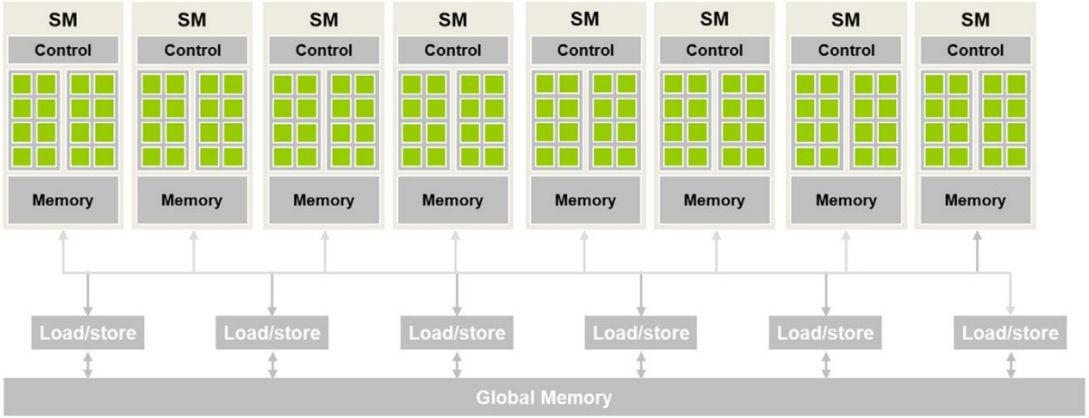
Forecasting GPU Performance for Deep Learning Training and Inference

Seonho Lee, Divya Mahajan Georgia Institute of Technology, Amar Phanishayee Meta

Presenter
Dipak Acharya
University of North Texas
Denton TX

GPU Architecture



- Performance prediction is essential to facilitate the AI model training in the rapid development and advancement in models as well as GPU hardware.
- GPUs consists of multiple identical blocks known as Streaming Multiprocessors(SMs)
- Each SM has its on L2 cache which is connected to off-chip memory units through I/O.
- Each SM has multiple processing units and a small L1 cache.
- This allows GPU architecture to be scaled up to tens of thousands of processing units allowing it to exploit parallelisms in deep learning kernels.

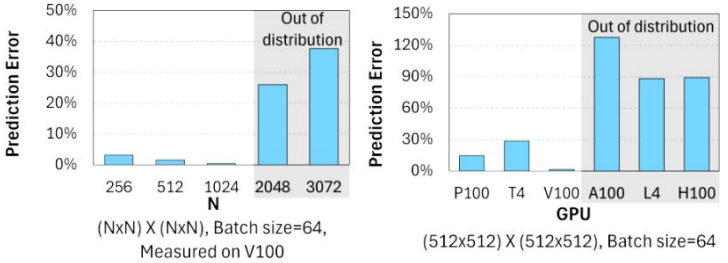
Deep Learning (DL) Execution on GPUs

- Each layer of a neural network is decomposed into multiple kernels by the DL library.
- These kernels are primarily decomposed into few types; General Matrix Multiplication (GEMM), fully-connected layer and pointwise operations (eg ReLU, GELU, tanh etc)
- Each of these kernel types executes differently
- Developers write the models in high libraries such as PyTorch and Tensorflow which call low level libraries (eg. CuDNN for NVIDIA) for the kernel.
- The DNN kernels execute sequentially in the GPU (supported by prior works).
- Thus the execution latency of the DNN is the sum of the latency of all the kernels.
- Distributed execution of the DNNs are done with parallelism techniques such as Pipeline parallelism and tensor parallelism.
- These includes the send/receive communication for pipeline parallel and collective communication such as all-reduce for tensor parallelism.

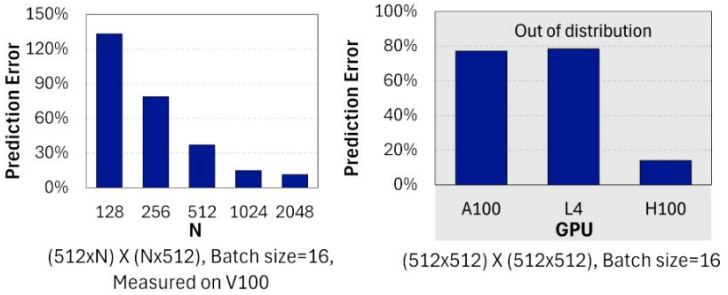
Predicting Batched MM by prior works

- **Habitat**
  - Habitat uses model trained with GPUs predating 2018, matris dim of 1024 and batch dim of 128
  - Thus it faces very high error on dim larger than 1024
  - It also struggles on unseen GPUs

- **Linear Regression**
  - This approach fails on matrix multiplication with smaller dimensions because of underutilization of GPUs
  - Similarly this also fails on unseen GPUs because of its simplistic extrapolation method



(a) Multi-Layer Perceptron Approach (Habitat) [62]



(b) Linear Regression-based Approach (Li et al.) [26]

Predicting Batched MM with larger predictors

To study the efficacy of using a larger model, Habitat's MLP was scaled and evey replaced with transformer based model

Result: Despite improved performance with larger models, all the evaluated models still show high percentage error exceeding 70% for out-of-distribution cases.

Predictor Architecture	Number of layers	In-distribution Prediction Error (%)	Out-of-distribution Prediction Error (%)
MLP	8	28.0	70.9
	16	22.3	81.4
Transformer	3	22.3	126.1
	6	21.0	86.4

Limitations of Existing Approaches

Limitations of existing approaches

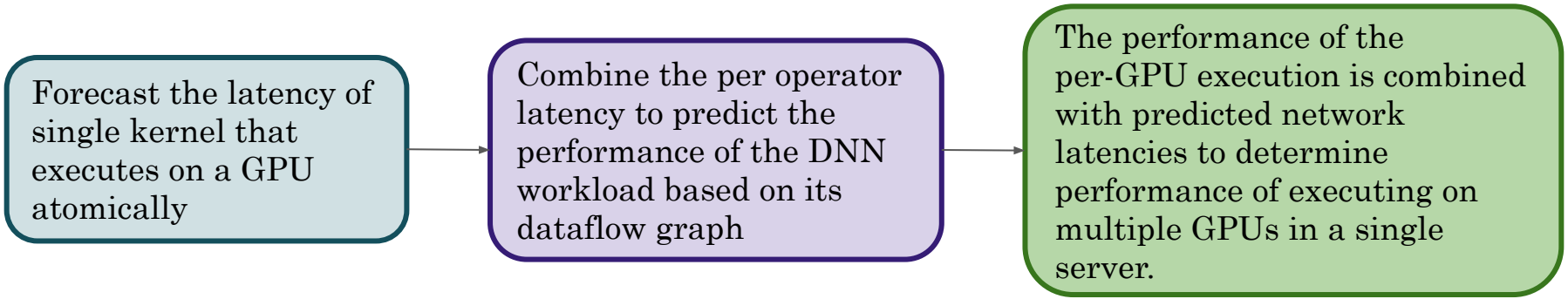
- **Lacks Foundational Knowledge**
  - Existing approach use rough analytical model without detailed information of GPU execution (hardware and software)
- **Extrapolation**
  - Some approaches require measurement on a certain device so it can be extrapolated to other devices.
- **GPU Architecture**
  - Models are unable to capture the necessary architectural information of the GPUs

GPU kernel execution involves complex hardware and software optimizations which is not captured by many machine learning predictors which only rely on high level features such as memory bandwidth and flops.

Existing approaches for DL latency prediction

- **Analytical Models**
  - Use simple analytical tools to estimate the performance of GPU
  - Some try to decompose the task into computation and communication, whereas some try to use reference performance from one GPU to estimate performance of another
- **Cycle accurate simulators**
  - Cycle simulators perform detailed modeling of each GPU component which is extremely slow and not suitable for end-to-end performance of large DL models
- **Machine Learning based approaches**
  - Several approaches have used regression models to leverage the linear nature of kernels for compute intensive tasks.
  - Some approaches use graph neural network to make prediction
  - However all these techniques require target GPUs for transfer learning.
- **CPU executions to predict speed up on GPU**
  - Executes the CPU code of target kernel to estimate the GPU speedup
  - This method assumes that the GPU implementation uses same algorithm with higher parallelizability which is not always the case.

NEUSIGHT Forecasting

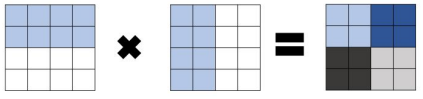


- Forecasting in NEUSIGHT works in 3 main steps
- Kernels uses several hardware and software optimizations to run across SMs
- NEUSIGHT leverages these architectural details such as memory size, number of SMs, HBM bandwidth, L2 cache size etc to make prediction of the kernels latency
- It differs from prior works which either fully rely on machine learning models or compute and memory capacity to make the predictions

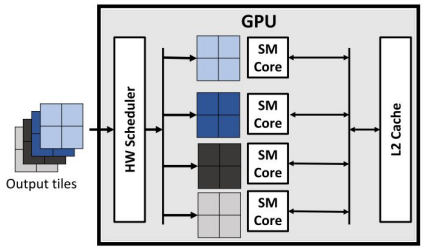
Kernel Execution in a GPU

Tiled execution

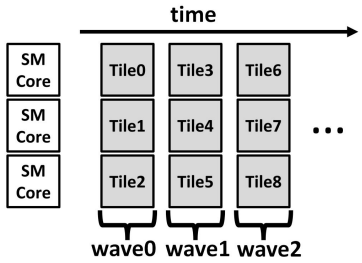
- Modern GPU libraries execute GEMM by decomposing it into multiple identical tiles where each tile is assigned to a SM
- These tiles are scheduled in the GPU in multiple waves based on the availability of the SMs
- The entire kernel is executed in multiple waves of tile groups or waves



(a) Tiling of output matrix.



(b) Mapping tiles to multiple SMs.



Fundamental Performance Laws of GPUs

- GPU kernel performance is bounded by the peak FLOPs and peak memory bandwidth
- Using roofline analysis, the performance can be predicted
- Although roofline analysis shows the peak achievable performance, usually the kernel is unable to fully utilize the available resources

$$K = \frac{flops_k}{mem_k};\ roofline_{BW} = min(K \times memBW_p, flops_p)$$

Kernel-wise Prediction

$$num_{tiles} = \Pi_{i=1}^{N} \lceil \frac{x_i}{t_i} \rceil$$

$$num_{waves} = \lceil \frac{num_{tiles}}{num_{sm}} \rceil$$

$$PerOpLatency = PerTileLatency \times num_{waves}$$

Tile Decomposition

DNN Libraries run larger GEMMs in smaller tiles (32-256). ML models can be used to predict per tile latency, combined together to predict the per operator latency.

x_i and t_i are the output and time dimensions whereas N is the number of output dimensions.

$$PerTileLatency = \frac{flops_{tile}}{achieved_{BW}}$$

$$achieved_{BW} = roofline_{BW} \times utilization$$

Imposing performance laws

However *PerTileLatency* relies on the compute utilization of SMs and the achieved bandwidth of the kernel and is physically bounded by $roofline_{BW}$

$$utilization = alpha - \frac{beta}{num_{waves}}$$

$$alpha, beta = \sigma(MLP(input\_features))$$

MLP Prediction

Finally the correlation between number of waves and utilization are formulated as ML model to capture the non-linear behavior between kernel latency and GPU characteristics and operator properties.

The *alpha* and *beta* factors constrain the utilization by applying the fundamental performance laws.

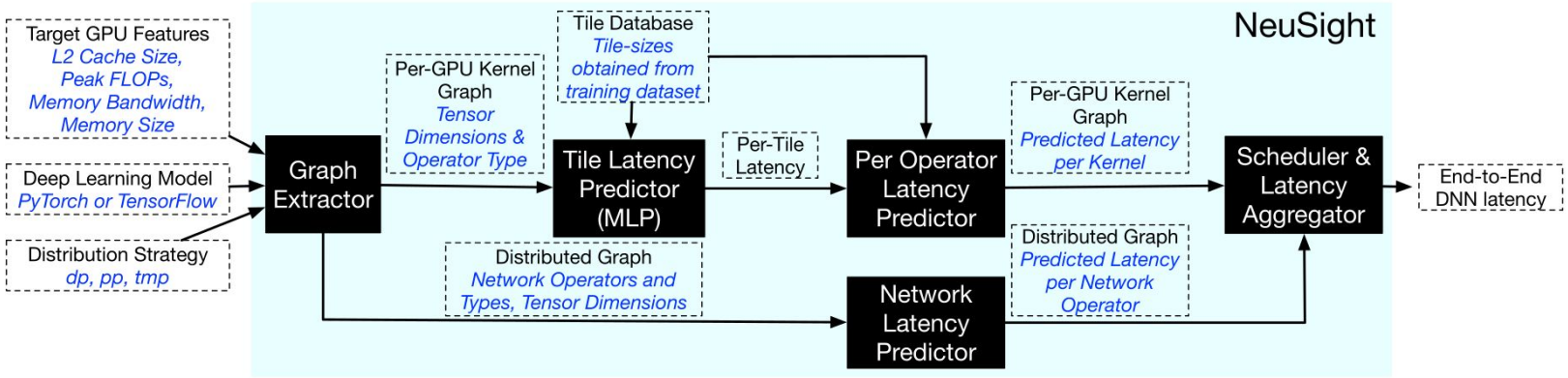
ML model to predict utilization

- **Predictor Model details:**
  - **Model**: MLP
  - **Layers**: 8
  - **Hidden-dim**: 512
  - **Activation**: ReLU
  - **GPU Features**: Memory size, Bandwidth, peak FLOPS, L2 Cache size
- **Predictor Classes:**
  - 5 different MLPs are used for different kernels.
  - These include BMM, Fully connected layers, Element wise operators, Softmax and layer norm
- For other unseen kernels, they are considered memory bound; latency is calculated by dividing memory requirement by bandwidth.

Input Features	Unit
$\frac{FLOPsPerTile}{PeakFLOPSPerSM}$	$\frac{GFLOPS}{TFLOPS/s}$
$\frac{MemoryPerTile}{MemoryBWPerSM}$	$\frac{MB}{GB/s}$
$\frac{num_{waves} \times MemoryPerTile}{L2CacheSizePerSM}$	$\frac{MB}{KB}$
$\frac{num_{waves} \times MemoryPerTile}{MemorySizePerSM}$	$\frac{MB}{MB}$
$\frac{FLOPsPerTile \,/\, MemoryPerTile}{PeakFLOPS \,/\, MemoryBW}$	$\frac{GFLOPS \,/\, MB}{(TFLOPS/s) \,/\, (GB/s)}$

Features used to train the MLP model

NEUSIGHT Workflow



- NEUSIGHT takes the description of the target deep learning model in PyTorch and extracts the operator/kernel graph using the Torch.fx library.
- Based on the kernel metadata and tile size, each operator is annotated with predictions made by the kernel-level predictor.
- By accumulating the predicted latency of each kernel executing on a single device, which aligns with device execution where kernels are executed sequentially on the GPU
- For parallel training NEUSIGHT can use parallel strategy(dp, pp degrees and schedule) to formulate the latency of the DNN model

Forecasting for Distributed Execution

- NEUSIGHT can forecast performance for distributed execution on a single multi-GPU server
- This is achieved by adding necessary communication for each type of parallelism on the DNN graph. (eg. send/recv for PP, allreduce for DP and TP etc.)
- The performance of network operators is estimated by measuring the link bandwidth, peak bandwidth and the network link utilization.
- Pipeline Parallelism also involves pipeline schedule (eg. 1f1b) which changes the overall latency. NEUSIGHT uses the GPipe schedule which can be easily extended for other form of parallelisms.
- For Tensor parallelism, NEUSIGHT supports the megatron tensor model strategy; it inserts the all-reduce operations wherever necessary based on the strategy.

Experimental Setup: Hardware and Dataset

Vendor	GPU	Year	Peak FLOPS (TFLOPS)	Memory Size (GB)	Memory BW (GB/s)	# SMs	L2 Cache (MB)
NVIDIA	P4	2016	5.4	8	192	40	2
	P100	2016	9.5	16	732	56	4
	V100	2017	8.1	32	900	80	6
	T4	2018	14.1	16	320	40	4
	A100-40GB	2020	19.5	40	1555	108	40
	A100-80GB	2020	19.5	80	1935	108	40
	L4	2023	31.3	24	300	60	48
	H100	2022	66.9	80	3430	132	50
AMD	MI100	2020	23.1, 46.1(Matrix)	32	1230	120	8
	MI210	2021	22.6, 45.3(Matrix)	64	1640	104	16
	MI250 (per die)	2021	22.6, 45.3(Matrix)	64	1640	104	16

Kernels	Data points	Range
BMM	87627	BS and dim: 1-1024
FC Layer	32256	BS: 1-8192 Output: 1-65536
Element-wise	26066	BS: 512-16384 Vector Size: 512-4096 Add, div, Mul, GELU, ReLU, tanh
Softmax	1807	BS: 4096 - 16384 Vector size:512 - 4096
Layer Norm	1501	BS: 4096 - 16384 Vector size: 512 - 4096

Hardware

Several NVIDIA and AMD GPUs were used, including newer GPUs as out-of-distribution GPUs.

Dataset

Data were measured by running operators 25 times and averaged results 20% of data was used for validation and rest for training.

Experimental Setup: Training and Workloads

Baselines:

- Roofline Analysis
- Habitat
- Li et al.(Linear Regression)

Training:

- MLPs for 100 epochs
- AdamW optimizer
- lr 1e-6 - 5e-3
- Batch Sizes 16 - 128

Tile Size:

- Obtained from the kernel name profiled by pytorch profiler for matrix multiplication
- For other kernels, deduced from the number of thread blocks

Model	Year	Parameter Size	# of Layers	# Attention Heads	Hidden Dimensions	Sequence Length
BERT Large	2018	340M	12	16	760	512
GPT2 Large	2019	774M	36	20	1280	1024
GPT3 XL	2020	1.3B	24	24	3072	2048
OPT 1.3B	2022	1.3B	24	24	2048	2048
GPT3 2.7B	2020	2.7B	32	32	2560	2048
SwitchTrans	2021	5.3B	24	32	1024	512

Workloads Evaluated:

- 6 transformers models were used to evaluate the prediction
- Time to generate the first token is used as latency metric
- Training time is the latency of single forward and backward pass
- Predictions were compared with the measurement to evaluation

Evaluation: end-to-end mean percentage error

Technique	Error (Inference)	Error (Training)
Roofline	31.2%	31.9%
Habitat	220.9%	725.8%
Li et al.	61.2%	58.3%
NEUSIGHT	7.3%	9.7%

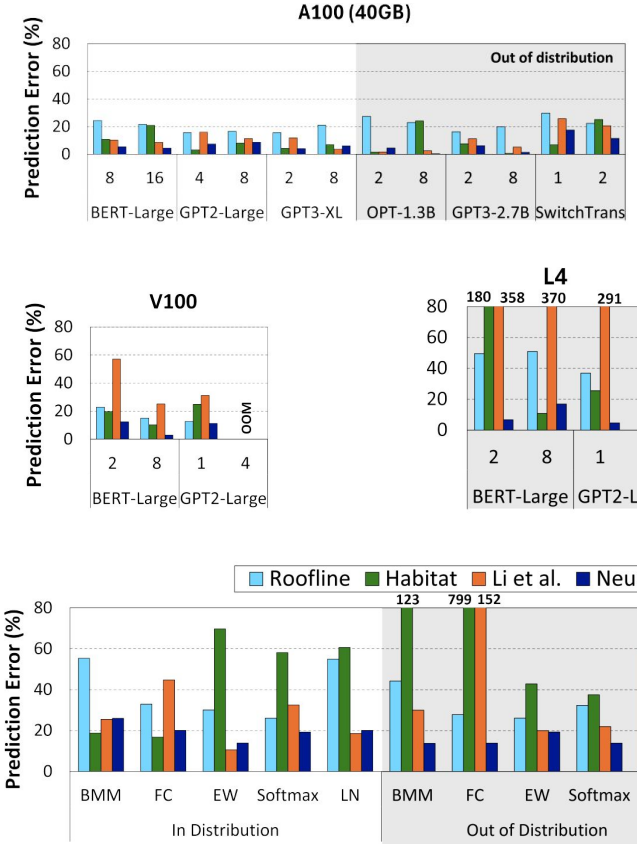
NEUSIGHT Specifically outperforms other approaches in out of distribution GPUs where it has significantly lower error compared to the other approaches.

Per operator Prediction

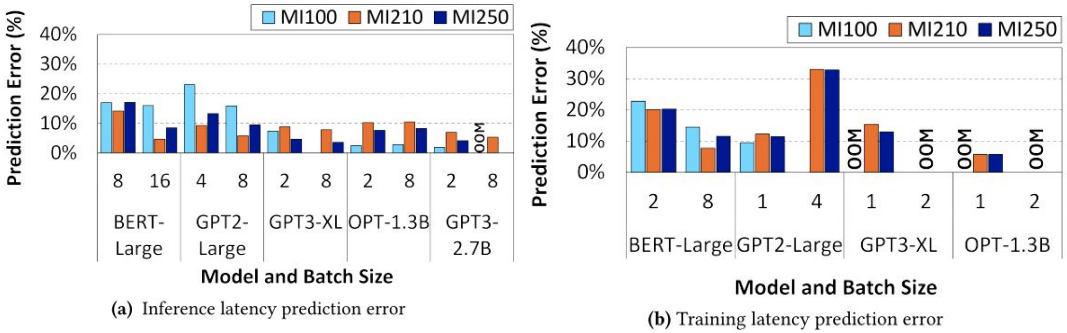
On per-operator prediction, Roofline analysis shows higher error compared to NEUSIGHT in bot in and out of distribution data

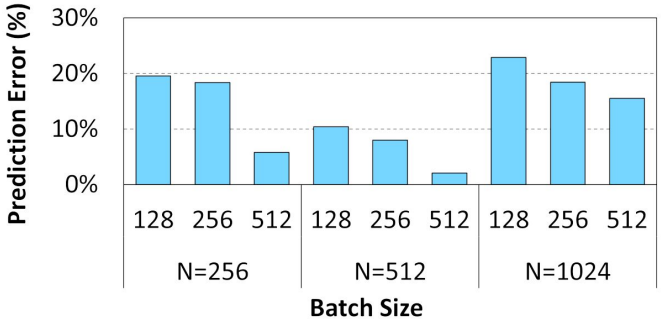
NEUSIGHT significantly outperforms Habitat and Li et al. on out-or distribution data

Prediction of GEMM is most important as it contributes most to training time



Evaluation: AMD GPUs and New Numerical Units





NEUSIGHT on AMD GPUs

NEUSIGHT trained with MI100 and MI210 GPUs
MI250 was used as out-of-distribution GPU

NEUSIGHT achieves an average percentage error of 8.8% for inference and 15.7% for training.

NEUSIGHT on New hardware and numerical units

While using NEUSIGHT On FP16 numerical units and H100 tensor cores, for BMM, average error of 13% was observed

Evaluation: Distributed Execution

	Global Batch Size		A100-40GB x 4 (NVLink)			H100 x 4 (DGX Box)		
			Data Parallel	Tensor Parallel	Pipeline Parallel	Data Parallel	Tensor Parallel	Pipeline Parallel
GPT2-Large	4	Measured Latency (ms)	452.3	484.9	1597.6	194.9	211.1	637.5
		NeuSight Prediction (ms)	510.5 (12.9%)	525.6 (8.4%)	1761.6 (10.3%)	214.1 (9.9%)	228.1 (8.0%)	650.2 (2.0%)
	16	Measured Latency (ms)	OOM	OOM	OOM	642.1	729.7	2384.0
		NeuSight Prediction (ms)				649.6 (1.2%)	825.1 (13.1%)	2638.4 (10.7%)
GPT3-XL	4	Measured Latency (ms)	OOM	OOM	OOM	OOM	1010.5	3554.4
		NeuSight Prediction (ms)					1047.6 (3.7%)	3717.0 (4.6%)

Distributed Setups

Setups	4 x A100s-40GB, NVLink (600GB/s) 4 x H100 DGX (900GB/s)
Models	GPT2-Large GPT3-XL

Across all the models, NEUSIGHT predicts the latency of distributed training with an average error of **7.7%**, including **6.7%** for the H100 server and **10.5%** for the A100 server

Evaluation: Multi-Node Distributed Execution

# Nodes	1	4	384	768	3840
NeuSight Prediction (ms)	1514.9	1836.7	12028.3	12135.5	12564.6

NeuSight's predictions can be integrated with network simulators to forecast multi-node distributed training.

Setup: 8 x H100 nodes, multi-level fat-tree network with InfiniBand(100Gbps). Levels consisting of 4, 383, 768 up to 2340 nodes, inter-node bandwidth of 900GB/s

Parallelism: tp=8, pp=number of nodes

NEUSIGHT handles per node prediction, whereas the inter-node network performance is predicted

Contributions

- Divide the problem of end-to-end latency prediction into regular and manageable sub-problems, predicting the latency at the tile granularity and use MLP model to predict the utilization of the device based on the DNN kernel and GPU properties
- Apply basic laws of performance to enhance the latency prediction at the tile granularity
- Extend this approach for latency prediction of distributed execution within a server by integrating the prediction of the latency of communication
- Perform evaluation on a diverse set of GPUs and workloads to demonstrate the versatility of the prediction technique even in out-of-distribution systems.

Limitations

- This approach has not explored this application with compilers such as XLA
- While this paper shows that prediction error is very low for out-of-distribution GPUs, training data still requires wide range of GPUs
- This work has not demonstrated the direct application of latency prediction for tasks such as buying or renting cloud GPUs or parallel optimization, which was done by previous approach (Habitat)
- While this approach uses the novel tile based estimation for the compute part of the kernels, the communication estimation still relies simple bandwidth based communication. This may be improved with better communication estimation for distributed training.

Thank You

Questions