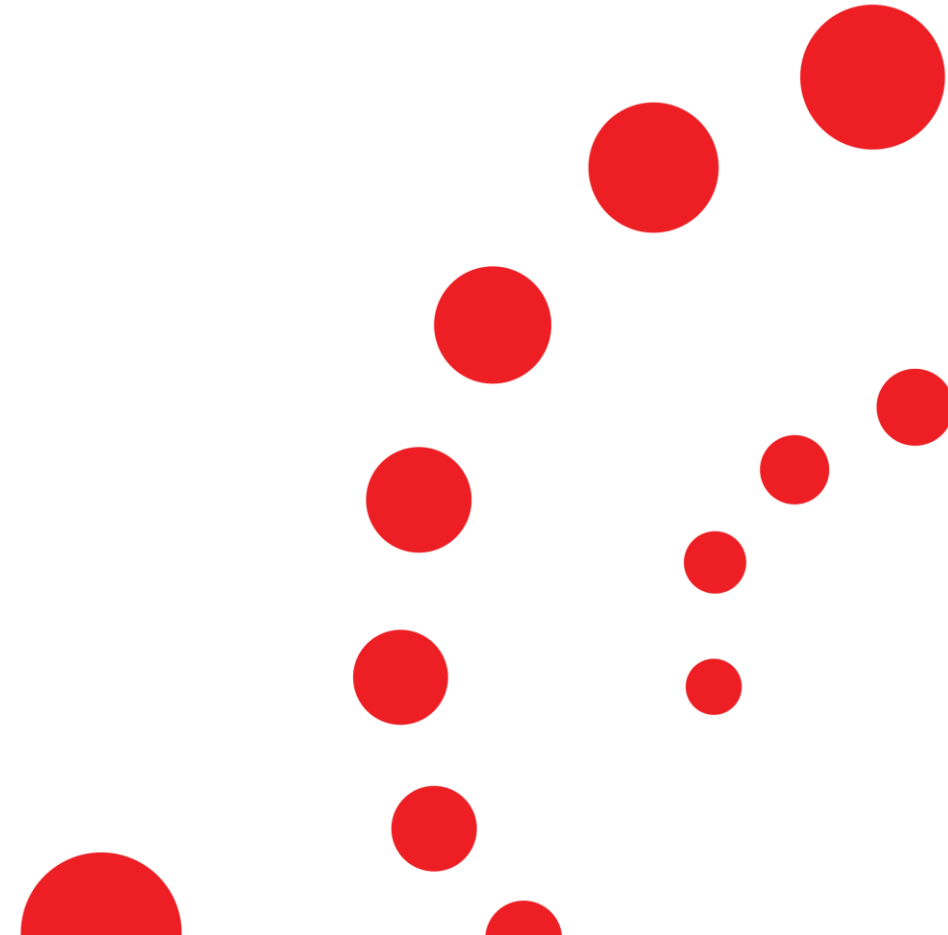


# MLSys 2022 Trip Report

9/30/2022

OpenMindSpore Project

Silicon Valley System Software Lab



# Contents

- Overview
- Outstanding Paper Awards
- Selected papers of interests
- Comments

# 5<sup>th</sup> Conference on Machine Learning and Systems (MLSys)

- Date: August 29 to September 1, 2022
- Location: Santa Clara, CA, USA
- In-person only, no hybrid/virtual attendance supported this year
- 247 papers submitted, 51 paper accepted (20.6% acceptance rate)

# MLSys 2022 Sponsors

- Platinum
  - TikTok, Meta
- Gold
  - Netflix, Microsoft, Moloco, Mosaic ML
  - Rebuy, Tesla, Snowflake, Bloomberg
  - Amazon Science
- Silver
  - ARM, Hewlett Packard Enterprise, Relational AI, Alibaba Cloud
  - Qualcomm

# Outstanding Paper Awards

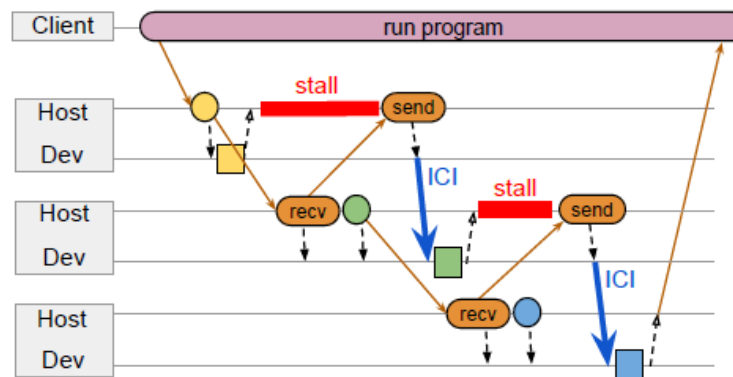
- Pathways: Asynchronous Distributed Dataflow for ML (Google)
- QuadraLib: A Performant Quadratic Neural Network Library for Architecture Optimization and Design Exploration
- Random Offset Block Embedding (ROBE) for Compressed Embedding Tables in Deep Learning Recommendation Systems
- ML-EXray: Visibility into ML Deployment on the Edge
- GPU Semiring Primitives for Sparse Neighborhood Methods (Univ of Maryland and Nvidia)

# MLSys Presentations of Interests

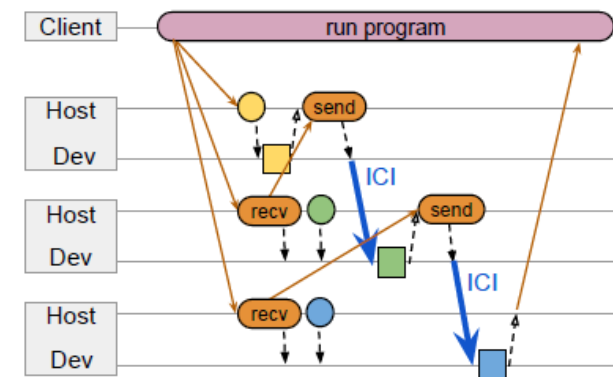
- Pathways: Asynchronous Distributed Dataflow for ML
- BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling
- The CoRa Tensor Compiler: Compilation for Ragged Tensors with Minimal Padding
- Apollo: Automatic Partition-based Operator Fusion through Layer by Layer Optimization
- DietCode: Automatic Optimization for Dynamic Tensor Programs
- Towards the Co-Design of Neural Networks and Accelerators
- Efficient Strong Scaling Through Burst Parallel Training
- Synthesizing Optimal Parallelism Placement and Reduction Strategies on Hierarchical Systems for Deep Learning
- dPRO: A Generic Performance Diagnosis and Optimization Toolkit for Expediting Distributed DNN Training

# Pathways: Asynchronous Distributed Dataflow for ML (Google)

- A new large scale orchestration layer for accelerators
  - Enabling exploration of new systems and ML research ideas (e.g., different model weights can be updated per example, or even per sub-example)
  - Addressing the restriction of current SPMD programming model, providing state-of-art performance
  - Using shared dataflow graph of **asynchronous** operators, and **gang-scheduling heterogeneous** parallel computations on accelerators
  - Overcoming JAX limitations that cannot scale beyond a single TPU pod, enabling communication among thousands of TPU cores over Inter-Core Interconnect (ICI) and DC Network (DCN)
- Pathways executors and schedulers could be replaced by Ray actors (*using PyTorch backend*) and cluster scheduling, respectively.



(a) Sequential dispatch



(b) Parallel dispatch

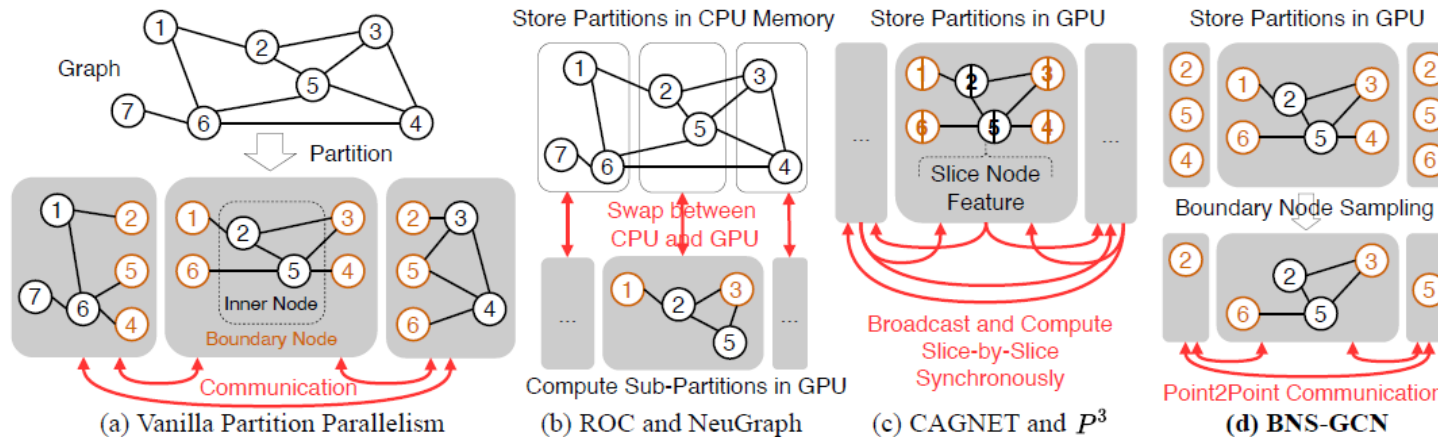
# MindSpore Perspective

- Pathways mainly tries to overcome JAX's limitation of scaling beyond a single TPU pod (due to collective communication over ICI).
- Current MindSpore using Ascend accelerators shares the same issue of restricted MPI programming model and exclusive resource utilization of accelerators.
- A distributed execution engine like Ray could be adapted so that MindSpore could provide more flexible scheduling of heterogeneous workers for sub-graph execution, resulting in easy expression of new parallelism patterns and comparable performance to SPMD case.



# BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling (Rice Univ & Univ of Illinois)

- Training large GCNs is challenging due to excessive number of boundary nodes of each partitioned subgraph, causing:
  - Heavy communication overhead
  - Prohibitive memory requirement
  - Imbalanced memory consumption
- BNS-GCN techniques
  - METIS-based **graph partition** achieves balanced computation time, avoiding stragglers blocking other partitions to proceed in synchronous training
  - Random boundary node sampling** stores and communicates selected nodes only. Random
- BNS shows lower feature approximation variance compared to state-of-art sampling methods (GraphSAGE, VR-GCN, FastGCN, LADIES)



**Algorithm 1:** Boundary node sampling for partition-parallel training (per-partition view)

**Input:** partition number  $m$ , partition id  $i$ , graph partition  $\mathcal{G}_i$ , boundary node set  $\mathcal{B}_i$ , node feature  $X_i$ , label  $Y_i$ , sampling rate  $p$ , initial model  $w[0]$ , learning rate  $\eta$

**Output:** trained model  $w[T]$  after  $T$  iterations

```

1  $\mathcal{V}_i \leftarrow \{\text{node } v \in \mathcal{G}_i : v \notin \mathcal{B}_i\};$   $\triangleright$  create inner node set
2  $H^{(0)} \leftarrow X_i;$   $\triangleright$  initialize input features
3 for  $t \leftarrow 1 : T$  do
4    $\mathcal{U}_i \leftarrow$  randomly pick elements in  $\mathcal{B}_i$  with probability  $p$ ;
5    $\mathcal{H}_i \leftarrow$  node induced subgraph of  $\mathcal{G}_i$  from  $\mathcal{V}_i \cup \mathcal{U}_i$ ;
6   Broadcast  $\mathcal{U}_i$  and Receive  $[\mathcal{U}_1, \dots, \mathcal{U}_m]$ ;
7    $[\mathcal{S}_{i,1}, \dots, \mathcal{S}_{i,m}] \leftarrow [\mathcal{U}_1 \cap \mathcal{V}_i, \dots, \mathcal{U}_m \cap \mathcal{V}_i]$ ;
8   for  $\ell \leftarrow 1 : L$  do
9     Send  $[H_{\mathcal{S}_{i,1}}^{(\ell-1)}, \dots, H_{\mathcal{S}_{i,m}}^{(\ell-1)}]$  to partition  $[1, \dots, m]$ 
       and Receive  $H_{\mathcal{U}_i}^{(\ell-1)}$ ;
10     $H^{(\ell)} \leftarrow GCN^{(\ell)}\left(\mathcal{H}_i, \begin{bmatrix} H^{(\ell-1)} \\ H_{\mathcal{U}_i}^{(\ell-1)} \end{bmatrix}, w[t-1]\right);$ 
11  end
12   $f_i \leftarrow \sum_{v \in \mathcal{V}_i} \text{loss}(h_v^{(L)}, y_v);$   $\triangleright$  calculate loss
13   $g_i[t] \leftarrow \frac{\partial f_i}{\partial w[t-1]};$   $\triangleright$  backward pass
14   $g[t] \leftarrow \text{AllReduce}(g_i[t]);$   $\triangleright$  share gradients
15   $w[t] \leftarrow w[t-1] - \eta \cdot g[t];$   $\triangleright$  update model
16 end
17 return  $w[T]$ 
    
```

# The CoRa Tensor Compiler: Compilation for Ragged Tensors with Minimal Padding (CMU & OctoML)

- Current deep learning frameworks use padding and masking to make data shapes uniform, and offloads computations to optimized kernels for dense tensor algebra.
- Such techniques cause waste computation and loss in performance.
- CoRa (Compiler or Ragged Tensors) proposes new scheduling primitives for **ragged tensors**, while providing generalized APIs.
  - Loop scheduling, operation splitting, horizontal fusion, loop and storage padding, tensor dimension scheduling, load balancing
- CoRa is implemented on TVM and shows comparable performance as highly hand-optimized code.
- Ragged tensors are usually much **denser** compared to sparse tensors and the applications are quite different.
- Recently DL frameworks have begun to support ragged tensors.
  - RaggedTensor (TensorFlow), NestedTensor (PyTorch)

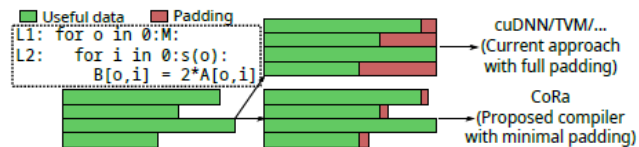


Figure 1: An example operation on ragged tensors.

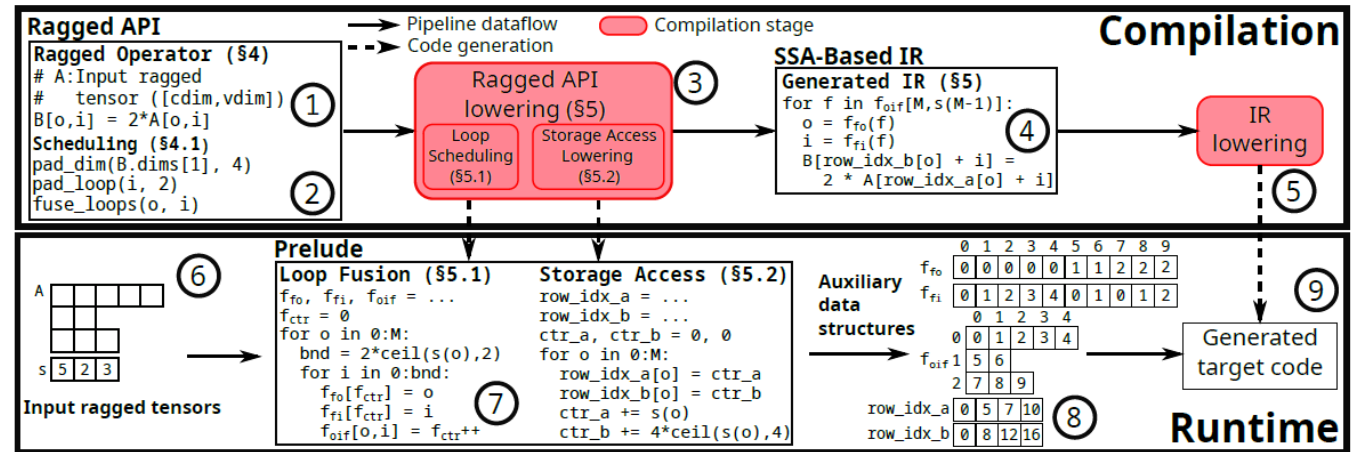


Figure 4: Overview of CoRa's compilation and runtime pipeline.

# MindSpore Perspective

- MindSpore has not supported ragged tensors yet.
- Similar to TensorFlow and PyTorch, MindSpore could introduce a new tensor type for ragged tensors, and its operators.
- Automated generate code could be highly portable across different hardware platforms and achieve comparable performance as hand-optimized code.

# Apollo: Automatic Partition-based Operator Fusion through Layer by Layer Optimization (State Key Lab & Huawei)

- Problems of existing operator fusion
  - Missing the opportunities to fuse with compute-bound ops and exploiting fusion within an incomplete space (TensorFlow, DLVM, Glow)
  - Scalability of the loop fusion heuristics (TVM)
  - Falling short in supporting custom ops in training scenarios (TASO, Rambler)

- Apollo approach

- Extending search space of fusion by considering more op types
- Allowing reverse feedback from operator-level optimizer for scalability
- Modeling both data locality and parallelism
- Reasonable JIT compilation overhead

- Apollo architecture

- Partition phase** (rule-based)
  - extracts maximum set of sub-graphs
  - splits them into individual subgraphs
- Fusion phase**
  - Layer I: carries out loop fusion for each sub-graph (polyhedral loop fusion)
  - Layer II: implements node grouping (by memory stitching)
  - Layer III: exploits parallelism between independent ops (by parallelism stitching)

- MindSpore integration

- <https://gitee.com/mindspore/mindspore>
- Apollo enabled by the parameter 'enable\_graph\_kernel'

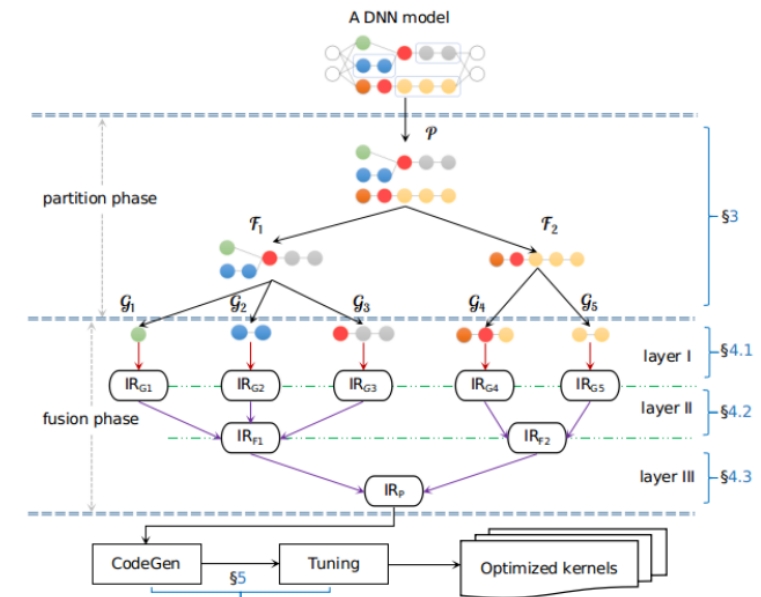


Figure 2: Architecture of APOLLO.

# DietCode: Automatic Optimization for Dynamic Tensor Programs (Univ of Toronto & AWS)

- **Dynamic tensor**
  - All shapes are unknown at compile-time
  - Examples: neural architecture search, dynamic by design (models in sequence learning), varying shapes depending on layer position in model
  - Existing auto-scheduler frameworks can only support static-shape workloads.
- DietCode
  - Joint learning approach that optimizes all the possible shapes of workload collectively within the same **shape-generic search space** (by applying micro-kernels) and collective learning the same cost model
  - $O(1)$  complexity of auto tuning (compared to  $O(|S|)$  where  $|S|$  is # of possible shapes)
  - Implemented on top of TVM

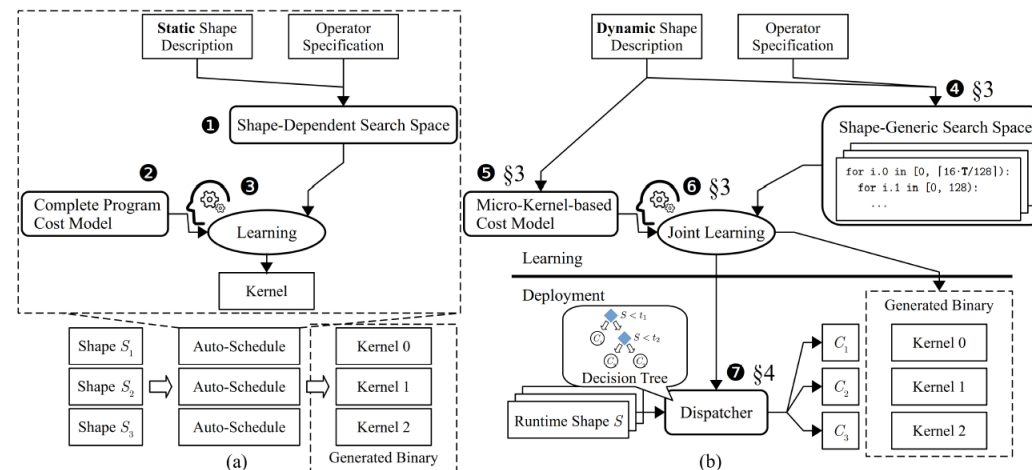


Figure 1. Code-generation comparison between (a) existing auto-schedulers and (b) DietCode. Existing approaches auto-schedule each shape individually. DietCode solves the problem by having all shapes jointly search within the same shape-generic search space and update the same micro-kernel cost model.

# MindSpore

- Dynamic tensor has not been considered in MindSpore yet.
- MindSpore could adapt the same idea of DietCode by adapting shape-generic search space in auto tuning and the cost calculation, similar to how DietCode did with TVM.

# Towards the Co-Design of Neural Networks and Accelerators (Google)

- **NaaS** (Neural Architecture and Accelerator Search) jointly searches the best configuration of neural architecture and accelerator.
  - NaaS parameterize neural architecture search and hardware architecture search in a unified joint search space.
  - NaaS is task-driven, not a set of fixed programs or graphs for generalization across vertical tasks.
  - NaaS is evaluated on a cycle-accurate accelerator simulator.
- NaaS outperforms **platform-aware NAS** (Neural Architecture Search) or hardware-aware design space exploration, which optimizes one end of space.
  - Optimizing indirect metrics (e.g., parameter counts or FLOPS) won't necessarily improve direct metrics of latency.
- NaaS optimization uses PyGlove, a symbolic programming library using a RL controller (PPO algorithm, Proximity Policy Optimization).

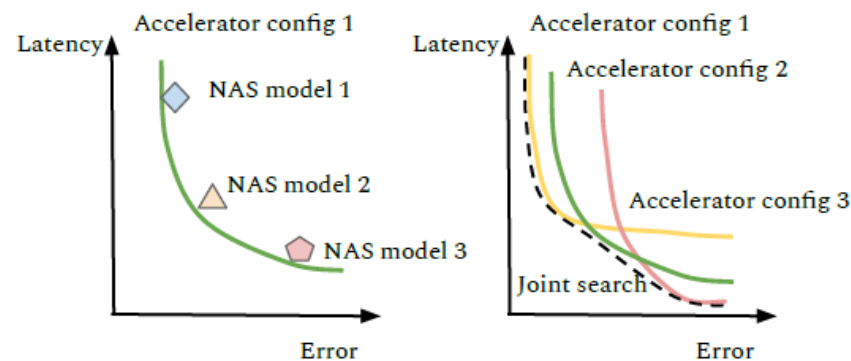


Figure 3. NaaS expands the pareto frontiers.

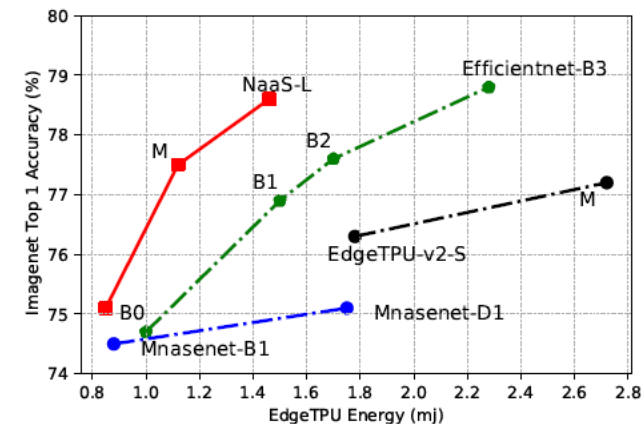


Figure 1. ImageNet Top 1 Accuracy vs. Energy (mj) comparing NaaS and related work.



# Efficient Strong Scaling Through Burst Parallel Training (MIT)

- Challenges for scaling DNN training efficiency
  - Training is less effective beyond a certain point, because larger batch sizes causes a loss in sample efficiency.
  - The uneven parallelism in some layers of many DNN models causes underutilization of GPUs.
  - Conventional data parallel approach is insufficient for GPU hardware utilization in this scenario.
- DeepPool** introduces **burst parallel training** to dynamically adjust # of GPUs to allocate to each layer.
  - This improves overall cluster efficiency by reclaiming underutilized GPUs for other training tasks.
- DeepPool's **GPU multiplexing** allows to train multiple models on each GPU simultaneously.
  - QoS-aware admission control in runtime low batch sizes for low priority tasks

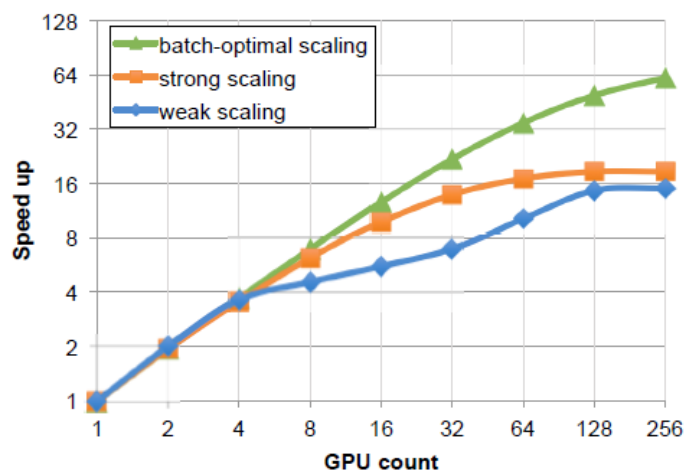


Figure 1. The estimated speedups for training VGG-11 to error =

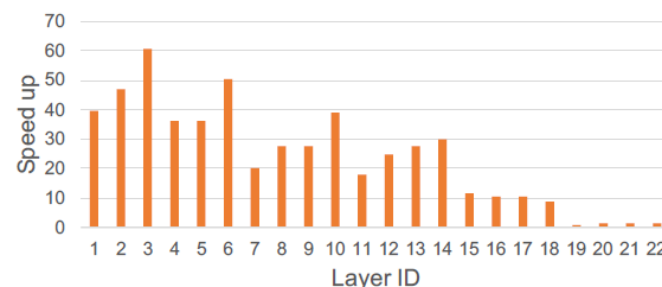


Figure 5. Heterogeneous scalability of layers in VGG16. Y-axis

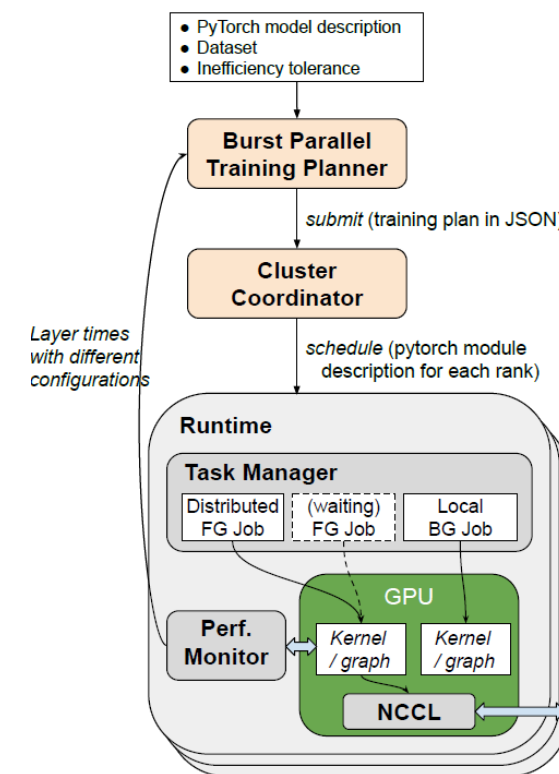


Figure 6. System architecture.



# MindSpore Perspective

- DeepPool's burst parallelism, uneven resource allocation of GPU to different layers of DNN models is similar to PipeTransformer's idea on different speed of training phase on different layers.
- PipeTransformer mostly considers DNN training for a single model, whereas DeepPool allows to run multiple DNN training tasks using the same GPU hardware pools.

# Synthesizing Optimal Parallelism Placement and Reduction Strategies on Hierarchical Systems for Deep Learning (Univ of Cambridge & DeepMind)

- P<sup>2</sup>: a tool for **parallelism placement** and placement-aware **synthesis of reduction strategies**
  - A novel mapping of data & model parallelism to hierarchical accelerators (e.g., rack, server, CPU, GPU)
  - Formal semantics for collectives based on Hoare triples
  - Domain-specific language expressing simultaneous reductions among device group (Nvidia GPUs forming a ring via NVLink), using **syntax-guided program synthesis**
  - Lowering programs into sequences of NCCL calls on XLA GPU backend
  - P<sup>2</sup> outperforms default all-reduce implementation on different GPU hierarchies.

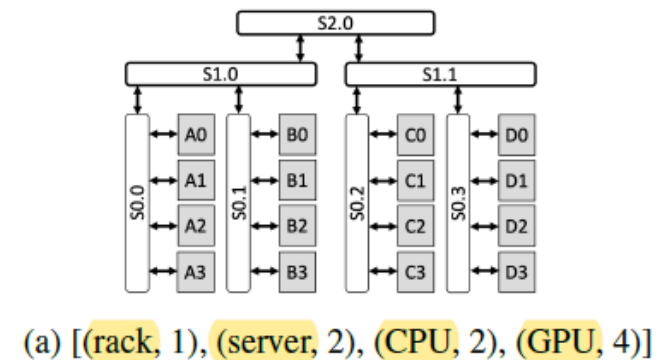
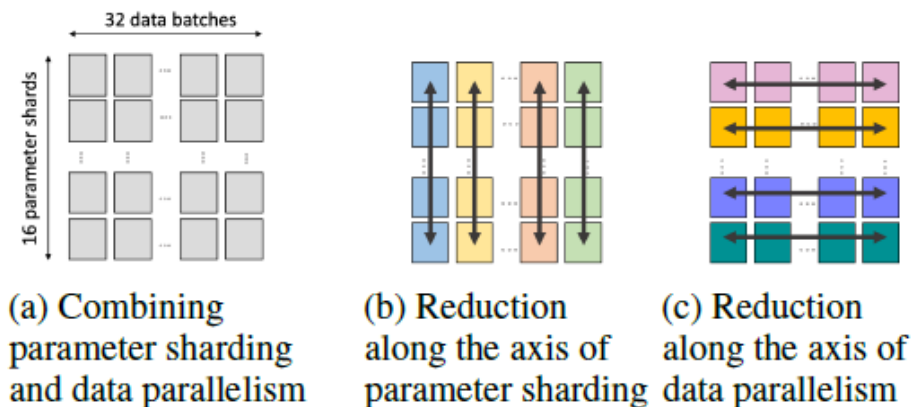


Figure 1: Parallelism combination

# dPRO: A Generic Performance Diagnosis and Optimization Toolkit for Expediting Distributed DNN Training (Univ of Hong Kong & ByteDance)

- Efficient **profiler**: collecting runtime traces of computations and communications
  - Existing systems are error-prone due to lack of global timestamp support and exact time of receiving data
  - Global DFG (Data Flow Graph) constructed from local DFGs and fine-grained communication topology
- **Optimizer**: identifying performance bottlenecks and exploring optimization strategies
- Search space speed-up by:
  - Coarsened view of global DFG (grouping tensor-producing and non-tensor-producing ops together)
  - Partial replay
  - Exploiting symmetry of DNN models (e.g., multiple transformers in BERT)
- **Replayer**: simulating execution of global DFG based on modified Kahn's algorithm
- Profiler added to TensorFlow, MXNet, communication (to Horovod for NCCL, BytePS for parameter-server-based training)
- dPRO achieves 3.48x speed-up over baseline for training time while maintaining < 5% errors in most cases.
- Source code available at <https://github.com/joapolarbear/dpro>

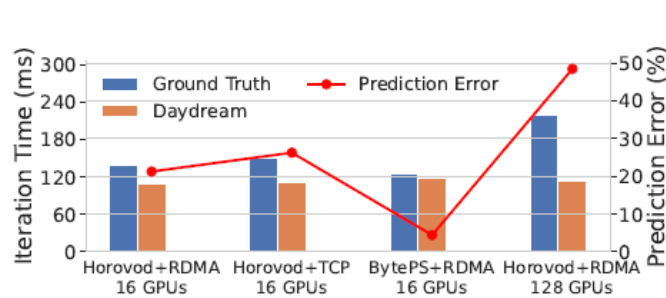


Figure 1. Training ResNet50 in 100Gbps network, batch size 32 per GPU (see Sec. 7.1 for testbed details)

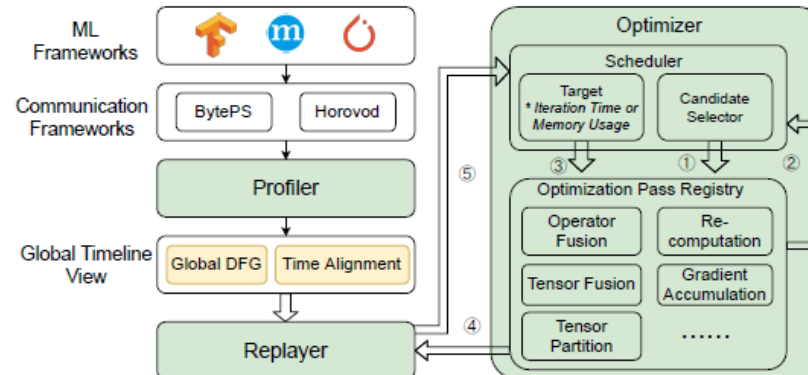


Figure 3. dPRO architecture.

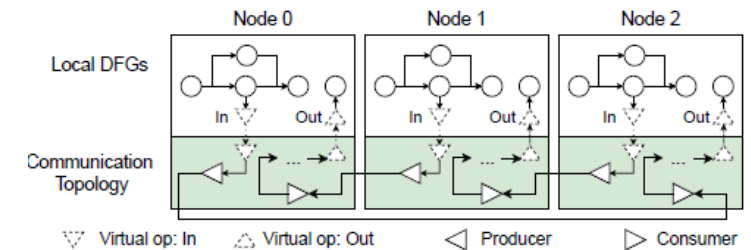


Figure 4. An illustration of the global DFG.

# Comments

- MLSys is a new conference specialized for ML systems.
- There were 3 invited talks
  - Systems for ML and ML for systems: a virtuous cycle (by Kunle Olukotun from Stanford)
  - Towards building a responsible data economy (by Dawn Song from Univ of Cal, Berkeley)
  - Accelerating engineering with machine learning (by Ryan Adams from Princeton Univ)
- 4 Workshops were co-held.
  - [Practical adaption challenges of ML for systems in industry](#)
  - [Benchmarking machine learning workloads on emerging hardware](#)
  - [Cloud intelligence: AI/ML for efficient and manageable cloud services](#)
  - [Cross-community federated learning: algorithms, systems, and co-designs](#)
- Chips & compilers symposium was held last day.
  - <https://chips-compilers-mlsys-22.github.io/>
  - Topics covered: carbon footprint of ML training, autotuning of ML, PyTorch compilers, ML for small devices, ML optimization for cloud service, efficient code generation for DNNs