Advancing Communication Technologies and System Architecture to Maximize Performance and Scalability of GPU Accelerated Systems

Craig Tierney
AGENDA

The connection between HPC and Deep Learning

GPUDirect technologies
NVLINK-enabled multi-GPU systems
GPUDirect Async

NCCL 2.0
CONVERGENCE OF HPC AND AI
HPC AND AI

HPC

AI
AI REVOLUTIONIZING OUR WORLD

- Search, Assistants, Translation, Recommendations, Shopping, Photos...
- Detect, Diagnose and Treat Diseases
- Powering Breakthroughs in Agriculture, Manufacturing, EDA
10+ YEARS OF GPU COMPUTING

- 2006: CUDA Launched
- 2008: World's First GPU Top500 System
- 2009: Fermi: World's First HPC GPU
- 2010: Oak Ridge Deploys World's Fastest Supercomputer w/ GPUs
- 2012: Stanford Builds AI Machine using GPUs
- 2014: World's First Atomic Model of HIV Capsid
- 2015: Google Outperforms Humans in ImageNet
- 2016: World's First 3-D Mapping of Human Genome
- 2016: GPU-Trained AI Machine Beats World Champion in Go
WHAT HPC NEEDS
**ROAD TO EXASCALE**

Volta to Fuel Most Powerful US Supercomputers

<table>
<thead>
<tr>
<th>cuFFT</th>
<th>Physics (QUDA)</th>
<th>Seismic (RTM)</th>
<th>STREAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8</td>
<td>1.5</td>
<td>1.6</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Summit Supercomputer
- 200+ PetaFlops
- ~3,400 Nodes
- 10 Megawatts

System Config Info: 2X Xeon E5-2690 v4, 2.6GHz, w/ 1X Tesla P100 or V100. V100 measured on pre-production hardware.
WHAT AI NEEDS
NEURAL NETWORK COMPLEXITY IS EXPLODING
To Tackle Increasingly Complex Challenges

- **2015 - Microsoft ResNet**
  - 7 ExaFLOPS
  - 60 Million Parameters
  
- **2016 - Baidu Deep Speech 2**
  - 20 ExaFLOPS
  - 300 Million Parameters
  
- **2017 - Google Neural Machine Translation**
  - 100 ExaFLOPS
  - 8700 Million Parameters

**7 ExaFLOPS**
- 60 Million Parameters

**20 ExaFLOPS**
- 300 Million Parameters

**100 ExaFLOPS**
- 8700 Million Parameters
POWERING THE DEEP LEARNING ECOSYSTEM

NVIDIA SDK accelerates every major framework

COMPUTER VISION
- OBJECT DETECTION
- IMAGE CLASSIFICATION

SPEECH & AUDIO
- VOICE RECOGNITION
- LANGUAGE TRANSLATION

NATURAL LANGUAGE PROCESSING
- RECOMMENDATION ENGINES
- SENTIMENT ANALYSIS

DEEP LEARNING FRAMEWORKS
- Caffe
- DL4J
- Mocha.jl
- Keras
- CNTK
- mxnet
- Purine
- Minerva
- OpenDeep
- Pylearn2
- Theano

NVIDIA DEEP LEARNING SDK
- cuDNN
- TensorRT
- DeepStream SDK
- cuBLAS
- cuSPARSE
- NCCL

developer.nvidia.com/deep-learning-software
WHAT DOES HPC HAVE TO DO WITH AI?
EVERYTHING!!!
ONE PLATFORM BUILT FOR BOTH
DATA SCIENCE & COMPUTATIONAL SCIENCE

Tesla Platform

Accelerating AI

Accelerating HPC
HPC AND AI

HPC

AI
HPC + AI
HPC + AI?
FACTORS DRIVING INNOVATION IN HPC

End of Dennard Scaling places a cap on single threaded performance

Increasing application performance will require fine grain parallel code with significant computational intensity

AI and Data Science emerging as important new components of scientific discovery

Dramatic improvements in accuracy, completeness and response time yield increased insight from huge volumes of data

Cloud based usage models, in-situ execution and visualization emerging as new workflows critical to the science process and productivity

Tight coupling of interactive simulation, visualization, data analysis/Al
THE EX FACTOR IN THE EXASCALE ERA
Multiple EXperiments Coming or Upgrading In the Next 10 Years

- How will SKA1 be better than today's best radio telescopes?
- Personal Genomics
- 15 TB/Day
- 10X Increase in Data Volume
- Exabyte/Day
- 30X Increase in power

ITER TOKAMAK
- A Giant: 25,000 tons
- 10X the core of the Sun: 150 kelvins
- Fusion energy: 500 MW

10X Increase in power

High Luminosity LHC

30X Increase in power
THE POTENTIAL OF EXASCALE HPC + AI

HPC

+40 years of Algorithms based on first principles theory
Proven statistical models for accurate results in multiple science domains

AI

New methods to improve predictive accuracy, insight into new phenomena and response time with previously unmanageable data sets

Commercially viable fusion energy
Understanding the Origins of the Universe
Clinically Viable Precision Medicine
Improve/validate the Standard Model of Physics
Climate/Weather forecasts with ultra high fidelity
Organizing HPC + AI Convergence

### Transformation
HPC + AI couple simulation with live data in real time detection/control system.

Experimental/simulated data is used to train a NN that is used to replace all or significant runtime portions of a conventional simulation. The NN is improved continuously as new simulated / live data is acquired.

### Augmentation
HPC + AI combined to improve simulation time to science > orders of magnitude.

Experimental/simulated data is used to train a NN that is used to replace all or significant runtime portions of a conventional simulation. The NN is improved continuously as new simulated / live data is acquired.

### Modulation
HPC + AI combined to reduce the number of runs needed for a parameter sweep.

Experimental/simulated data used to train a NN which steers simulation/experiment between runs. The steering NN can be trained continuously as new simulated / live data is acquired.

---

**Potential for Breakthroughs in Scientific Insight**
Despite the latest development in computational power, there is still a large gap in linking relativistic theoretical models to observations. 

Max Plank Institute

Background
The aLIGO (Advanced Laser Interferometer Gravitational Wave Observatory) experiment successfully discovered signals proving Einstein’s theory of General Relativity and the existence of cosmic Gravitational Waves. While this discovery was by itself extraordinary it is seen to be highly desirable to combine multiple observational data sources to obtain a richer understanding of the phenomena.

Challenge
The initial a LIGO discoveries were successfully completed using classic data analytics. The processing pipeline used hundreds of CPU’s where the bulk of the detection processing was done offline. Here the latency is far outside the range needed to activate resources, such as the Large Synaptic Space survey Telescope (LSST) which observe phenomena in the electromagnetic spectrum in time to “see” what aLIGO can “hear”.

Solution
A DNN was developed and trained using a data set derived from the CACTUS simulation using the Enistein Toolkit. The DNN was shown to produce better accuracy with latencies 1000x better than the original CPU based waveform detection.

Impact
Faster and more accurate detection of gravitational waves with the potential to steer other observational data sources.
Background
Grand challenge of fusion energy offers mankind changing opportunity to provide clean, safe energy for millions of years. ITER is a $25B international investment in a fusion reactor.

Challenge
Fusion is highly sensitive, any disruption to conditions can cause reaction to stop suddenly. Challenge is to predict when a disruption will occur to prevent damage to ITER and to steer the reaction to continue to produce power. Traditional simulation and ML approaches don’t deliver accurate enough results.

Solution
DL network called FRNN using Theano exceeds today’s best accuracy results. It scales to 200 Tesla K20s, and with more GPUs, can deliver higher accuracy. Goal is to reach 95% accuracy.

Impact
Vision is to operate ITER with FRNN, operating and steering experiments in real-time to minimize damage and down-time.
Background
Developing a new drug costs $2.5B and takes 10-15 years. Quantum chemistry (QC) simulations are important to accurately screen millions of potential drugs to a few most promising drug candidates.

Challenge
QC simulation is computationally expensive so researchers use approximations, compromising on accuracy. To screen 10M drug candidates, it takes 5 years to compute on CPUs.

Solution
Researchers at the University of Florida and the University of North Carolina leveraged GPU deep learning to develop ANAKIN-ME, to reproduce molecular energy surfaces with super speed (microseconds versus several minutes), extremely high (DFT) accuracy, and at 1-10/millionths of the cost of current computational methods.

Impact
Faster, more accurate screening at far lower cost

Background
Unexpected fog can cause an airport to cancel or delay flights, sometimes having global effects in flight planning.

Challenge
While the weather forecasting model at MeteoSwiss work at a 2km x 2km resolution, runways at Zurich airport is less than 2km. So human forecasters sift through huge simulated data with 40 parameters, like wind, pressure, temperature, to predict visibility at the airport.

Solution
MeteoSwiss is investigating the use of deep learning to forecast type of fog and visibility at sub-km scale at Zurich airport.

Impact
TRADITIONAL HPC METHOD

Traditional HPC Systems

Large Scale Simulation

Scalable Data Analytics and Visualization
EVOLUTION OF METHOD

- Traditional HPC Systems
- Large Scale Simulation
- Artificial Intelligence/Deep Learning
- Scalable Data Analytics
CONVERGED EXASCALE SYSTEM

- Traditional HPC System
- Large Scale Simulation
- Scalable Data Analytics/Machine Learning
- Artificial Intelligence/Deep Learning

Concept plagiarized and slightly modified from Rick Stevens CANDLE overview
INTRODUCTION TO GPUDIRECT TECHNOLOGIES
GPUDIRECT FAMILY

Technologies, enabling products !!!

GPUDIRECT SHARED GPU-SYSMEM
GPU pinned memory shared with other RDMA capable devices
Avoids intermediate copies

GPUDIRECT P2P
Accelerated GPU-GPU memory copies
Inter-GPU direct load/store access

GPUDIRECT RDMA
Direct GPU to 3rd party device transfers
E.g. direct I/O, optimized inter-node communication

GPUDIRECT ASYNC
Direct GPU to 3rd party device synchronizations
E.g. optimized inter-node communication

GPUDIRECT

scopes

- GPUDirect P2P → data
  - Intra-node
  - GPUs both master and slave
  - Over PCIe or NVLink
- GPUDirect RDMA → data
  - Inter-node
  - GPU slave, 3rd party device master
  - Over PCIe
- GPUDirect Async → control
  - GPU & 3rd party device, master & slave
  - Over PCIe
NVLINK-enabled Multi-GPU servers
NVIDIA DGX-1
AI Supercomputer-in-a-Box

170 TFLOPS | 8x Tesla P100 16GB | NVLink Hybrid Cube Mesh
2x Xeon | 8 TB RAID 0 | Quad IB 100Gbps, Dual 10GbE | 3U – 3200W
NVIDIA DGX-1 VOLTA

960 TFLOPS | 8x Tesla V100 16GB | 300 GB/s NVLink Hybrid Cube Mesh
2x Xeon | 8 TB RAID 0 | Quad IB 100Gbps, Dual 10GbE | 3U — 3200W
DGX-1 SYSTEM TOPOLOGY

GPU - CPU link:
- PCIe
- 12.5+12.5 GB/s eff BW

GPUDirect P2P:
- GPU - GPU link is NVLink
- Cube mesh topology
- not all-to-all

GPUDirect RDMA:
- GPU - NIC link is PCIe
IBM MINSKY

2 POWER8 with NVLink
4 NVIDIA Tesla P100 GPUs
256 GB System Memory
2 SSD storage devices
High-speed interconnect: IB or Ethernet

Optional:
Up to 1 TB System Memory
PCIe attached NVMe storage
IBM MINSKY SYSTEM TOPOLOGY

GPU - CPU link:
2x NVLINK
40+40 GB/s raw BW

GPUDirect P2P:
GPU - GPU link is 2x NVLink
Two cliques topology

GPUDirect RDMA:
Not supported
GPUDIRECT AND MULTI-GPU SYSTEMS
THE CASE OF DGX-1
HOW TO’S

Device topology, link type and capabilities

- GPUa - GPUb link: P2P over NVLINK vs PCIe, speed, etc
- Same for CPU - GPU link: NVLINK or PCIe
- Same for NIC - GPU link (HWLOC)

Select an optimized GPU/CPU/NIC combination in MPI runs

Enable GPUDirect RDMA
CUDA LINK CAPABILITIES

basic info, GPU-GPU links only

// CUDA driver API
typedef enum CUdevice_P2PAttribute_enum {
  CU_DEVICE_P2P_ATTRIBUTE_PERFORMANCE_RANK = 0x01,
  CU_DEVICE_P2P_ATTRIBUTE_ACCESS_SUPPORTED = 0x02,
  CU_DEVICE_P2P_ATTRIBUTE_NATIVE_ATOMIC_SUPPORTED = 0x03
} CUdevice_P2PAttribute;
cuDeviceGetP2PAttribute(int* value, CUdevice_P2PAttribute attrib, CUdevice srcDevice, CUdevice dstDevice)

// CUDA runtime API
cudaDeviceGetP2PAttribute(int *value, enum cudaDeviceP2PAttr attr, int srcDevice, int dstDevice)

A relative value indicating the performance of the link between two GPUs (NVLINK ranks higher than PCIe).
Can do remote native atomics in GPU kernels.
GPUDIRECT P2P: NVLINK VS PCIe

NVLINK transparently picked if available

cudaSetDevice(0);
cudaMalloc(&buf0, size);
cudaCanAccessPeer (&access, 0, 1);
assert(access == 1);
cudaEnablePeerAccess (1, 0);
cudaSetDevice(1);
cudaMalloc(&buf1, size);
...
cudaSetDevice (0);
cudaMemcpy (buf0, buf1, size, cudaMemcpyDefault);

Note: some GPUs are not connected e.g. GPU0-GPU7
Note2: others have multiple potential link (NVLINK and PCIe) but cannot use both at the same time!!!
MULTI GPU RUNS ON DGX-1

Select best GPU/CPU/NIC for each MPI rank

Create wrapper script

Use local MPI rank (MPI impl dependent)

Don’t use CUDA_VISIBLE_DEVICES, hurts P2P!!!

Environment variables to pass selection down to MPI and app

In application
cudaSetDevice("USE_GPU")

Run wrapper script

```bash
$ cat wrapper.sh
if [ ! -z $OMPI_COMM_WORLD_LOCAL_RANK ]; then
  lrank=$OMPI_COMM_WORLD_LOCAL_RANK
elif [ ! -z $MV2_COMM_WORLD_LOCAL_RANK ]; then
  lrank=$MV2_COMM_WORLD_LOCAL_RANK
fi
if (( $lrank > 7 )); then echo "too many ranks"; exit; fi
case ${HOSTNAME} in
  *dgx*)
    USE_GPU=$((2*($lrank%4)+$lrank/4)) # 0,2,4,6,1,3,5,7
    export USE_SOCKET=$((($USE_GPU/4)) # 0,0,1,1,0,0,1,1
    HCA=mlx5_$(($USE_GPU/2)) # 0,1,2,3,0,1,2,3
    export OMPI_MCA_btl_openib_if_include=${HCA}
    export MV2_IBA_HCA=${HCA}
    export USE_GPU;;
  ...
esac
numactl --cpunodebind=${USE_SOCKET} -l $@

$ mpirun -np N wrapper.sh myapp param1 param2 ...
```
NVML¹ NVLINK
Link discovery and info APIs

nvmlDeviceGetNvLinkVersion(nvmlDevice_t device, unsigned int link, unsigned int *version)
nvmlDeviceGetNvLinkState(nvmlDevice_t device, unsigned int link, nvmlEnableState_t *isActive)
nvmlDeviceGetNvLinkCapability(nvmlDevice_t device, unsigned int link, nvmlNvLinkCapability_t capability, unsigned int *capResult)
nvmlDeviceGetNvLinkRemotePciInfo(nvmlDevice_t device, unsigned int link, nvmlPciInfo_t *pci)

nvmlDevice separate from CUDA
gpu id’s (all devices vs
CUDA_VISIBLE_DEVICES)

NVML_NVLINK_MAX_LINKS=6

See later for capabilities
domain:bus:device.function PCI
identifier of device on the other
side of the link, can be socket
PCIe bridge (IBM POWER8)

¹ http://docs.nvidia.com/deploy/nvml-api/
NVLINK CAPABILITIES
On DGX-1

nvidia-smi nvlink -l <GPU id> -c

typedef enum nvmlNvLinkCapability_enum {
    NVML_NVLINK_CAP_P2P_SUPPORTED = 0,
    NVML_NVLINK_CAP_SYSMEM_ACCESS = 1,
    NVML_NVLINK_CAP_P2P_ATOMICS = 2,
    NVML_NVLINK_CAP_SYSMEM_ATOMICS = 3,
    NVML_NVLINK_CAP_SLI_BRIDGE = 4,
    NVML_NVLINK_CAP_VALID = 5,
} nvmlNvLinkCapability_t;

Link 0, P2P is supported: true
Link 0, Access to system memory supported: true
Link 0, P2P atomics supported: true
Link 0, System memory atomics supported: false
Link 0, SLI is supported: false
Link 0, Link is supported: false
Link 1, P2P is supported: true
Link 1, Access to system memory supported: true
Link 1, P2P atomics supported: true
Link 1, System memory atomics supported: false
Link 1, SLI is supported: false
Link 1, Link is supported: false
Link 2, P2P is supported: true
Link 2, Access to system memory supported: true
Link 2, P2P atomics supported: true
Link 2, System memory atomics supported: false
Link 2, SLI is supported: false
Link 2, Link is supported: false
Link 3, P2P is supported: true
Link 3, Access to system memory supported: true
Link 3, P2P atomics supported: true
Link 3, System memory atomics supported: false
Link 3, SLI is supported: false
Link 3, Link is supported: false
NVLINK COUNTERS

On DGX-1

Per GPU (-i 0), per link (-l <0..3>)

Two sets of counters (-g <0|1>)

Per set counter types: cycles, packets, bytes (-sc xyz)

Reset individually (-r <0|1>)

drossetti@  12:34 (174) samples> nvidia-smi nvlink -i 0 -r 1
drossetti@  12:34 (175) samples> nvidia-smi nvlink -i 0 -g 1
    Link 0: Rx1: 0 KBytes, Tx1: 0 KBytes
    Link 1: Rx1: 0 KBytes, Tx1: 0 KBytes
    Link 2: Rx1: 0 KBytes, Tx1: 0 KBytes
    Link 3: Rx1: 0 KBytes, Tx1: 0 KBytes
drossetti@  12:35 (176) samples> bin/x86_64/linux/release/p2pBandwidthLatencyTest
drossetti@  12:35 (177) samples> nvidia-smi nvlink -i 0 -g 1
    Link 0: Rx1: 600320 KBytes, Tx1: 600320 KBytes
    Link 1: Rx1: 600320 KBytes, Tx1: 600320 KBytes
    Link 2: Rx1: 600320 KBytes, Tx1: 600320 KBytes
    Link 3: Rx1: 600320 KBytes, Tx1: 600320 KBytes
NVML TOPOLOGY
GPU-GPU & GPU-CPU topology query APIs

nvmlDeviceGetTopologyNearestGpus(nvmlDevice_t device, nvmlGpuTopologyLevel_t level, unsigned int* count, nvmlDevice_t* deviceArray)
	nvmlDeviceGetTopologyCommonAncestor(nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuTopologyLevel_t* pathInfo)
	nvmlSystemGetTopologyGpuSet(unsigned int cpuNumber, unsigned int* count, nvmlDevice_t* deviceArray)

nvmlDeviceGetCpuAffinity(nvmlDevice_t device, unsigned int cpuSetSize, unsigned long *cpuSet);

1 http://docs.nvidia.com/deploy/nvml-api/group__nvmlDeviceQueries.html
### SYSTEM TOPOLOGY

**On DGX-1**

```bash
$ nvidia-smi topo -m
```

<table>
<thead>
<tr>
<th>GPU0</th>
<th>GPU1</th>
<th>GPU2</th>
<th>GPU3</th>
<th>GPU4</th>
<th>GPU5</th>
<th>GPU6</th>
<th>GPU7</th>
<th>mlx5_0</th>
<th>mlx5_2</th>
<th>mlx5_1</th>
<th>mlx5_3</th>
<th>CPU Affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>NV1</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>NV1</td>
<td>X</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>PIX</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>SOC</td>
<td>SOC</td>
<td>NV1</td>
<td>NV1</td>
<td>NV1</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>mlx5_0</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>mlx5_2</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>PIX</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>mlx5_1</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>mlx5_3</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>X</td>
<td>20-39,60-79</td>
</tr>
</tbody>
</table>

**Legend:**

- **X** = Self
- **SOC** = Connection traversing PCIe as well as the SMP link between CPU sockets (e.g. QPI)
- **PHB** = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
- **PIX** = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
- **N#** = Connection traversing a single PCIe switch
- **N#** = Connection traversing a bonded set of # NVLinks
# SYSTEM TOPOLOGY

On DGX-1, PCIe only

$ nvidia-smi topo -mp

<table>
<thead>
<tr>
<th></th>
<th>GPU0</th>
<th>GPU1</th>
<th>GPU2</th>
<th>GPU3</th>
<th>GPU4</th>
<th>GPU5</th>
<th>GPU6</th>
<th>GPU7</th>
<th>mlx5_0</th>
<th>mlx5_2</th>
<th>mlx5_1</th>
<th>mlx5_3</th>
<th>CPU Affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0</td>
<td>K</td>
<td>PIX</td>
<td>PEB</td>
<td>PEB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIX</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>GPU1</td>
<td>X</td>
<td>PEB</td>
<td>PEB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>PIC</td>
<td>SOC</td>
<td>PHB</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>GPU2</td>
<td>PEB</td>
<td>X</td>
<td>PIX</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>SOC</td>
<td>PIC</td>
<td>SOC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>GPU3</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>SOC</td>
<td>PIC</td>
<td>0-19,40-59</td>
</tr>
<tr>
<td>GPU4</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>PHB</td>
<td>PIC</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>GPU5</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>GPU6</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>SOC</td>
<td>PIC</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>GPU7</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PEB</td>
<td>PEB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>PEB</td>
<td>PHB</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>mlx5_0</td>
<td>PIX</td>
<td>PIX</td>
<td>PEB</td>
<td>PEB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td>20-39,60-79</td>
</tr>
<tr>
<td>mlx5_2</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PIC</td>
<td>PIC</td>
<td>PEB</td>
<td>PEB</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td></td>
</tr>
<tr>
<td>mlx5_1</td>
<td>PHB</td>
<td>PHB</td>
<td>PIX</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>X</td>
<td>SOC</td>
<td>PHB</td>
<td></td>
</tr>
<tr>
<td>mlx5_3</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>PHB</td>
<td>PHB</td>
<td>PHB</td>
<td>PHB</td>
<td>PHB</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td>SOC</td>
<td></td>
</tr>
</tbody>
</table>

Legend:

- X = Self
- SOC = Connection traversing PCIe as well as the SMP link between CPU sockets (e.g., QPI)
- PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
- PIX = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
- PIX = Connection traversing a single PCIe switch
GPUDIRECT P2P
DGX-1 P2P PERFORMANCE

p2pBandwidthLatencyTest

in CUDA toolkit samples

Sources:
samples/1.Utilities/p2pBandwidthLatencyTest

Binary:
samples/bin/x86_64/linux/release/p2pBandwidthLatencyTest

| Unidirectional P2P-Enabled Bandwidth Matrix (GB/s) |
|---------------------------------|---------|---------|---------|---------|---------|---------|---------|---------|
| D\0 N | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 502.05 | 18.43 | 18.43 | 18.42 | 18.44 | 10.29 | 10.30 | 10.29 |
| 1 | 18.43 | 500.44 | 18.44 | 18.44 | 10.36 | 18.44 | 10.37 | 10.36 |
| 2 | 18.44 | 18.43 | 499.92 | 18.44 | 10.35 | 10.36 | 18.43 | 10.36 |
| 3 | 18.44 | 18.43 | 18.44 | 499.70 | 10.30 | 10.31 | 10.29 | 18.44 |
| 4 | 18.44 | 10.33 | 10.32 | 10.32 | 499.16 | 18.43 | 18.44 | 18.43 |
| 5 | 10.31 | 18.43 | 10.30 | 10.32 | 18.43 | 499.76 | 18.44 | 18.44 |
| 6 | 10.28 | 10.29 | 18.44 | 10.28 | 18.44 | 18.44 | 498.25 | 18.44 |
| 7 | 10.33 | 10.34 | 10.36 | 18.43 | 18.44 | 18.44 | 18.43 | 500.90 |

| Bidirectional P2P-Enabled Bandwidth Matrix (GB/s) |
|---------------------------------|---------|---------|---------|---------|---------|---------|---------|---------|
| D\D N | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 509.85 | 36.82 | 36.81 | 36.82 | 36.83 | 17.02 | 17.58 | 17.57 |
| 1 | 36.82 | 508.90 | 36.80 | 36.79 | 16.98 | 36.84 | 17.00 | 17.39 |
| 2 | 36.82 | 36.82 | 509.16 | 36.81 | 17.59 | 16.96 | 36.82 | 17.55 |
| 3 | 36.83 | 36.79 | 36.82 | 508.04 | 17.51 | 16.94 | 17.51 | 36.84 |
| 4 | 36.83 | 16.97 | 17.51 | 17.40 | 510.05 | 36.83 | 36.84 | 36.83 |
| 5 | 17.00 | 36.81 | 16.99 | 17.06 | 36.83 | 507.89 | 36.84 | 36.84 |
| 6 | 17.38 | 16.98 | 36.82 | 17.51 | 36.83 | 36.83 | 508.78 | 36.84 |
| 7 | 17.49 | 17.09 | 17.57 | 36.82 | 36.83 | 36.84 | 36.83 | 509.06 |
DGX-1 P2P PERFORMANCE

busGrind

In CUDA toolkit demo suite:

```
/usr/local/cuda-8.0/extras/demo_suite/busGrind -h
```

Usage:  
-<p [0,1]: enable or disable pinned memory tests (default on)
-<u [0,1]: enable or disable unpinned memory tests (default off)
-<e [0,1]: enable or disable p2p enabled memory tests (default on)
-<d [0,1]: enable or disable p2p disabled memory tests (default off)
-a enable all tests
-n disable all tests

```
<table>
<thead>
<tr>
<th>Test Description: Bus bandwidth between pairs of devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2P Bandwidth Matrix (GB/s) - Unidirectional, P2P-Enabled</td>
</tr>
<tr>
<td>D0</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Description: Bus bandwidth between pairs of devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2P Bandwidth Matrix (GB/s) - Bidirectional, P2P-Enabled</td>
</tr>
<tr>
<td>D0</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
```
Intra-node MPI BW

GPU-aware MPI running over GPUDirect P2P
Dual IVB Xeon 2U server (K40 PCIe) vs DGX-1 (P100-nvlink)

~35 GB/sec Bi-dir
~17 GB/sec Uni-dir
GPUDIRECT RDMA
GPUDirect RDMA over RDMA networks
for better network communication latency

For Linux rdma subsystem

open-source nvidia_peer_memory kernel module¹

important bug fix in ver 1.0-3 !!!

enables NVIDIA GPUDirect RDMA on OpenFabrics stack

Multiple vendors

Mellanox²: ConnectX3 to ConnectX-5, IB/RoCE

Chelsio³: T5, iWARP

Others to come

¹ [https://github.com/Mellanox/nv_peer_memory](https://github.com/Mellanox/nv_peer_memory)
³ [http://www.chelsio.com/gpudirect-rdma](http://www.chelsio.com/gpudirect-rdma)
GPUDirect RDMA over Infiniband

Benchmarking bandwidth

For bandwidth:

$ git clone git://git.openfabrics.org/~grockah/perf test.git

$ cd perftest

$ ./autogen.sh

$ export CUDA_H_PATH=/usr/local/cuda-8.0/include/cuda.h

$ ./configure --prefix=$HOME/test

$ make all install

E.g. host to GPU memory (H-G) BW test:

server$ ~/test/bin/ib_write_bw -n 1000 -O -a --use_cuda

client $ ~/test/bin/ib_write_bw -n 1000 -O -a server.name.org

GPU to GPU memory (G-G) BW test:

server$ ~/test/bin/ib_write_bw -n 1000 -O -a --use_cuda

client $ ~/test/bin/ib_write_bw -n 1000 -O -a --use_cuda server.name.org
DGX-1 GPUDirect RDMA uni-dir BW

IB message size, 5000 iterations, RC protocol
GPUDIRECT ASYNC
GPUDIRECT ASYNC
leverage GPU front-end unit

Communications prepared by CPU
- hardly parallelizable, branch intensive
- GPU orchestrates flow

Run by GPU front-end unit
- Same one scheduling GPU work
- Now also scheduling network communications
GPUDIRECT ASYNC OVER OFA VERBS

SW stack bits

- Prerequisites: nvidia_peer_memory driver, GDRcopy\(^1\) library
- CUDA 8.0+ Stream Memory Operations (MemOps) APIs
- MLNX OFED 4.0+ Peer-Direct Async Verbs APIs
- libgdsync\(^2\): bridging CUDA & IB Verbs
- MPI: experimental support in MVAPICH2-GDS\(^3\)
- libmp: lightweight, MPI-like stream-sync primitives, internal benchmarking

\(^1\) [http://github.com/NVIDIA/gdrcopy](http://github.com/NVIDIA/gdrcopy)
\(^2\) [http://github.com/gpudirect/libgdsync](http://github.com/gpudirect/libgdsync), devel branch
\(^3\) DK Panda’s talk @ GTC 2017
GPUDIRECT ASYNC OVER INFINIBAND

Requirements

May need special HCA configuration on Kepler/Maxwell GPUs, e.g. on Mellanox:

$ mlxconfig -d /dev/mst/mtxxx_pciconf0 set NON_PREFETCHABLE_PF_BAR=1

$ reboot

Enable GPU peer mappings:

$ cat /etc/modprobe.d/nvidia.conf options nvidia NVreg_RegistryDwords="PeerMappingOverride=1"
CUDA STREAM MemOps

CU_STREAM_WAIT_VALUE_GEQ  = 0x0,
CU_STREAM_WAIT_VALUE_EQ   = 0x1,
CU_STREAM_WAIT_VALUE_AND  = 0x2,
CU_STREAM_WAIT_VALUE_NOR  = 0x3,
CU_STREAM_WAIT_VALUE_FLUSH = 1<<30
CUresult cuStreamWaitValue32(CUstream stream, CUdeviceptr addr,
cuuint32_t value, unsigned int flags);
CUresult cuStreamWaitValue64(CUstream stream, CUdeviceptr addr,
cuuint64_t value, unsigned int flags);

CU_STREAM_WRITE_VALUE_NO_MEMORY_BARRIER = 0x1
CUresult cuStreamWriteValue32(CUstream stream, CUdeviceptr addr,
cuuint32_t value, unsigned int flags);
CUresult cuStreamWriteValue64(CUstream stream, CUdeviceptr addr,
cuuint64_t value, unsigned int flags);

CU_STREAM_MEM_OP_WAIT_VALUE_32  = 1,
CU_STREAM_MEM_OP_WRITE_VALUE_32 = 2,
CU_STREAM_MEM_OP_WAIT_VALUE_64  = 4,
CU_STREAM_MEM_OP_WRITE_VALUE_64 = 5,
CU_STREAM_MEM_OP_FLUSH_REMOTE_WRITES = 3
CUresult cuStreamBatchMemOp(CUstream stream, unsigned int count,
CUstreamBatchMemOpParams *paramArray, unsigned int flags);

guarantees memory consistency for RDMA
polling on 32/64bit word
32/64bit word write
lower-overhead batched work submission
CUDA STREAM MemOps

APIs features

- batching multiple consecutive MemOps save ~1.5us each op
  - use cuStreamBatchMemOp()
- APIs accept device pointers
  - memory need registration (cuMemHostRegister)
  - device pointer retrieval (cuMemHostGetDevicePointer)
- 3rd party device PCIe resources (aka BARs)
  - assumed physically contiguous & uncached
  - special flag needed in cuMemHostRegister
ASYNC: LIBMP
LIBMP

Lightweight message passing library
  thin layer on top of IB Verbs
  in-order receive buffer matching
  no tags, no wildcards, no data types
  zero copy transfers, uses flow control of IB RC transport

Eases benchmarking of multi-GPU applications
Not released (yet)
# LIBMP

Prototype message passing library

<table>
<thead>
<tr>
<th></th>
<th>PREPARED BY</th>
<th>TRIGGERED BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU synchronous</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>Stream synchronous</td>
<td>CPU</td>
<td>GPU front-end unit</td>
</tr>
<tr>
<td>Kernel initiated</td>
<td>CPU</td>
<td>GPU SMs</td>
</tr>
</tbody>
</table>
LIBMP CPU-SYNC COMMUNICATION

// Send/Recv
int mp_irecv (void *buf, int size, int peer, mp_reg_t *mp_reg, mp_request_t *req);
int mp_isend (void *buf, int size, int peer, mp_reg_t *mp_reg, mp_request_t *req);

// Put/Get
int mp_window_create(void *addr, size_t size, mp_window_t *window_t);
int mp_iput (void *src, int size, mp_reg_t *src_reg, int peer, size_t displ, mp_window_t *dst_window_t, mp_request_t *req);
int mp_iget (void *dst, int size, mp_reg_t *dst_reg, int peer, size_t displ, mp_window_t *src_window_t, mp_request_t *req);

// Wait
int mp_wait (mp_request_t *req);
int mp_wait_all (uint32_t count, mp_request_t *req);
Standard CPU-sync communications

Loop {
    mp_irecv(...)
}

Loop {
    compute <<<...,stream>>> (buf)
    compute (on GPU)
    cudaStreamSynchronize (stream)
    near neighbor communication exchange
    mp_isend(...)
}

mp_wait_all (...)

100% CPU utilization
Limited scaling!
LIBMP STREAM-SYNC COMMUNICATION

// Send

int mp_isend_on_stream (void *buf, int size, int peer, mp_reg_t *mp_reg, mp_request_t *req, cudaStream_t stream);

// Put/Get

int mp_iput_on_stream (void *src, int size, mp_reg_t *src_reg, int peer, size_t displ, mp_window_t *dst_window_t, mp_request_t *req, cudaStream_t stream);

int mp_iget_on_stream (void *dst, int size, mp_reg_t *dst_reg, int peer, size_t displ, mp_window_t *src_window_t, mp_request_t *req, cudaStream_t stream);

// Wait

int mp_wait_on_stream (mp_request_t *req, cudaStream_t stream);

int mp_wait_all_on_stream (uint32_t count, mp_request_t *req, cudaStream_t stream);
CUDA stream-sync communications

Loop {
    mp_irecv(...)
    compute <<<...,stream>>> (buf)
    mp_isend_on_stream(...)
    mp_wait_all_on_stream (...)
}
LIBMPI STREAM-SYNC batch submission

// Prepare single requests, IB Verbs side only

int mp_send_prepare (void *buf, int size, int peer, mp_reg_t *mp_reg, mp_request_t *req);

// Post set of prepared requests

int mp_isend_post_on_stream (mp_request_t *req, cudaStream_t stream);

int mp_isend_post_all_on_stream (uint32_t count, mp_request_t *req, cudaStream_t stream);
LIBMP KERNEL-SYNC COMMUNICATION

// Host side code
// extract descriptors from prepared requests

int mp::mlx5::get_descriptors(send_desc_t *send_info, mp_request_t *req);
int mp::mlx5::get_descriptors(wait_desc_t *wait_info, mp_request_t *req);

// Device side code

__device__ int mp::device::mlx5::send(send_desc_t * send_info);
__device__ int mp::device::mlx5::wait (wait_desc_t * wait_info);
Kernel-sync communication

Loop {
    mp_irecv(..., rreq);
    mp_get_descriptors(wi, rreq);
    mp_send_prepare(..., sreq);
    mp_get_descriptors(si, sreq);

    compute_and_communicate <<<...,stream>>> (buf,wi,si) {
        do_something(buf);
        mlx5::send(si);
        do_something_else();
        mlx5::wait(wi);
        keep_working();
    }
}

CPU is free!
Deep learning on GPUs

Deeper neural networks, larger data sets … training is a very, very long operation!
NCCL

To other systems
- Sockets (Ethernet)
- Infiniband, with GPU Direct RDMA

Within a system
- NVLink
- PCIe
- GPU Direct P2P
NCCL

Deep Learning Frameworks

Caffe  Caffe2  Torch  TF  MXNET  CNTK

NCCL  CUDNN  CUBLAS

CUDA

NVIDIA GPUs
AGENDA

NCCL
  History
  Design

NCCL 2.0
  New features
  API Changes

Performance

Future
History

Q4 2015: NCCL 1.x

Open-source research project on github, helping Deep Learning frameworks compute on multiple GPUs with efficient collective operations.

Limited to intra-node.

Q2 2017: NCCL 2.x and beyond

NVIDIA Library, multi-node support and improved API.
**Design**

**Optimized collective communication library** between CUDA devices. Implements Allreduce, Reduce, Broadcast, Reduce-scatter, Allgather.

**Easy to integrate** into any DL framework, as well as traditional HPC apps using MPI.

Runs on the GPU using asynchronous CUDA kernels, for faster access to GPU memory, parallel reductions, NVLink usage.

Operates on **CUDA pointers**. Operations are tied to a CUDA stream.

Uses as little threads as possible to **permit other computation** to progress simultaneously.
Design

NCCL uses **rings** to move data across all GPUs and perform reductions.
NCCL uses **rings** to move data across all GPUs and perform reductions.

PCIe / QPI : 1 unidirectional ring
NCCL uses **rings** to move data across all GPUs and perform reductions.

**Diagram:**

- **PCiE / QPI:** 1 unidirectional ring
- **DGX-1:** 4 unidirectional rings
NCCL 2.0
NCCL 2.0

Inter-node communication using Sockets or Infiniband verbs, with multi-rail support, topology detection and automatic use of GPU Direct RDMA.

Optimal combination of NVLink, PCI and network interfaces to maximize bandwidth and create rings across nodes.
NCCL 2.0

Inter-node communication using Sockets or Infiniband verbs, with multi-rail support, topology detection and automatic use of GPU Direct RDMA.

Optimal combination of NVLink, PCI and network interfaces to maximize bandwidth and create rings across nodes.
NCCL 2.0

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0 API

NCCL 2.0 is introducing mandatory new verbs `ncclGroupStart/ncclGroupEnd` when managing multiple devices from a single thread.

NCCL 1.x:

```c
for (int i=0; i<ngpus; i++) {
    cudaSetDevice(devices[i]);
    ncclAllReduce(..., comms[i], streams[i]);
}
```

NCCL 2.0:

```c
ncclGroupStart();
for (int i=0; i<ngpus; i++) {
    ncclAllReduce(..., comms[i], streams[i]);
}
ncclGroupEnd();
```
NCCL 2.0 API

Inter-node communicator creation still uses the NCCL 1.x verbs: `ncclGetUniqueId/ncclCommInitRank`

```c
if (rank == 0) ncclGetUniqueId(&id)
My_Bcast(&id);
ncclCommInitRank(&comm, nranks, id, rank);
```

Multi-process + multi-GPU per process (from a single thread): combine `ncclCommInitRank` with `ncclGroupStart/ncclGroupEnd`

```c
if (rank == 0) ncclGetUniqueId(&id)
My_Bcast(&id);
ncclGroupStart();
for (int i=0; i<ndev; i++) {
  cudaSetDevice(devices[i]);
  ncclCommInitRank(&comm, ndev*nranks, id, ndev*rank+i);
}
ncclGroupEnd();
```
NCCL 2.0 API

Other small API adjustments over the NCCL 1.x API:

Counts are now of type `size_t` instead of `int`

`allGather` arguments order has been fixed to be similar to other operations

Additions/clarification on `datatypes`:

- integral: `int8 = char, uint8, int32 = int, uint32, int64, uint64`
- floating point: `float16 = half, float32 = float, float64 = double`

Clarifications and fixes for `allgather` and `reduce_scatter` send/receive `counts` and `in-place` operations
Performance
performance

AllReduce bandwidth (OMB, size=128MB, in GB/s)
performance

AllReduce bandwidth (OMB, size=128MB, in GB/s)

- MPI
- Baidu Allreduce
- NCCL

2 nodes x 4 GPUs (IB EDR, PCI Switch)
4 nodes x 8 GPUs (DGX-1: 4x IB EDR, 4x NVLink)
CNTK Scaling, ResNet50, Images/Sec

![Graph showing CNTK scaling with MPI, NCCL, and Ideal lines.](image-url)
Future
Future

Additional communication primitives:
- point-to-point communication
- scatter (1 to N), gather (N to 1), alltoall (N to N)
- neighbor collectives (send/receive in multiple dimensions)

User-defined reduction operations
- also, trying to merge computation and communication better

Windows support

Please let us know your needs!