Large Scale DNN Training with Keras + TensorFlow + Horovod + Azure Batch AI

Chris Auld (@cauld)
Software Engineering Manager
Microsoft - Southeast Asia

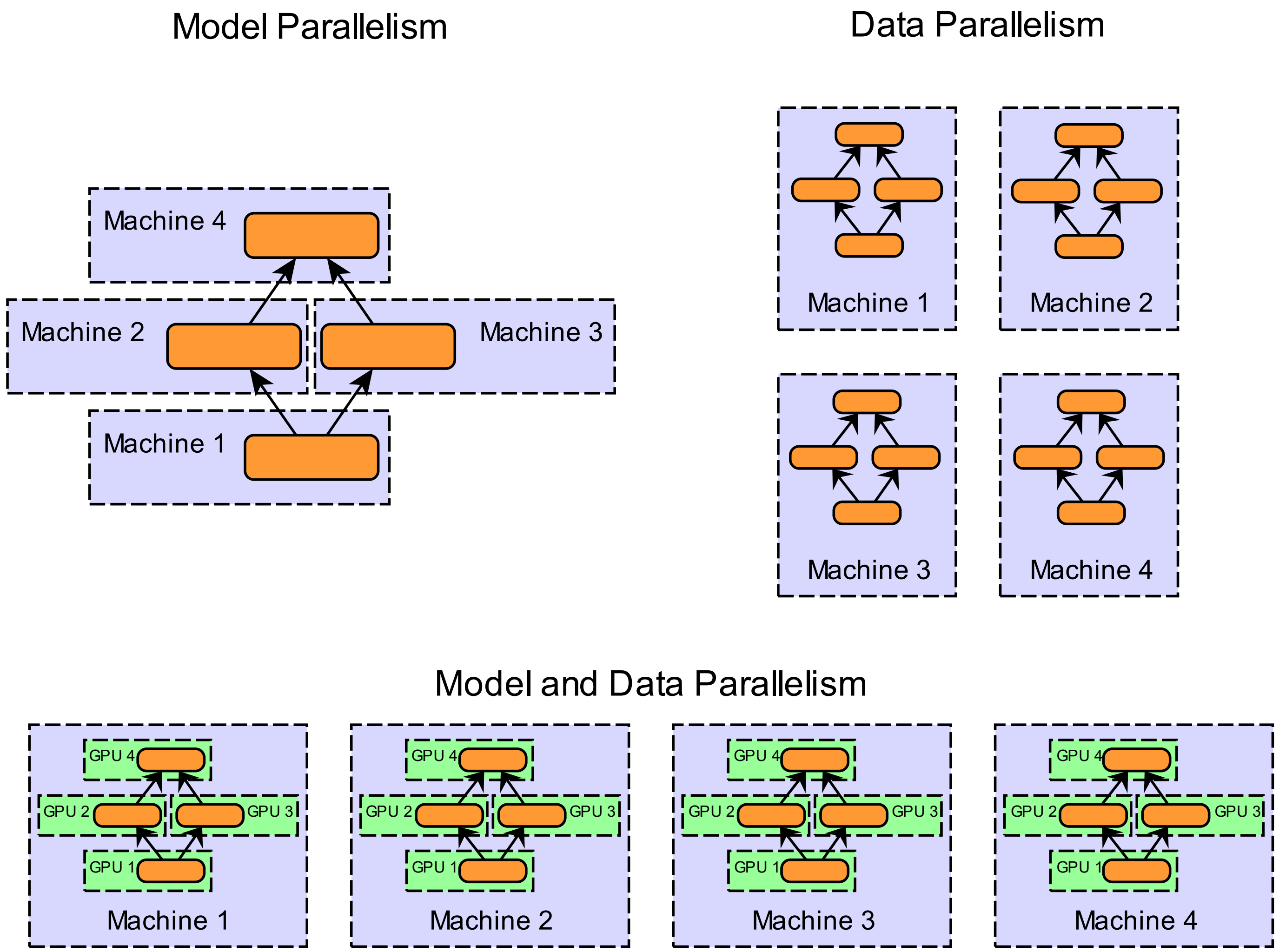
Microsoft

Distributed Training Approaches

Distributed Training

- More Models
  - Hyper-parameter optimization
  - Architecture experimentation
- Faster Models
  - Too much data, too much computation
  - Data Parallel; Need to update weights or share gradients
  - Bandwidth rapidly becomes a problem.
- Bigger models
  - GPUs have more RAM… but…
  - Model parallel or Hybrid
  - Complexity of mapping computation graph to physical topology

Different distribution approaches



https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

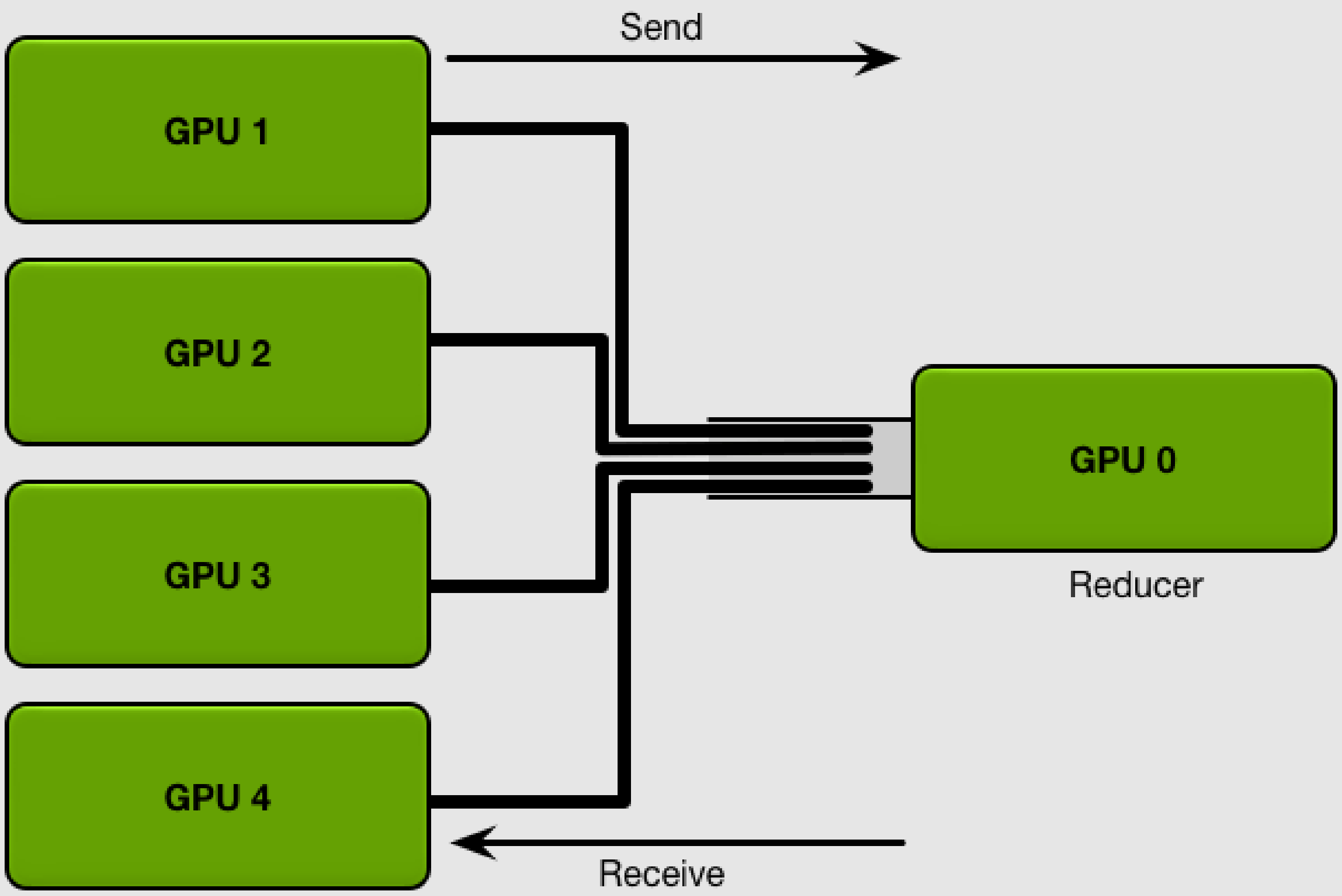
Approaches with Tensorflow

Tensorflow Distributed

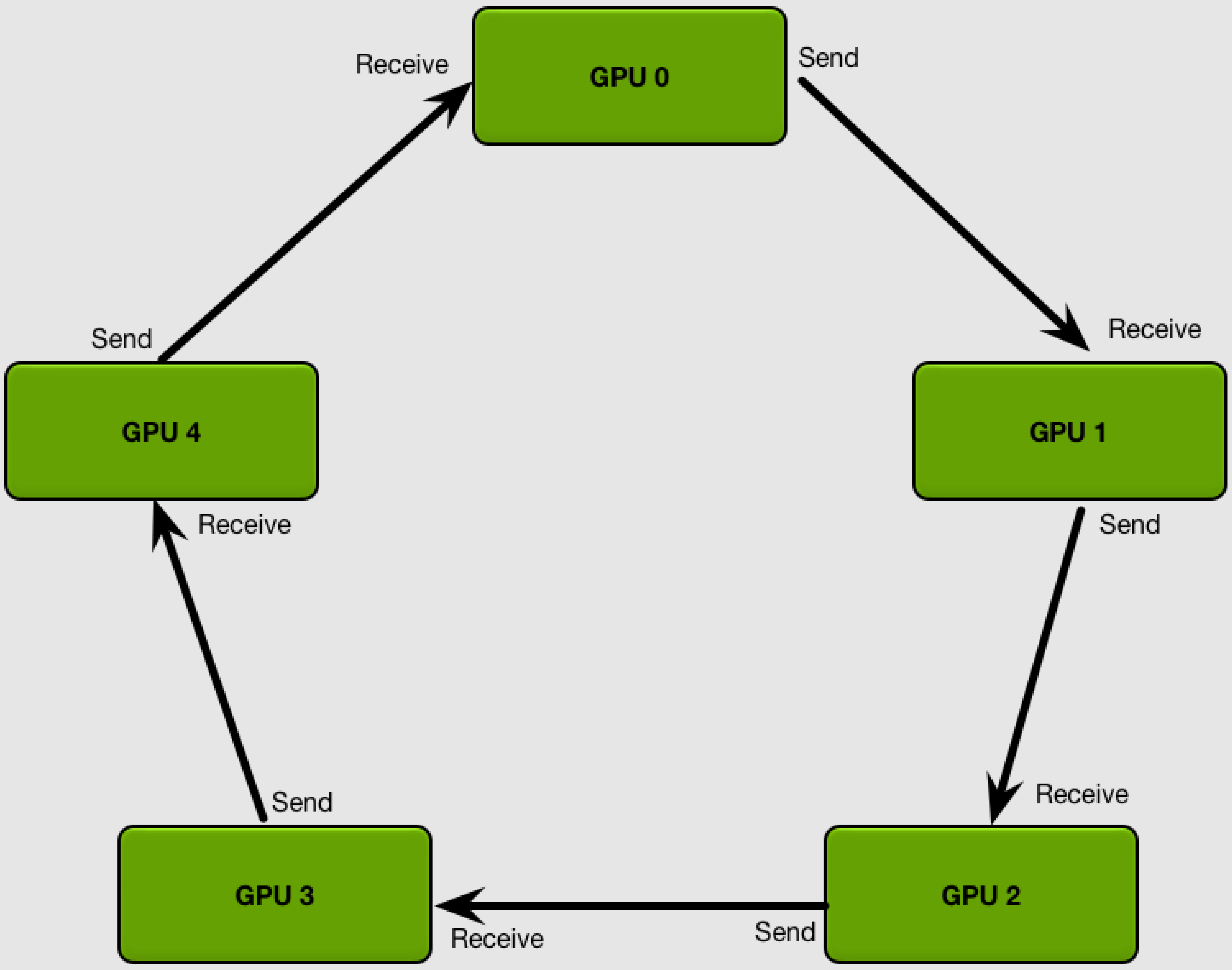
- Model Parallel
- Data Parallel via Parameter Server
- Limits to scale on smaller models
- Significant code changes

Horovod (Uber)

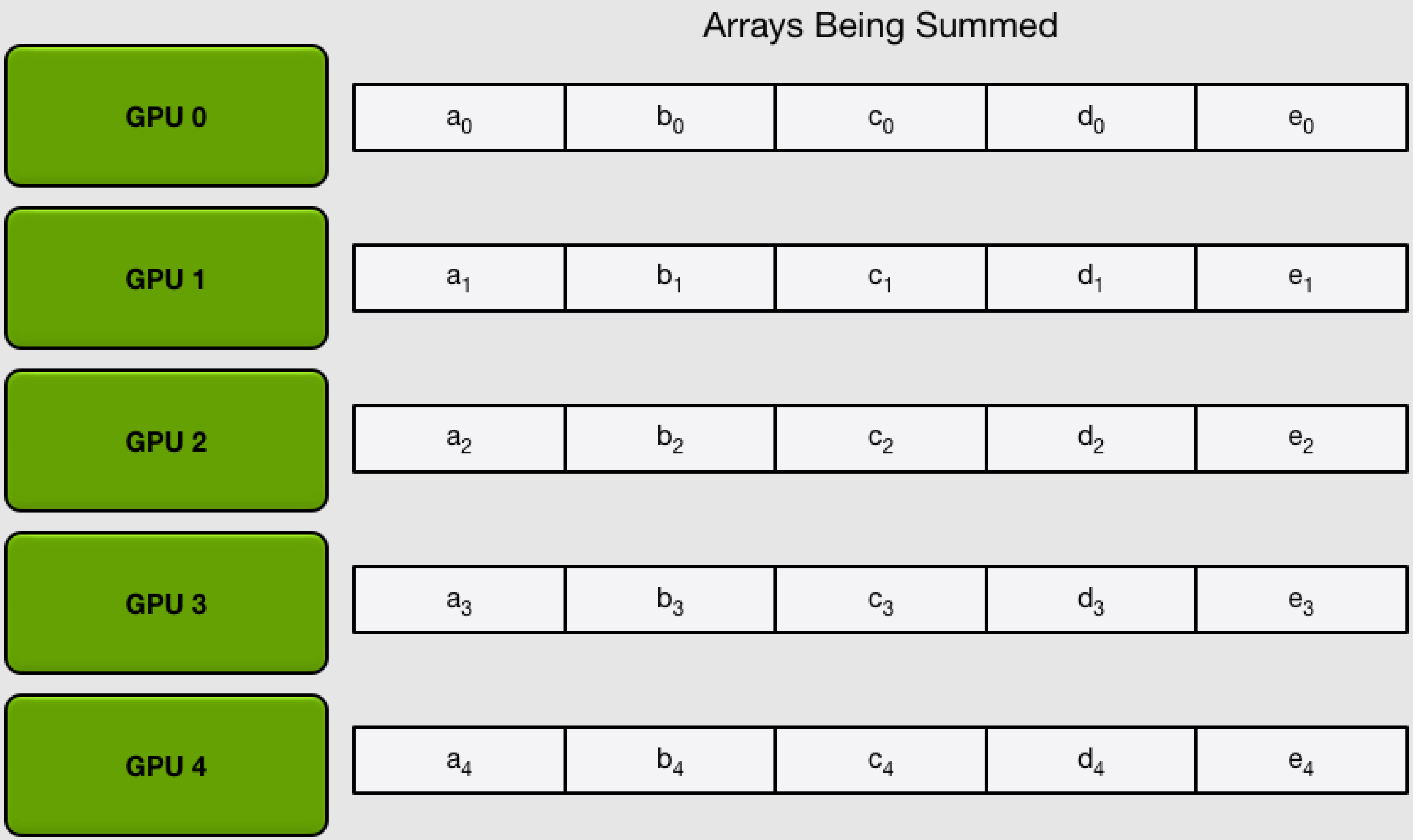
- Data parallel
- Implements the (Baidu) MPI ring all reduce pattern
- Minimal code changes

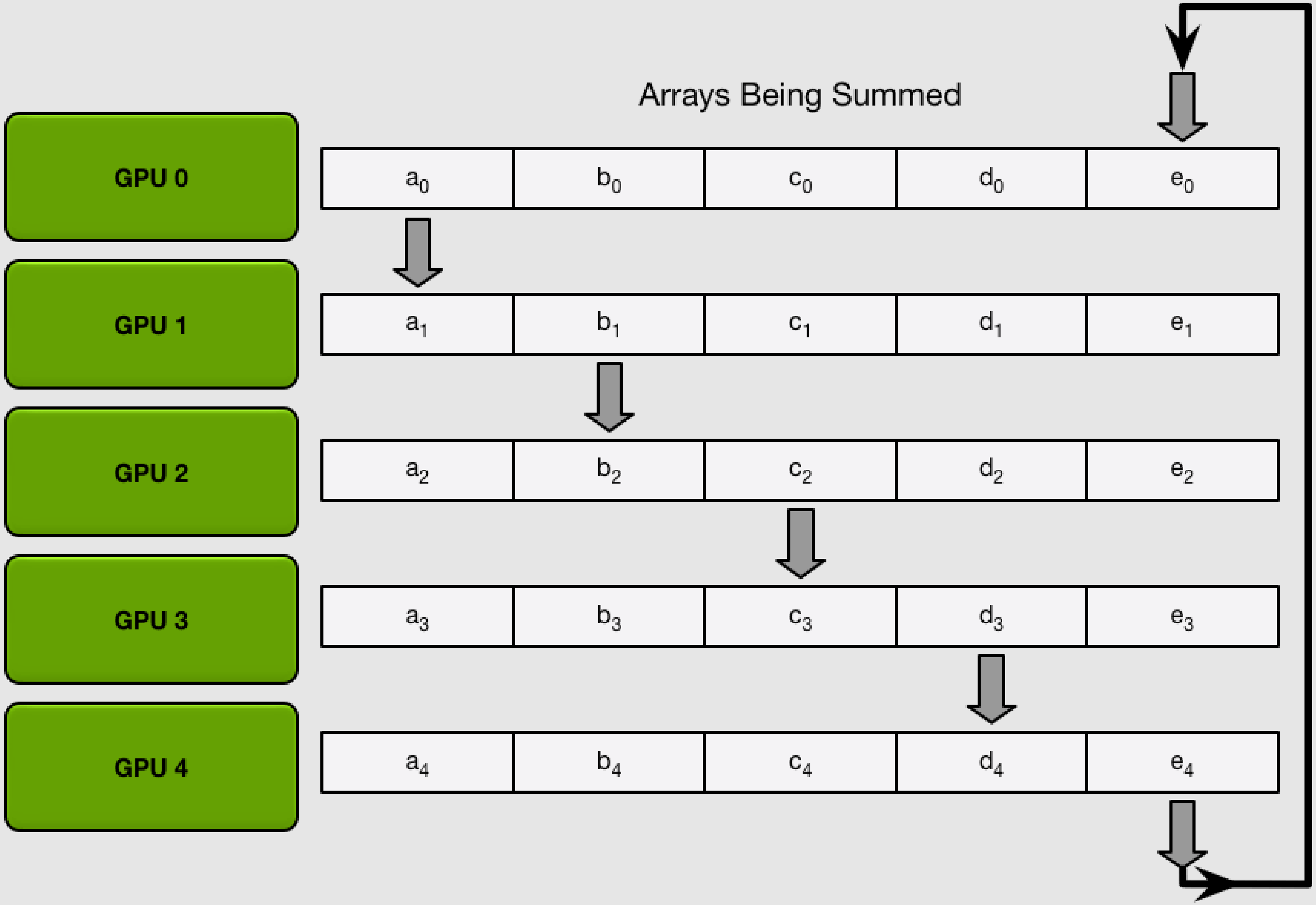


Bringing HPC Techniques to Deep Learning
http://research.baidu.com/bringing-hpc-techniques-deep-learning/

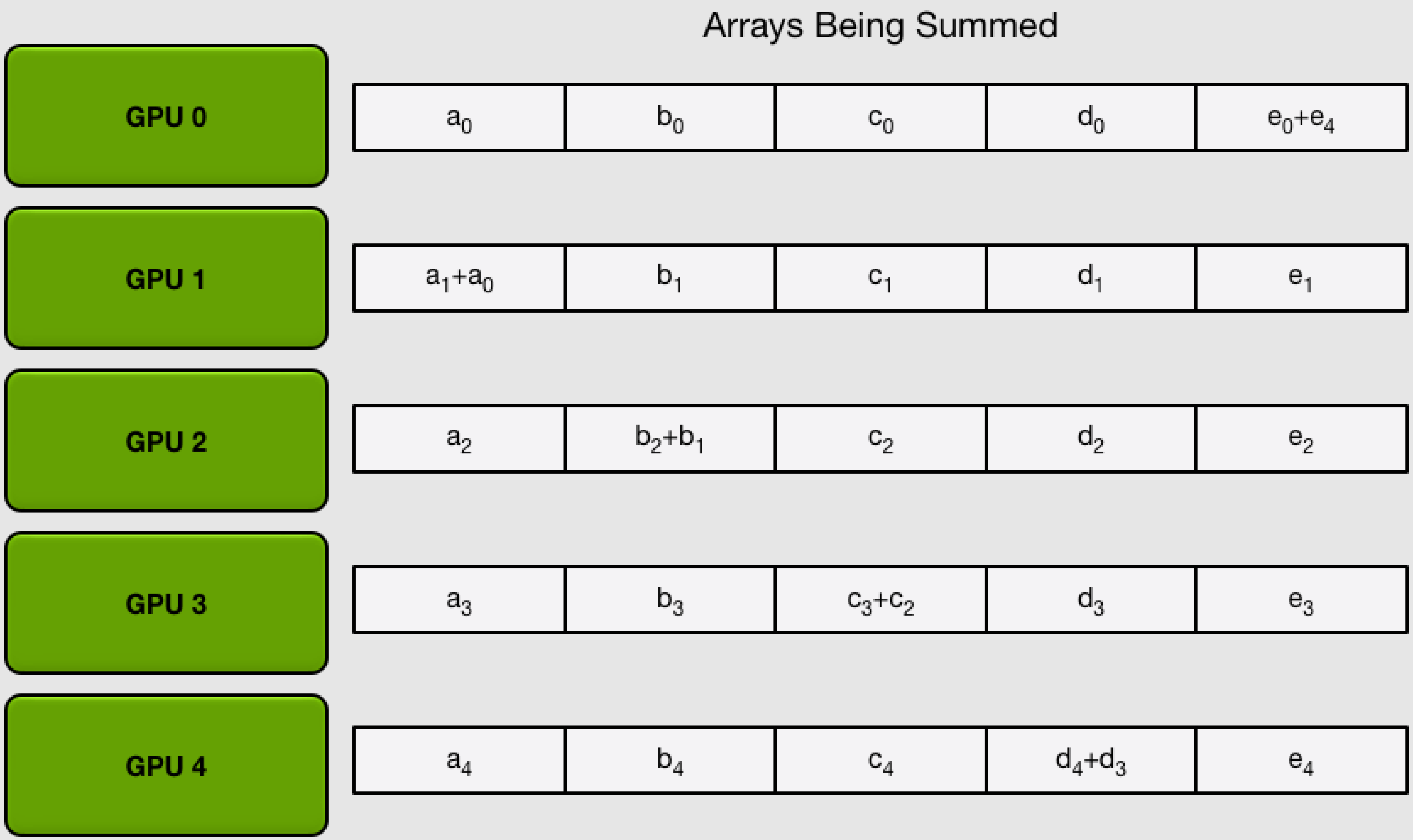

GPU 0
Send
Receive
GPU 1
Send
Receive
GPU 2
Send
Receive
GPU 3
Send
Receive
GPU 4
Send
Receive

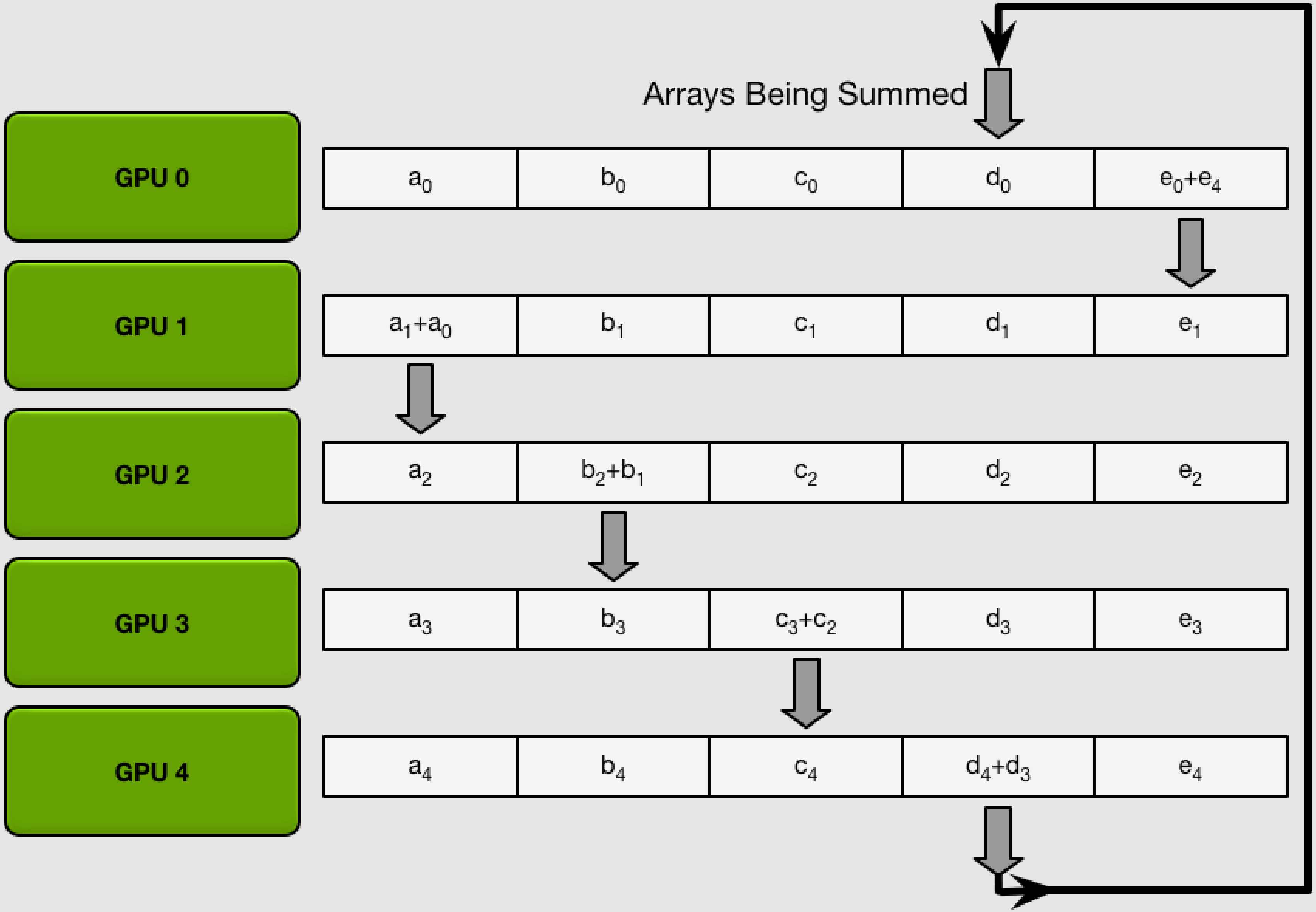
ScatterReduce

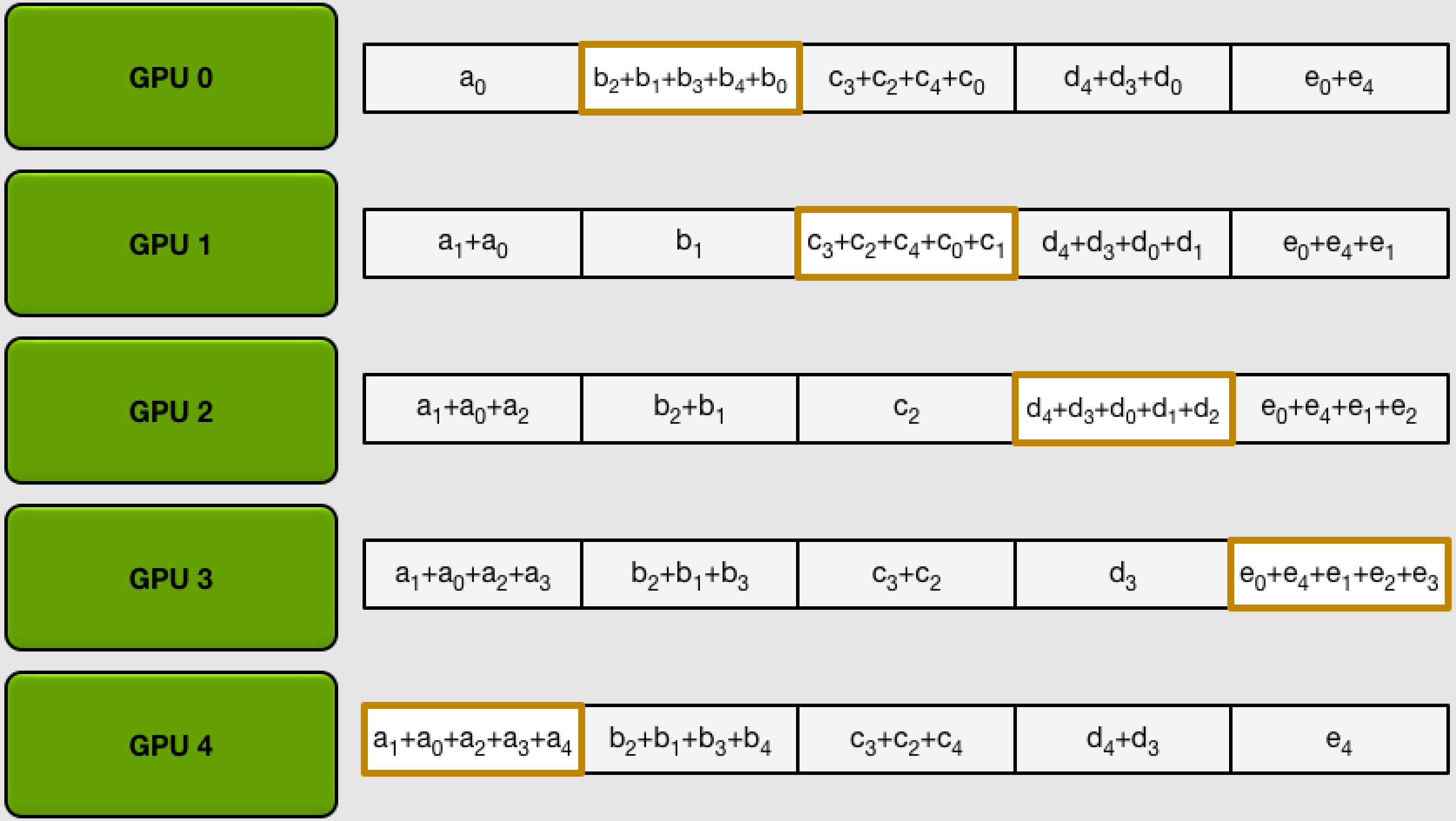




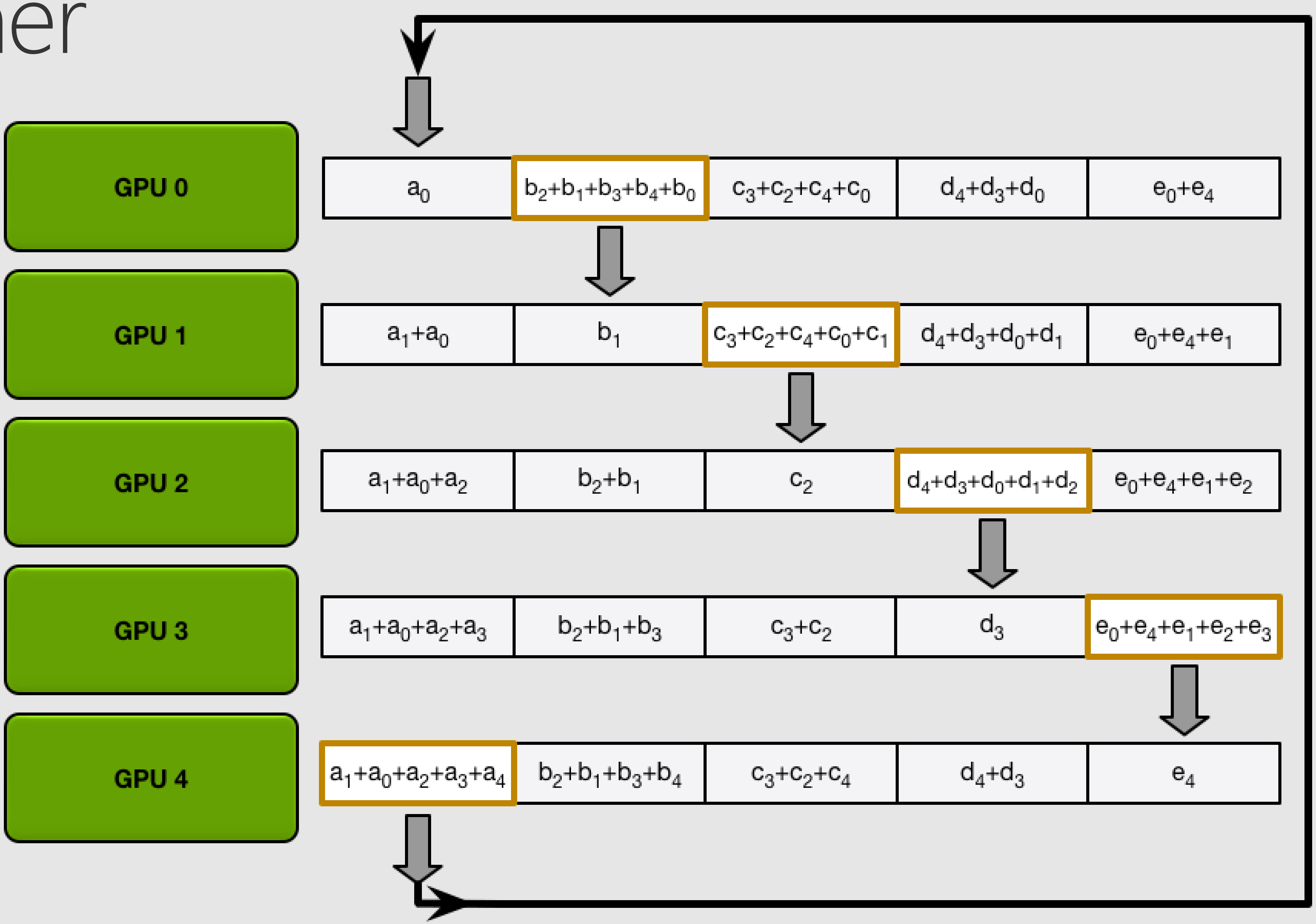
Arrays Being Summed
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a0 b0 c0 d0 e0
a1 b1 c1 d1 e1
a2 b2 c2 d2 e2
a3 b3 c3 d3 e3
a4 b4 c4 d4 e4

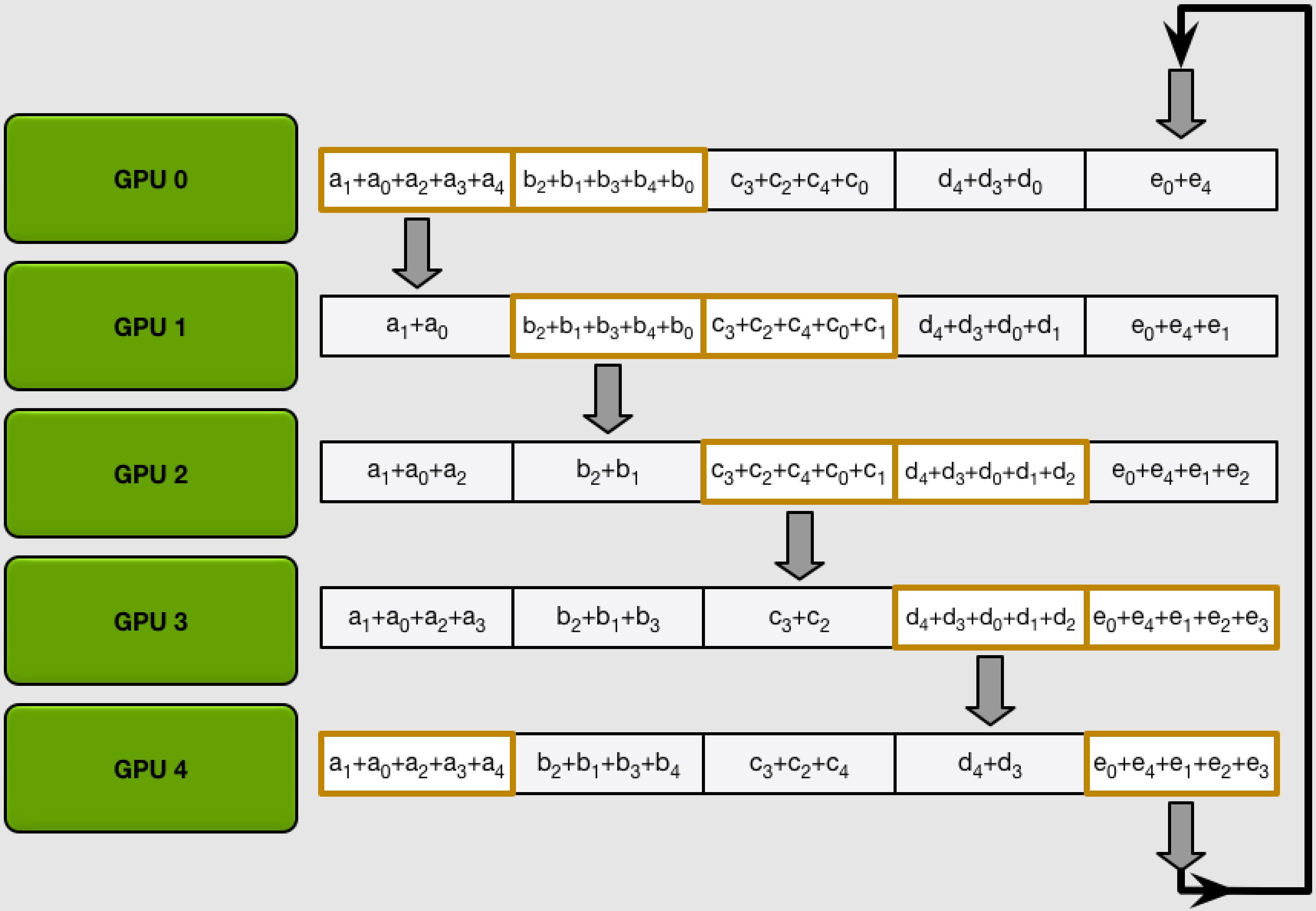

Arrays Being Summed
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a_0 b_0 c_0 d_0 e_0+e_4
a_1+a_0 b_1 c_1 d_1 e_1
a_2 b_2+b_1 c_2 d_2 e_2
a_3 b_3 c_3+c_2 d_3 e_3
a_4 b_4 c_4 d_4+d_3 e_4


Arrays Being Summed
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a_0 b_0 c_0 d_0 e_0+e_4
a_1+a_0 b_1 c_1 d_1 e_1
a_2 b_2+b_1 c_2 d_2 e_2
a_3 b_3 c_3+c_2 d_3 e_3
a_4 b_4 c_4 d_4+d_3 e_4

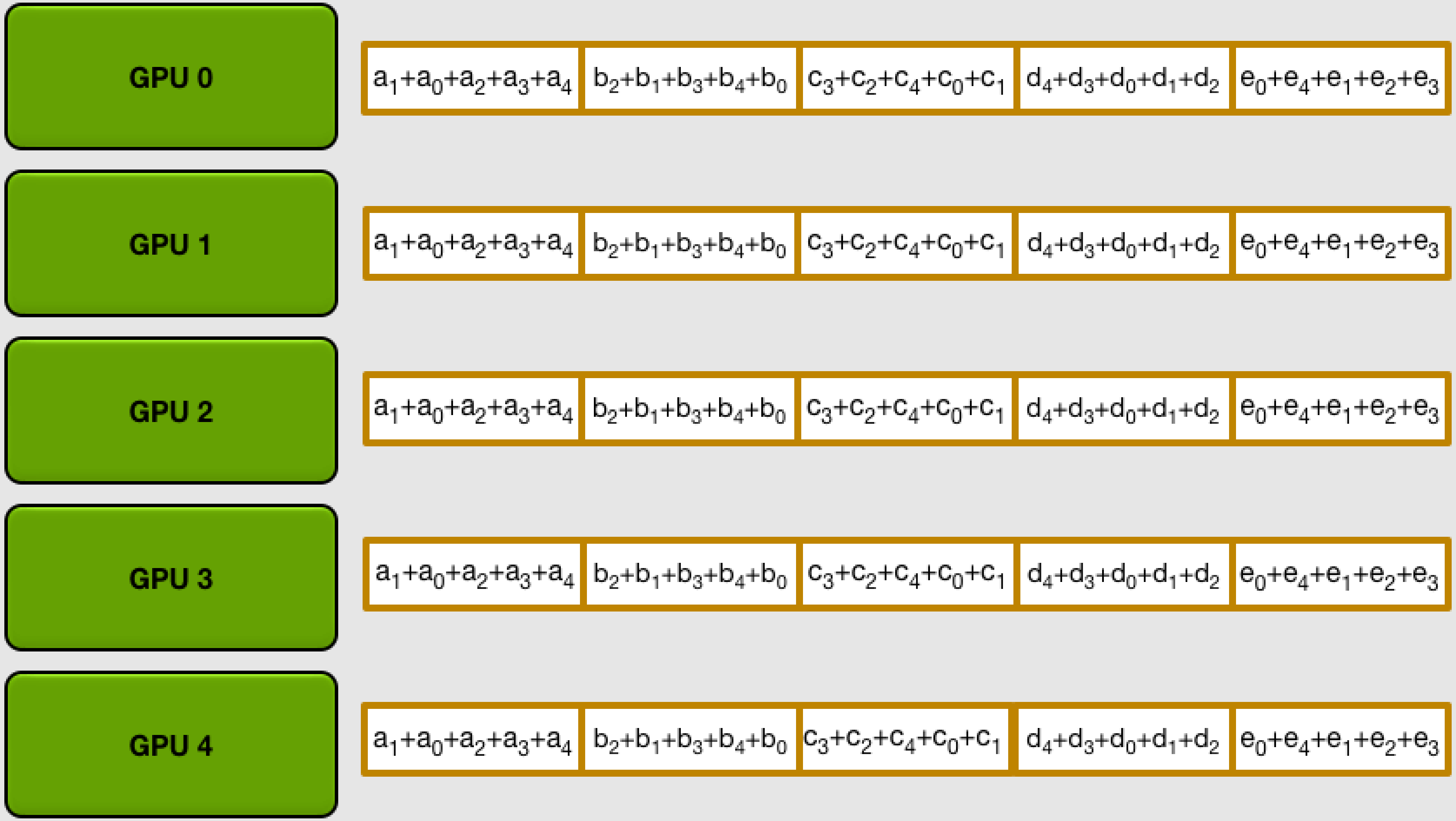

GPU 0
a_0
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0$
$d_4+d_3+d_0$
e_0+e_4
GPU 1
a_1+a_0
b_1
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1$
$e_0+e_4+e_1$
GPU 2
$a_1+a_0+a_2$
b_2+b_1
c_2
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2$
GPU 3
$a_1+a_0+a_2+a_3$
$b_2+b_1+b_3$
c_3+c_2
d_3
$e_0+e_4+e_1+e_2+e_3$
GPU 4
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4$
$c_3+c_2+c_4$
d_4+d_3
e_4

AllGather


GPU 0
a_0
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0$
$d_4+d_3+d_0$
e_0+e_4
GPU 1
a_1+a_0
b_1
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1$
$e_0+e_4+e_1$
GPU 2
$a_1+a_0+a_2$
b_2+b_1
c_2
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2$
GPU 3
$a_1+a_0+a_2+a_3$
$b_2+b_1+b_3$
c_3+c_2
d_3
$e_0+e_4+e_1+e_2+e_3$
GPU 4
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4$
$c_3+c_2+c_4$
d_4+d_3
e_4



GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
a1+a0+a2+a3+a4
b2+b1+b3+b4+b0
c3+c2+c4+c0
d4+d3+d0
e0+e4
a1+a0
b2+b1+b3+b4+b0
c3+c2+c4+c0+c1
d4+d3+d0+d1
e0+e4+e1
a1+a0+a2
b2+b1
c3+c2+c4+c0+c1
d4+d3+d0+d1+d2
e0+e4+e1+e2
a1+a0+a2+a3
b2+b1+b3
c3+c2
d4+d3+d0+d1+d2
e0+e4+e1+e2+e3
a1+a0+a2+a3+a4
b2+b1+b3+b4
c3+c2+c4
d4+d3
e0+e4+e1+e2+e3


GPU 0
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2+e_3$
GPU 1
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2+e_3$
GPU 2
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2+e_3$
GPU 3
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2+e_3$
GPU 4
$a_1+a_0+a_2+a_3+a_4$
$b_2+b_1+b_3+b_4+b_0$
$c_3+c_2+c_4+c_0+c_1$
$d_4+d_3+d_0+d_1+d_2$
$e_0+e_4+e_1+e_2+e_3$

Nuts and Bolts

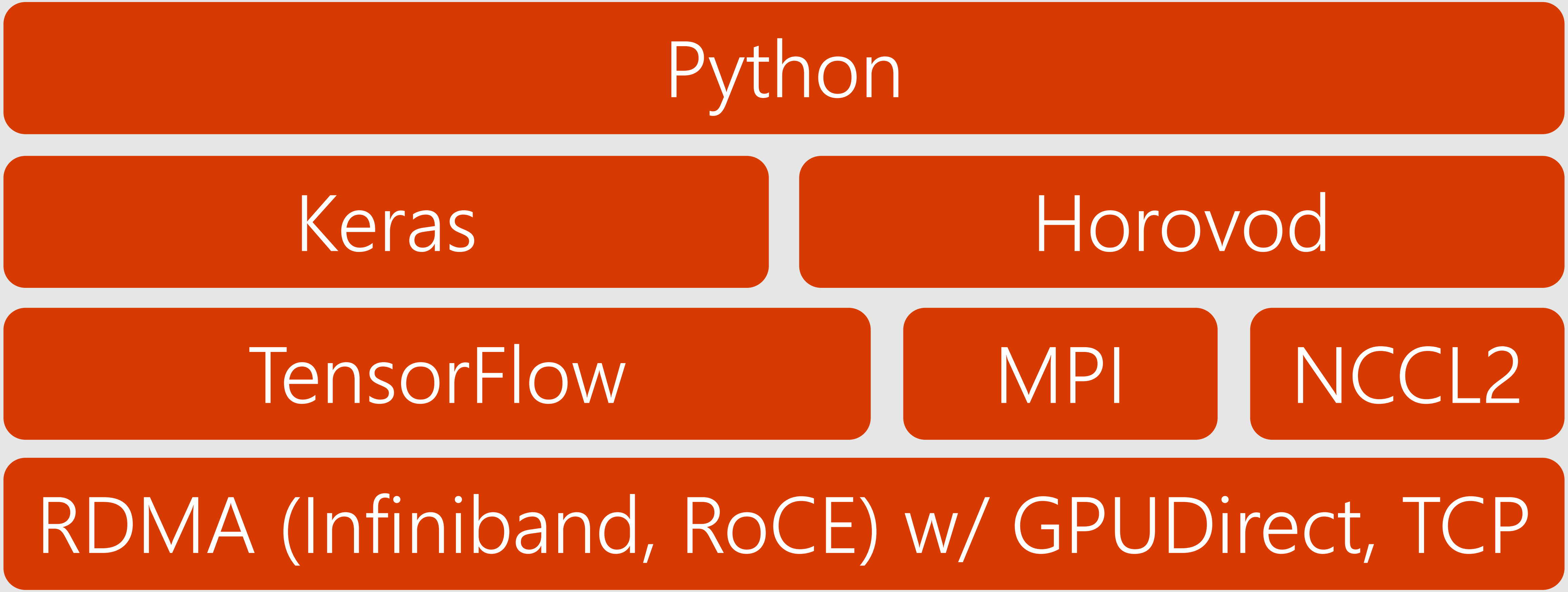
Horovod

- Distributed training framework
- Open Source (Apache 2.0) from Uber
- Implements ScatterReduce + AllGather for Tensorflow
- Uses NCCL2 under the hood with RDMA if available
- ‘TensorFusion’ batching for better latency tolerance
- ‘Timeline’ distributed execution logging viewable with chrome ://tracing
- Very minimal changes required to TensorFlow program
- `pip install horovod` ☺

- Horovod optimizer wraps native optimizer

```
opt = keras.optimizers.Adadelta()
opt = hvd.DistributedOptimizer(opt)
```

- MPI for discovery and co-ordination
- NVIDIA Collective Communications Library for ScatterReduce & AllGather



TensorFusion

- Gradient exchange happens as available by layer
- Deep but narrow networks (e.g. ResNet) have many small tensors
- Latency (especially on Ethernet and SDN stacks) becomes a problem (chatty communications)
- TensorFusion buffers multiple tensors then runs AllReduce on the buffer

Scripting w/ Keras

```
import tensorflow as tf
import horovod.keras as hvd

# Initialize Horovod. Determines cluster size etc...
hvd.init()

# Pin GPU to be used to process local rank (one GPU per
process)
...
config.gpu_options.visible_device_list =
str(hvd.local_rank())
...

# Adjust number of epochs based on number of GPUs.
epochs = int(math.ceil(12.0 / hvd.size()))
...

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
         activation='relu’,
         input_shape=input_shape))
...
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
# Adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

model.compile(
         loss=keras.losses.categorical_crossentropy,
         optimizer=opt,
         metrics=['accuracy'])

# Push initial parameters to all nodes
model.fit(...,
callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallba
ck(0)],
...)
```

Training

- Pre process data or parallelize pipeline (mpi4py)
  Minimize pre-processing in training program
- Shuffle and sample records; effectively distributed bootstrapping
- Broadcast initial weights to all nodes
- Facebook guidance on learn rate
  - Scale linearly with minibatch size
- Google guidance
  - Don't decay learn rate; increase batch size
- Checkpoint/TensorBoard etc. on global rank=1

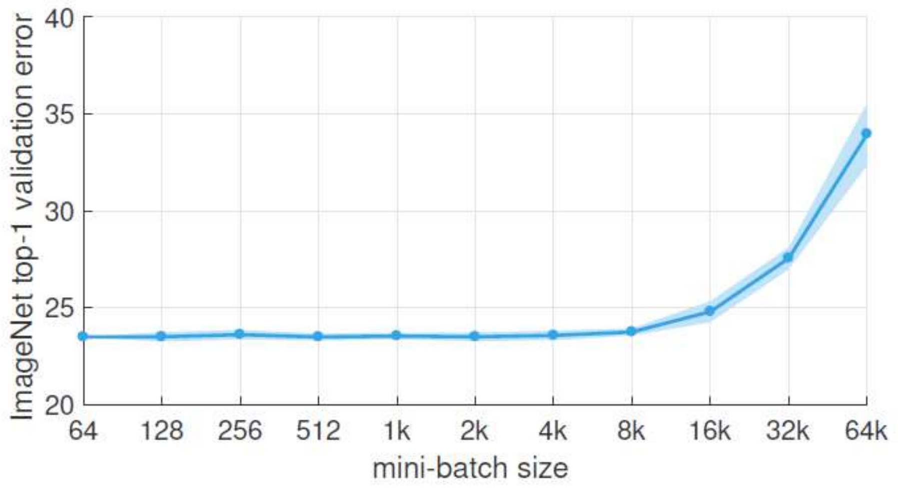


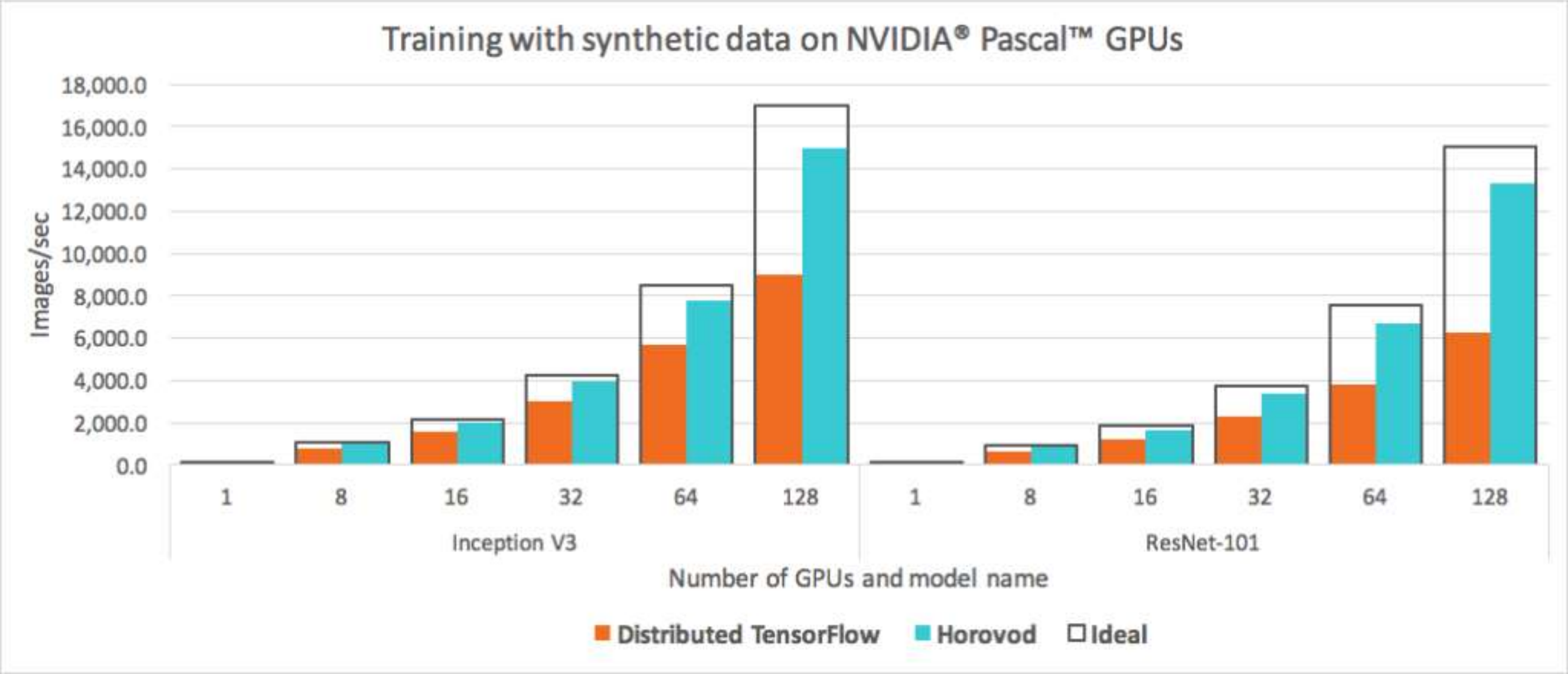
Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
https://arxiv.org/abs/1706.02677 (Facebook Research)

Don't Decay the Learning Rate, Increase the Batch Size
https://arxiv.org/abs/1711.00489 (Google Brain Team)

Demo

Basic RNN Model
Advanced MNIST Model



Horovod: fast and easy distributed deep learning in TensorFlow
https://arxiv.org/abs/1802.05799

Other Frameworks

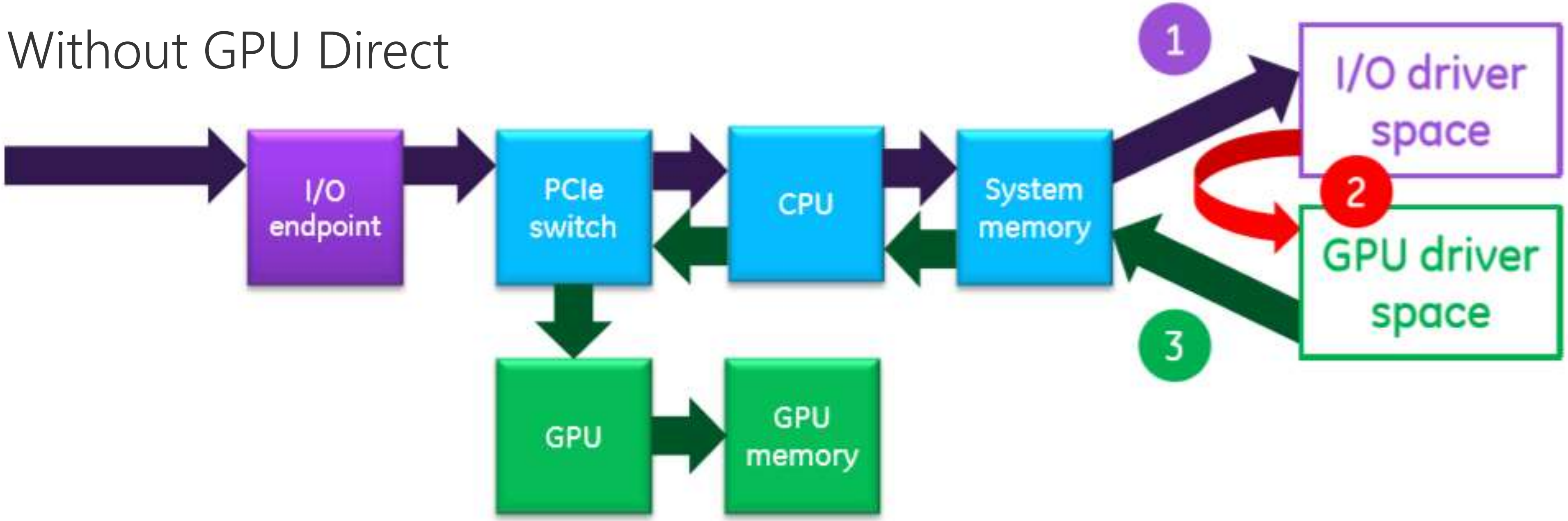
- Tensorflow
  - Baidu All-Reduce
- Caffe
  - Supported via NCCL 2.0
- CNTK
  - Uses NCCL 2.0. Also has quantized gradient exchange (1-bit SGD) which is now (as of last Friday!) also covered by MIT License
- Chainer (as ChainerMN)
  - AllReduce via NCCL 2.0. Does not support FP16 yet.
- MXNet
  - Uses a parameter server. Uses explicit distribution functions (code) within the program. NCCL is supported (to parameter server)
- PyTorch
  - torch.multiprocessing; multi-GPU via shared memory. Super easy. torch.distributed; multi-GPU & multi-node; faster via NCCL MXNet

Theano RIP.

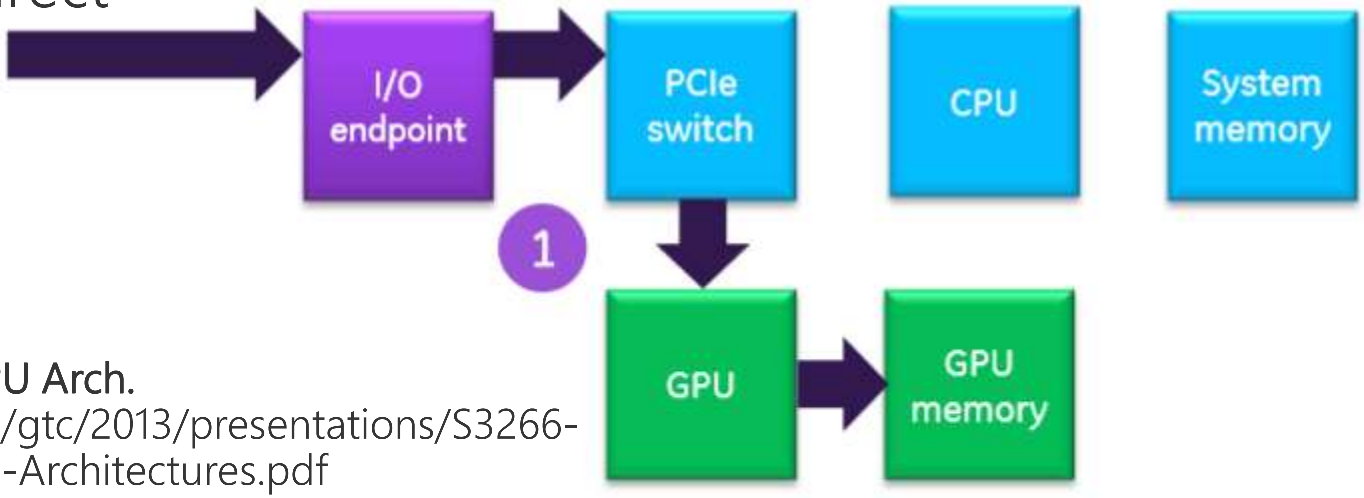
Hardware & Cloud

GPUDirect

Without GPU Direct



With GPU Direct



- Network
- Storage
- GPU Peers

GPUDirect RDMA & Green Multi-GPU Arch.
http://on-demand.gputechconf.com/gtc/2013/presentations/S3266-GPUDirect-RDMA-Green-Multi-GPU-Architectures.pdf

Volta GPU VM: NC_v3

	NC6s_v3	NC12s_v3	NC24s_v3	NC24rs_v3
Cores	6	12	24	24
GPU	1 x V100 GPU	2 x V100 GPU	4 x V100 GPU	4 x V100 GPU
Memory	112 GB	224 GB	448 GB	448 GB
Disk	~700 GB SSD	~1.4 TB SSD	~1.4 TB SSD	~1.4 TB SSD
Network	Azure Network	Azure Network	Azure Network	InfiniBand

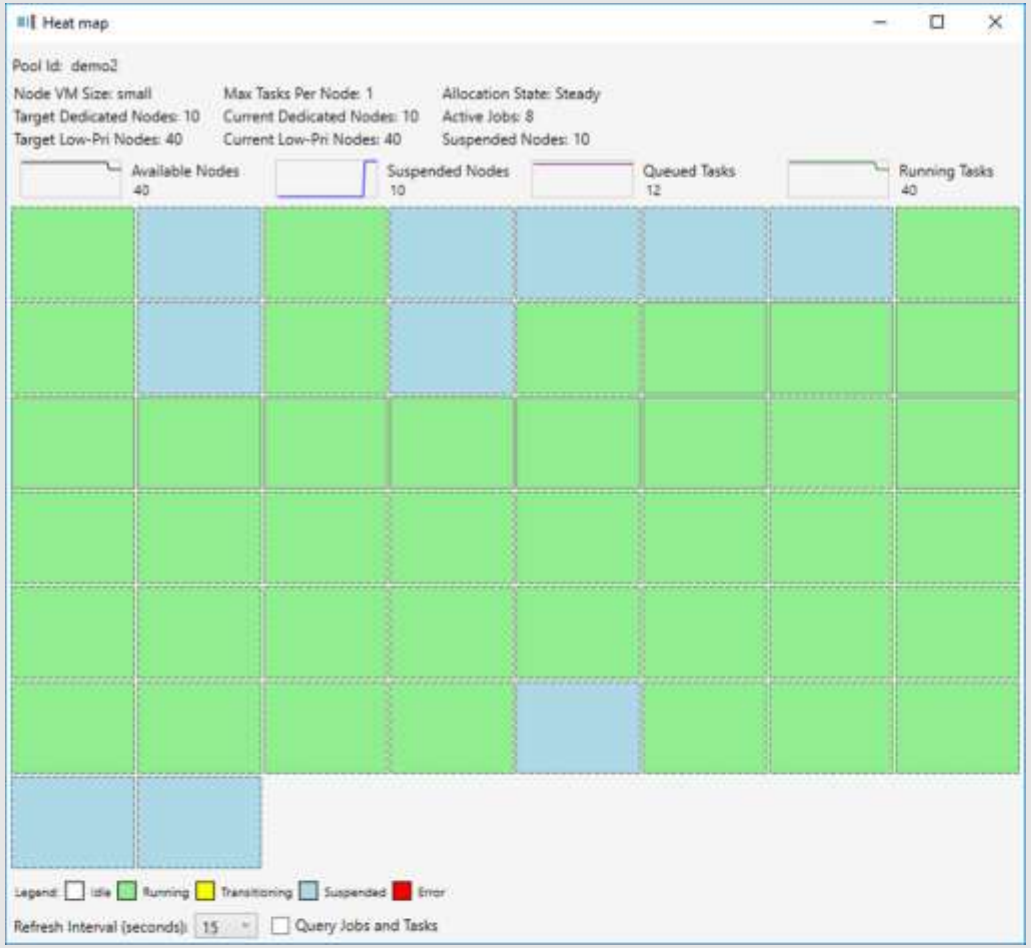
Next-Gen GPU Deep Learning VM: ND

	ND6s	ND12s	ND24s	ND24rs
Cores	6	12	24	24
GPU	1 x P40 GPU	2 x P40 GPU	4 x P40 GPU	4 x P40 GPU
Memory	112 GB	224 GB	448 GB	448 GB
Disk	~700 GB SSD	~1.4 TB SSD	~3 TB SSD	~3 TB SSD
Network	Azure Network	Azure Network	Azure Network	InfiniBand

Spare Capacity VMs

Azure Low Priority VMs, Google Pre-emptible VMs, AWS Spot Instances

- Azure Low Priority VMs
  - 60%-90% discount compared to on-demand price; fixed price
  - Biggest discount applies to NCv3 Series; 4 x V100 Volta GPUs @ $1.22/hr
  - VMs could be pre-empted at any time; using spare and backup capacity
  - VM sizes and regions; availability and pre-emption rate/risk vary
  - Azure batch can requeue your job to another low priority VM and keep track of it
- Issues for DNN Training
  - Losing one node kills whole training job
  - Can checkpoint and restart; run spare capacity for large MPI clusters
  - Separate data preparation pipeline from training; stage training data
  - Use flattened containers rather than VM images for job specific requirements
  - Scale Up before Scale Out



Batch Shipyard

- Make it easier to run Docker apps using Python tooling
- Built on production Batch service and API's
- Main Docker-related capabilities:
  - Deploy Docker engine to nodes
  - Accelerated Docker image deployment at scale via private peer-to-peer distribution
  - Deploy required application images to nodes
  - Can use private registry
- Other capabilities:
  - Deploy GlusterFS for use by pool nodes
  - Install required GPU and RDMA drivers
- Recipes:
  - Specify JSON configuration files
  - Large number of pre-supplied recipes in GitHub; e.g. CNTK, TensorFlow, Caffe

Batch AI; Fully managed ML Clusters

Cloud-scale resource management and task execution

Shipyard project for containers and CLI experience

Specify what program to run with any parameters, where to run it, and how many instances in parallel

Just pay for the compute you use

Standard and low priority VMs

https://azure.microsoft.com/services/batch/

Train From Configuration or Code

Python, C#, Java, REST APIs

Azure Command Line Interface (CLI)

JSON parameter files

Data Storage Options

Local disk

Azure Files (stream with CLI)

Azure Blob with FUSE (stream with CLI)

Managed NFS (mount over SSH)

Parallel File Server: Lustre, Gluster, BeeGFS…

Mount remote volume to VM and into container

Transfer to/from blob, FUSE, NFS, etc

Get data into fastest store first; system RAM > disk

Final Tips and Tricks

Tips for best price/performance

- Use Linux.
- Every second at < 100% utilization (network/GPU) = $$$
- Scale Up, Then Out (GPUs), Then Out (Nodes)
  - Stay on a single machine for as long as you possibly can
- Use the largest possible batch size; but no larger
- Only as fast as your fastest GPU and fastest interconnect
  - Don't mix GPUs. Stay on a single machine if possible
  - Beware anything which could make different minibatches take different times to process
- Use a job/cluster management tool. Azure Batch, Spark
  - Allows easier use of low priority VMs

Useful Links & Papers

- http://github.com/cauldnz/highscale-dnn-training
- NBCL (Ohio State) Hot Interconnects '17 Tutorial http://www.hoti.org/tutorials/HOTI25_Tutorial_1b.pdf
- Good discussion of Asynchronous SGD https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/
- Automated Model-Parallel Distribution http://on-demand.gputechconf.com/gtc/2017/presentation/s7724-minjie-wong-tofu-parallelizing-deep-learning.pdf