A High-Density GPU Solution for DNN Training

NUANCE: *F. Mana, R. Gemello, V. Kataev, D. Albesano, P. Zhan*
NVIDIA: *P. Micikevicius*
March 17, 2015

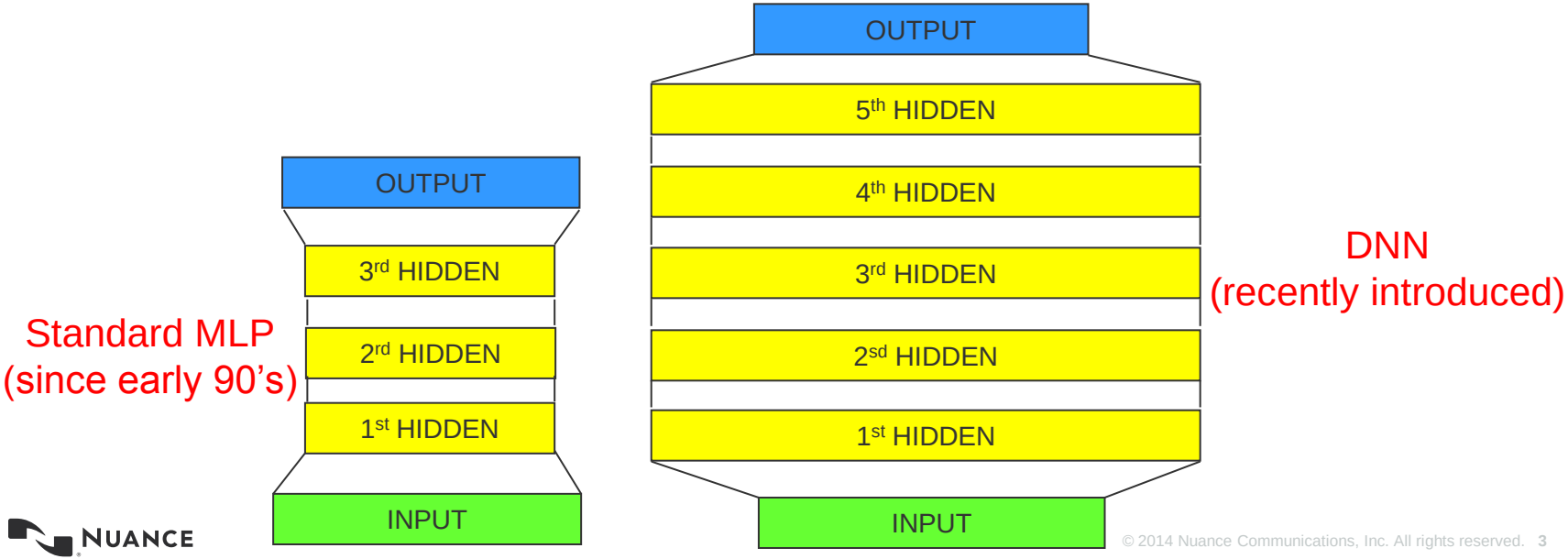
Agenda

- NN technology and HMM-NN hybrid ASR system
- DNN training algorithms
- Speeding up strategies
  - Algebraic approach
  - Model splitting approach
  - Data splitting approach
- Why High-Density GPU solution?
- Experimental results on Nvidia PSG cluster.

NUANCE

Neural Network Technology

Neural network is a computational model based on simple processing units (neuron) arranged in layers. The kind of computation is ruled by the weight matrixes connecting each layer. Processing takes place feeding an input vector and propagating it forward till the output layer.

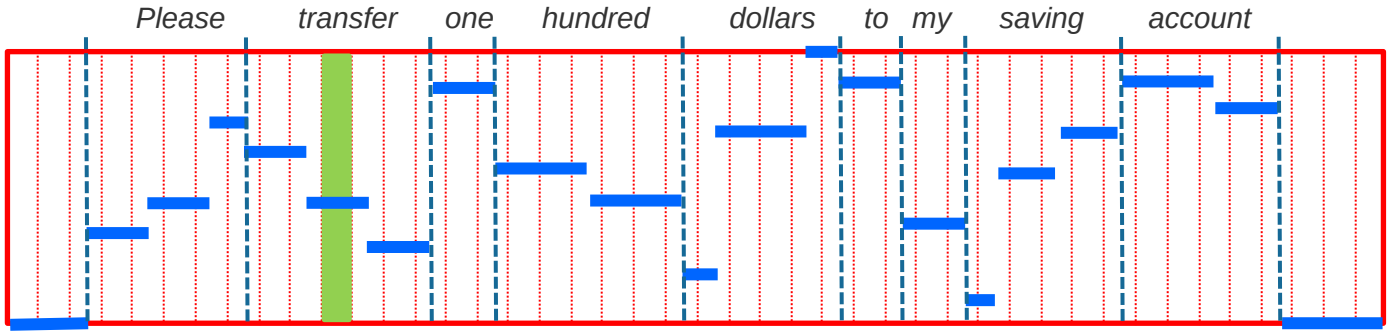


NUANCE

ASR Overall Architecture

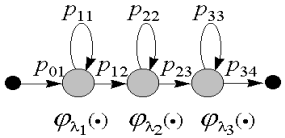
Word domain
(most likely states or word sequence)

States domain
(inventory of language based on elementary sounds)



Observation domain
(states posterior probability)

HMM
(state emission prob)



$$\varphi_{\lambda=\{w_i,\mu_{ij},\sigma_{ij}\}}(o)=\sum_{i=1}^{M} w_i \cdot \frac{1}{\sqrt{2\pi D}} e^{\sum_{j=1}^{D}\frac{(o_{ij}-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$$\sum_{i=1}^{M} w_i = 1$$

left context center right context

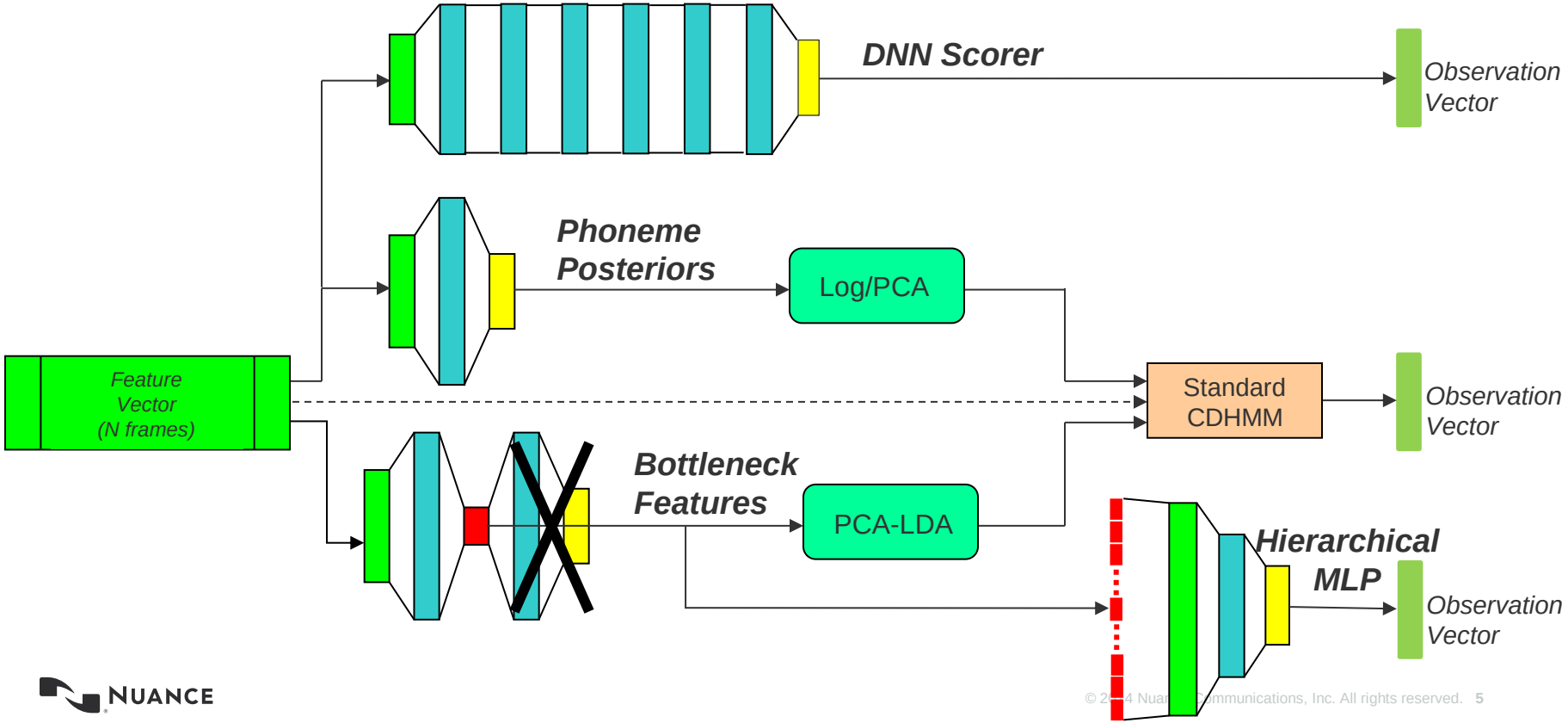
time

Feature domain
(sequence of frames, after signal processing)

Time domain
(sequence of samples)

NUANCE

Hybrid HMM/NN ASR System



DNN Training Algorithm

- DNN training is done using a training data sets that is made up by a set of *(input, target)*. It is based on minimizing an error function (i.e. Cross-Entropy) defined on the desired targets and the current DNN outputs.

- *Stochastic Gradient Descent (SGD)* is the most commonly used optimization procedure for frame level discriminative training of DNN.

$$\boldsymbol{W(t+1) = W(t) + \mu * \Delta W(t)}$$

$$\boldsymbol{\Delta w_{ij} = -\frac{\partial E(output, target)}{\partial w_{ij}}}$$

- Unfortunately, optimization in SGD is an inherently sequential process that is difficult to parallelize. This is because DNN weights are updated for every mini-batch feature vectors.

NUANCE

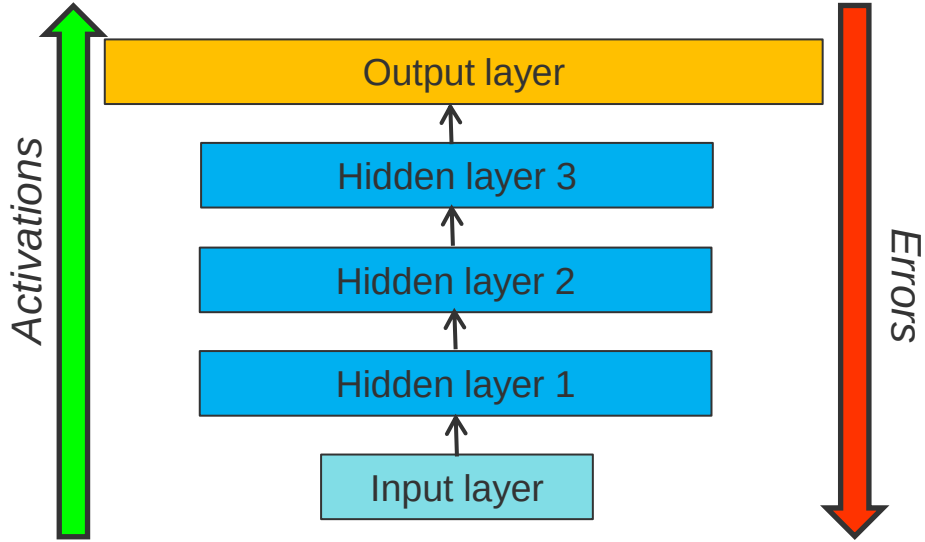
DNN Training Parallelization

Outline of the Methods for DNN training parallelization.

- Algebraic approach → to leverage parallel and efficient implementation of matrix product
- Partitioning of Data (Data Parallelization) → to assign to each computational node the whole model and a part of the data
- Partitioning the Net Structure → to assign to each computational node all the data and a part of the model.
  - Vertical Partitioning (Striping)
  - Horizontal Partitioning (Pipelining)

DNN Training Parallelization: Algebraic Approach

- DNN training is based on two main steps: forward & backward.
- First action: using a mini-batch of input data. Three (parallel) Matrix-Matrix operations involved.
  → **O** = f(**X*W** + **b**)
- Second Action: dispatch a single Matrix product through several GPUs
  → CUDA 6.0+ supports MultiGPU cuBLAS



NUANCE

DNN Training Parallelization: Algebraic Approach

	CUDA 6.0			CUDA 6.5
Layer / (Layer+1) Inputs x Outputs	Nvidia K10	Nvidia K20	Nvidia K40	Nvidia K80
600x2048	1.35 ms	0.86 ms	0.89 ms	**0.78 ms**
2048x2048	3.63 ms	2.40 ms	2.35 ms	**1.96 ms**
2048x30000	52.20 ms	28.69 ms	27.71 ms	**23.52 ms**
2048x256	0.62 ms	0.60 ms	0.59 ms	**0.45 ms**
256x30000	13.80 ms	5.78 ms	5.21 ms	**4.37 ms**

Benchmark detail for [600x2048] row (elapsed msec):

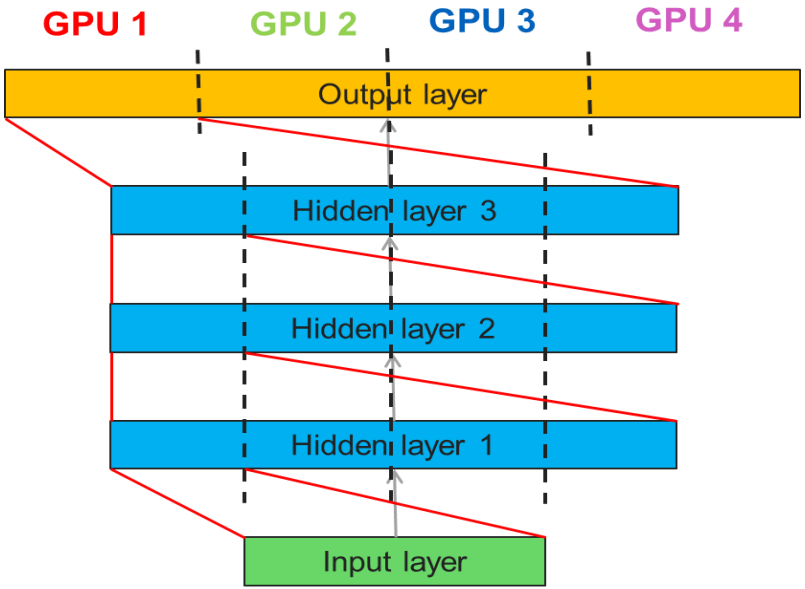
Iterates Niter times the SGEMMs (BatchSize=128):

1) $O = I*W$ [BSx2048]=[BS 600]*[600 2048]
2) $I = O*W^T$ [BSx600]=[BSx2048]*[2048x600]
3) $W = I^T*O$ [600x2048]=[600 BS][BS 2048]

NUANCE

DNN Training Parallelization: Striping

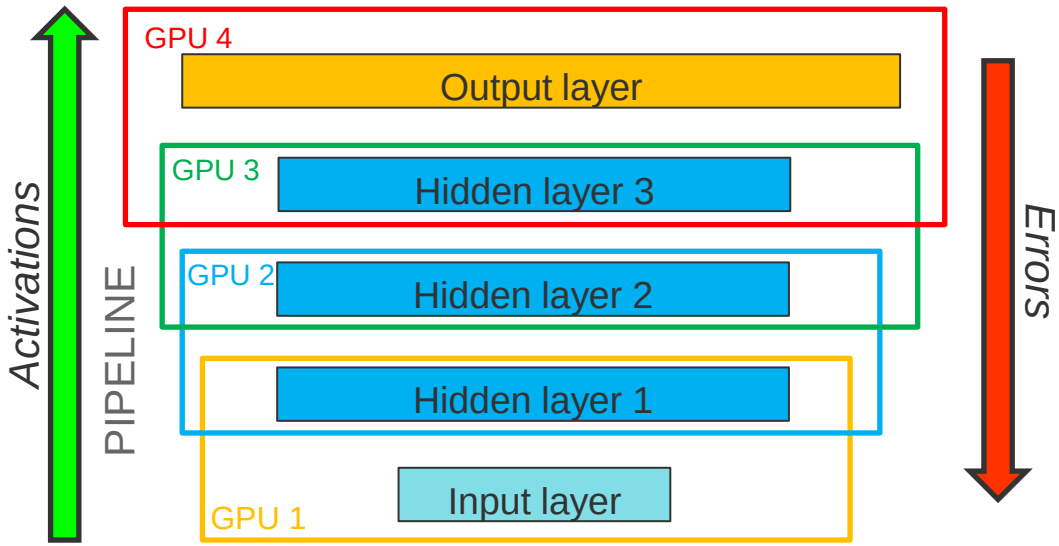
- NN is split vertically. The basic idea is to partition each layer of the network into stripes and parallelize across them.
- Activations must be transferred across stripes in order to arrange the full layer activation vector
- This method is suitable when layer size is large enough to benefit from splitting them on many GPUs (usually output layer).



NUANCE

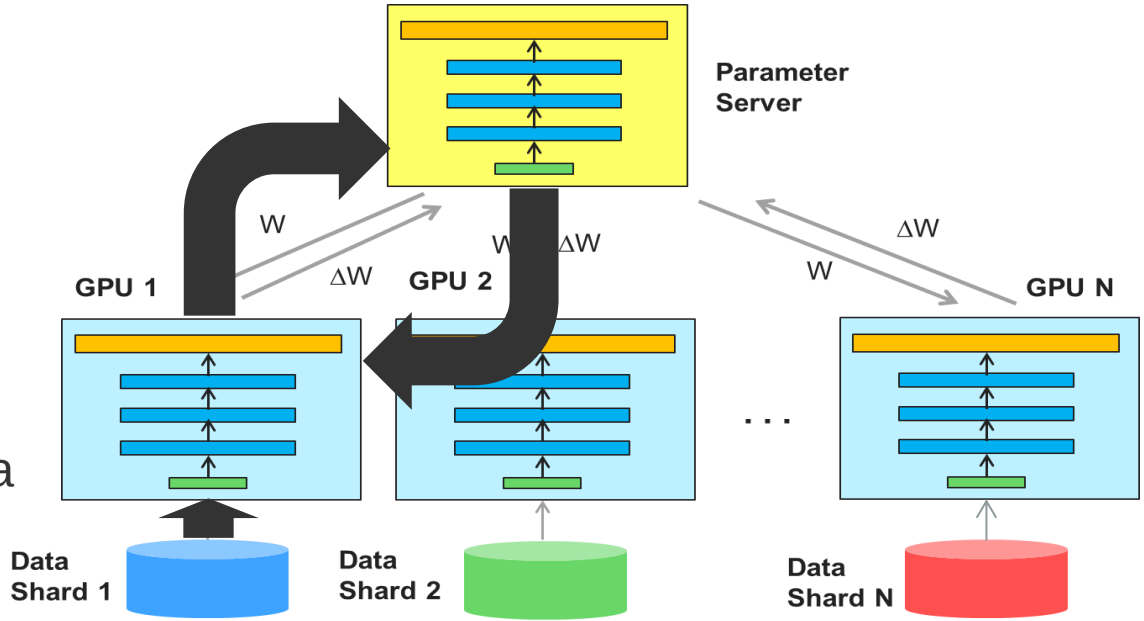
DNN Training Parallelization: Pipelining

- NN is split horizontally and one or more layers are assigned to each computational node. This is called *Pipelined back-propagation*.
- All GPUs work simultaneously on the data they have, like in an *assembly line*, and data flows from GPU to GPU, Activations flow forward and Errors flow backward.



DNN Training Parallelization: Data Splitting

- It is the idea of parallelizing through splitting the training corpora
- There is a central coordinator (parameter server) and several computational nodes.
- Each computational node has a replica of the whole model and its own shard of the training material.

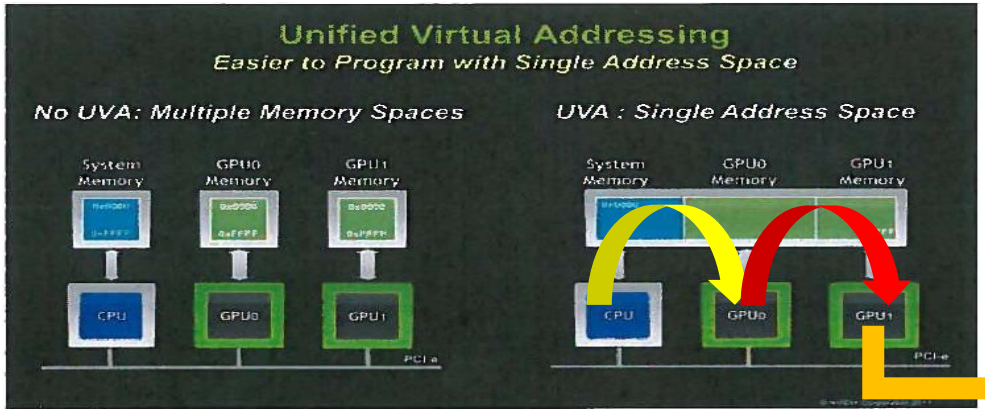


GPU: The Hardware

- Each single GPU: several cores, RAM, multi thread execution, floating point.

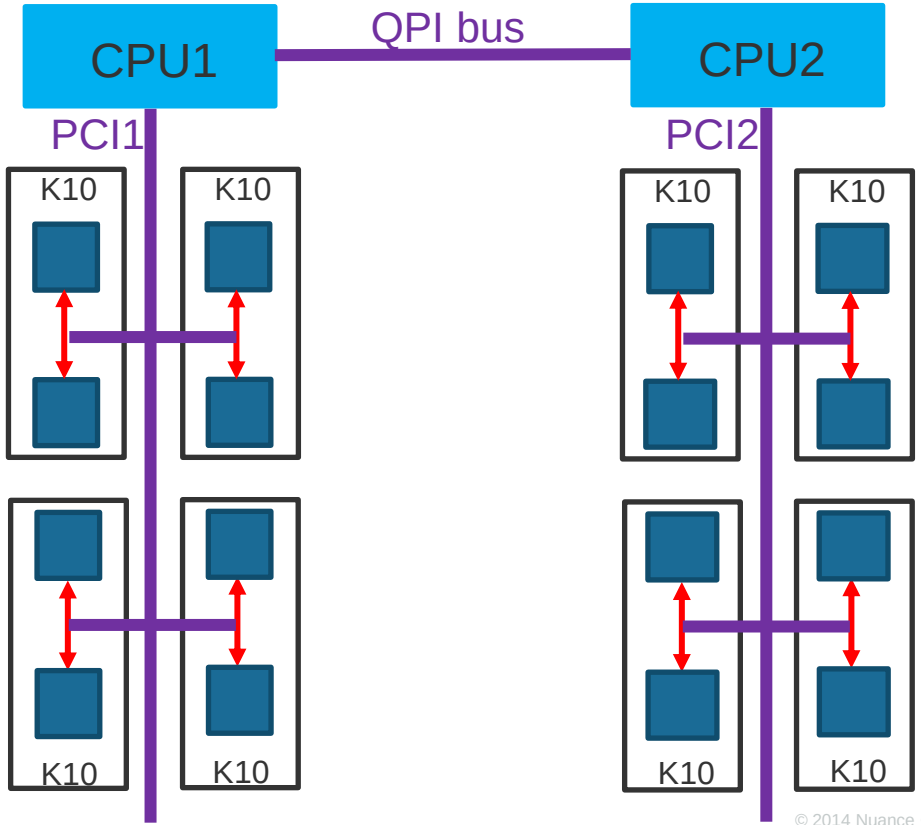
K10 (2xGPU per board)	3072cores / 8G	/ 4.58 Tflops	SP
K20 (1xGPU per board)	2496cores / 5G	/ 3.52 Tflops	SP
K40 (1xGPU per board)	2880cores / 12G	/ 4.29 Tflops	SP (Base/Boost Clocks)
K80 (2xGPU per board)	4992cores / 24G	/ 5.60 Tflops	SP (Base/Boost Clocks)

- CUDA supports UVA plus P2P for a good connectivity inside/outside the node.



NUANCE

GPU: High-Density Solutions



GPU Server Setup

GPU Server Nickname	GPU Configuration	GPU Direct/NotDirect P2P bandwidth	GPU<->CPU bandwidth	CPU
K10	**8x K10.G2.8GB (4peered+4peered)**	**Root1|1 ~11.0GB** *GPU0<->GPU1|3* **Root1|2 ~9.5GB** *GPU0<->GPU4|7*	**10.95GB**	**E5-2690 v2 @3.00GHz**
K80	**8x K80 (4peered+4peered)**	**Root1|1 ~11.5GB** *GPU0<->GPU1|3* **Root1|2 ~9.5GB** *GPU0<->GPU4|7*	**10.80GB**	**X5670 @2.93GHz**

NUANCE

DNN Training Setup

We used an internal DNN training tool in this evaluation

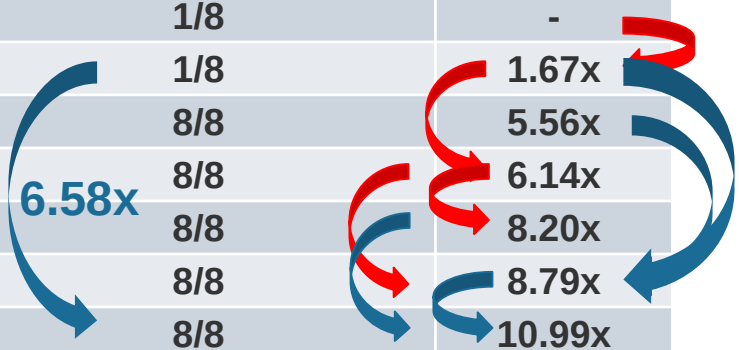
We used 95 hours data to train a 8 hidden layer DNN with SGD algorithm and Cross-Entropy criterion. Each DNN hidden layer has 2048 units while the LRF layer has 256 units. The softmax output layer has 10000 units.

We parallelized the training process by splitting the model and data in the trainings which use multiple GPU devices [ref 1,2].

We benchmarked K10 vs K80 systems provided us by NVIDIA PSG cluster with a number of GPU devices varying from 1 to 8.

DNN Training Times

GPU	Connection Type	#Jobs	#UsedGPUs / #AvailableGPUs	Speed Up
K10	-	1	1/8	-
K80	-	1	1/8	1.67x
K10	Not Peered Custom	8	8/8	5.56x
K80	Not Peered	8	8/8	6.14x
K80	Not Peered	16	8/8	8.20x
K80	Not Peered Custom	8	8/8	8.79x
K80	Not Peered Custom	16	8/8	10.99x



- K80 is faster than K10 (1.67x 1.58x).
- High-Density solution with 8GPUs can provide nice training speed up (3.67x 5.26x).
- More boosts can be achieved by submitting more tasks to the same GPU (1.34x 1.25x).
- A custom allocation of tasks over the GPUs optimizes the bandwidth capacity of the server (1.43x 1.34x).

NUANCE

Conclusions

- NN is a computational model based on distributed computation and well suitable for parallel hardware (Matrix-Matrix operation) implementation.
- The SGD training is not a batch algorithm and it cannot be easily parallelized. Any effort to change SGD in a way more comfortable for parallel computation introduces approximations that can harm the training convergence.
- Both Data and Model Splitting techniques need to transfer frequently activations and/or errors and/or models across the computational nodes. These transfers require a lot of bandwidth.
- High-Density GPU solution is an efficient way for parallelizing DNN training process. The highest bandwidth (9-11GB/s) inside a single server allows to achieve good speed-up rates w.r.t 1Job/1GPU.
- K80 greatly improves K10 in terms of training speed.

NUANCE

References

[1] Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide, "Pipelined Back-Propagation for Context-Dependent Deep Neural Networks", Interspeech 2012, Portland, OR, USA.

[2] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, Bo Xu, "Asynchronous Stochastic Gradient Descent for DNN Training", ICASSP 2013, p. 6660-6663

NUANCE

Thanks
for your attention

