
Architecting and
Managing GPU Clusters
Dale Southard, NVIDIA
NVIDIA

# About the Speaker and You

[Dale] is a senior solution architect with NVIDIA (I fix things). I primarily cover HPC in Gov/Edu/Research and on occasion cloud computing. In the past I was an architect in the LLNL systems group designing the vis/post-processing solutions.

[You] are here because you are interested in architecting and/or managing GPU clusters for high performance computing.

# Outline

- Hardware Selection
- GPU Management

# Hardware Selection

# Start with the Correct GPU

- Passively Cooled
- Higher Performance
- Chassis/BMC Integration
- Out-of-Band Monitoring

## Tesla M-series is Designed for Servers

(C-series GPUs will work, but those target workstation environments, not servers)

# Choosing the Correct Server

Historically sites wanted “x16 slots”

…then we wanted “x16 electrical slots”

…then we wanted “full bandwidth”

…then we wanted “full simultaneous bandwidth”

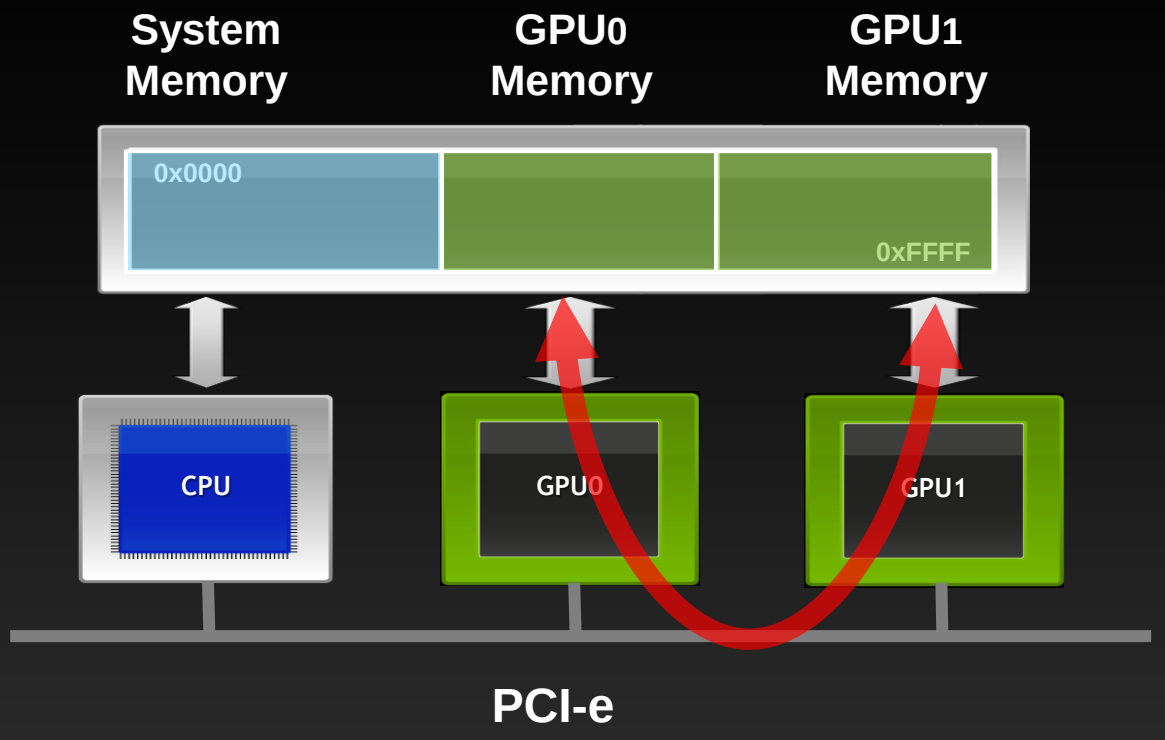
…then we wanted “dual IOH bridges”

**Now we will want “peer-to-peer capable”**

# Doing it Wrong

| | |
|---|---|
| QPI @3.2 | 12.8 GB/s |
| HT 3.0 (16b) | 10.4 GB/s |
| PCIe gen2 | 8 GB/s |
| PCIe gen3 | 16 GB/s |

# UVA is Driving Server Design

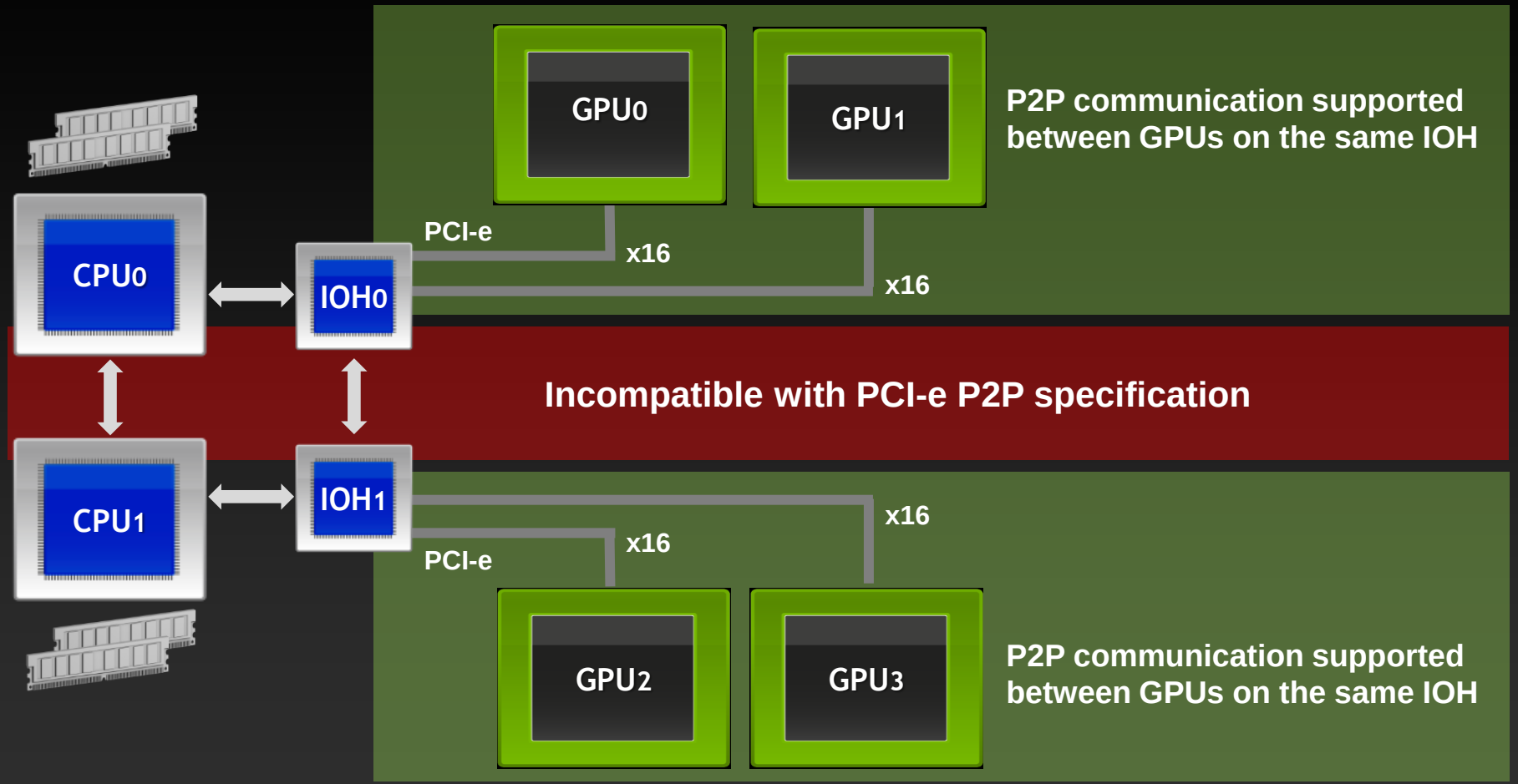



## Direct Transfer & Direct Access between GPUs

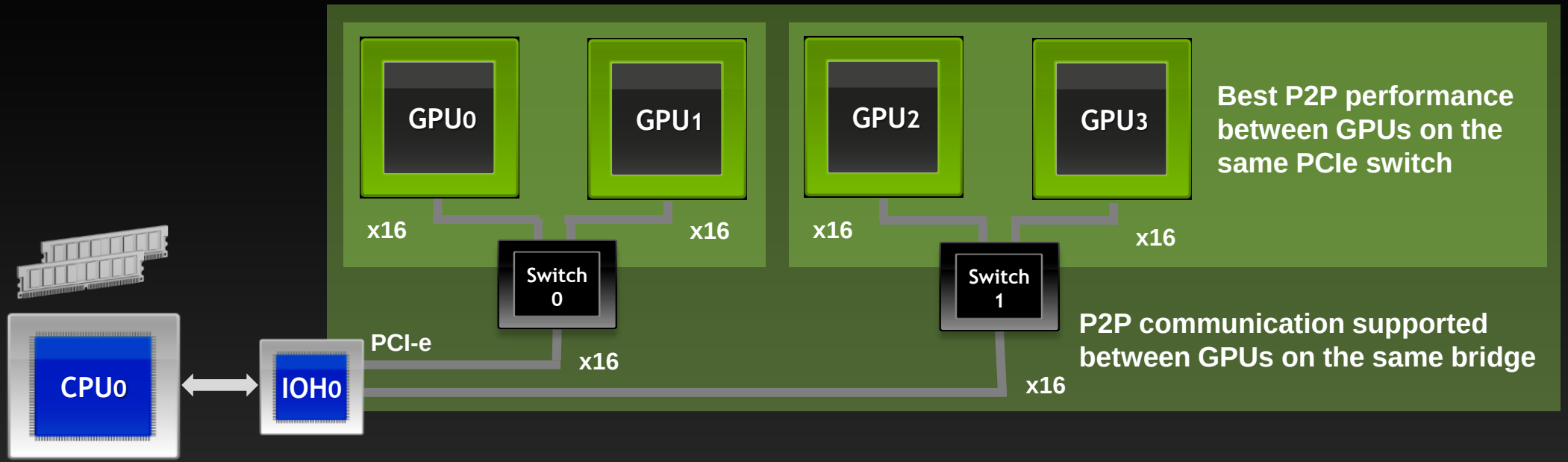
without going through host memory

# Topology Matters for P2P Communication

**P2P Communication is Not Supported Between Bridges**

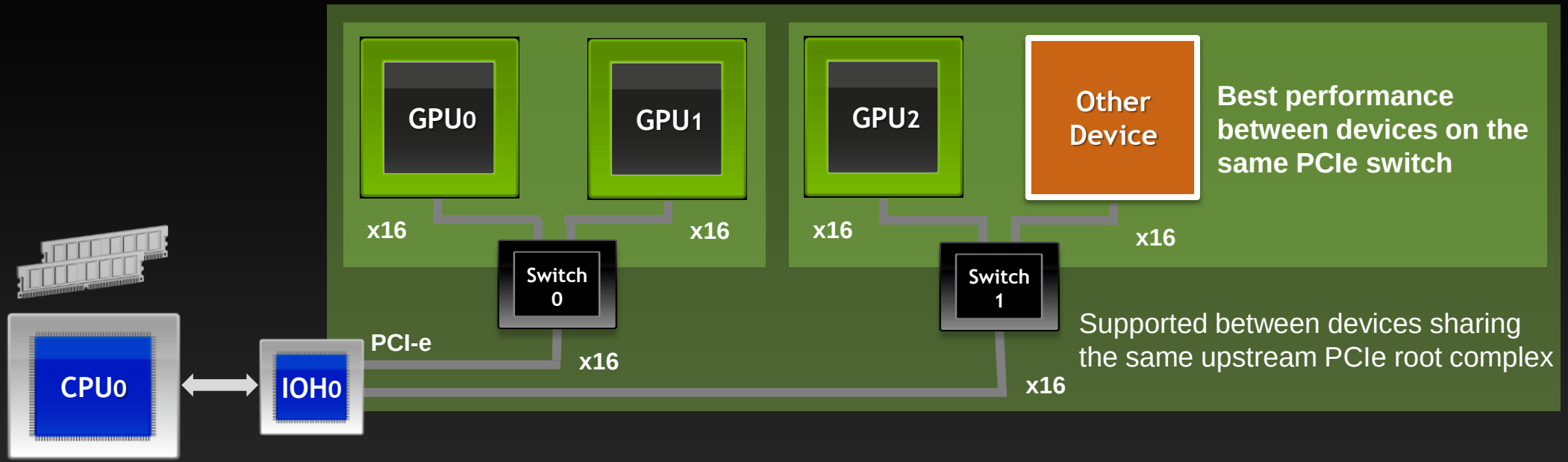


# Topology Matters, PCIe Switches



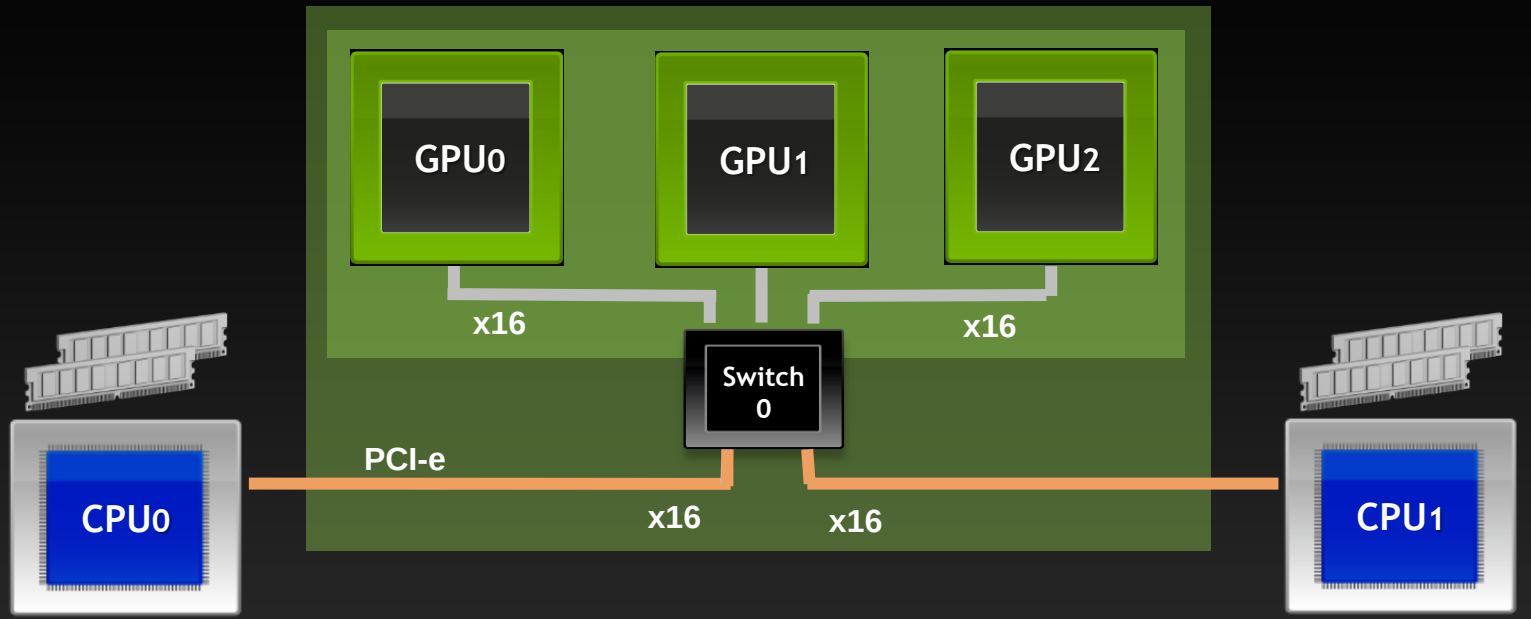

## PCI-e Switches: Fully Supported

# Topology Matters, GPUDirect RDMA




## PCI-e Switches: Fully Supported

# Non-transparent Switches




**Non-transparent PCI-e Switches: Future Flexibility**

# GPU Management

# GPU Management: Provisioning

- Both diskfull and diskless work
- May want to repackage driver
- May want to install libs on login nodes without GPUs

```
sh driver.run --no-kernel-module --no-x-check --no-opengl-files
```

- No real gotcha's for compute use
- For OpenGL use, beware of dual ownership of libGL

# Provisioning Distro'isms

Don't let Nouveau driver load/modeset

- `echo -e 'blacklist nouveau\noptions nouveau modeset=0' >\`
  `/etc/modprobe.d/disable-nouveau.conf`
- May require `dracut` or `update-initramfs`

Ubuntu

- Upstart will respawn display manager,
  use `` `stop gdm` `` or `` `stop lightdm` ``
- maybe from ssh connection in 11.04+

# GPU Management: Booting

Most clusters operate at runlevel 3 (no xdm), so best practice is to configure the GPUs from an init.d script:

- `modprobe nvidia`
- `mknod` devices
- Assert that ECC is set correctly
- Set compute-exclusive mode (optional)
- Set persistence (optional)

NVIDIA provides both command-line (`nvidia-smi`) & API (NVML)

# Compute-exclusive Mode

- Can be set with NVML or nvidia-smi
- Two flavors thread-exclusive and process-exclusive
- Fast fail when things escape the batch system
- Not a substitute for an authoritative batch/resource scheduler

# Persistence Mode

- Can be set with NVML or nvidia-smi
- Causes driver to maintain a persistent connection to the GPU
  - Faster job startup
  - But GPU memory isn't cleared between jobs

No longer required, but still (debatably) best practice

# GPU Management: Monitoring

- Environmental and utilization metrics are available
- Tesla M-series may provide OoB access to telemetry via BMC
- NVML can be used from Python or Perl
- NMVL has been integrated into Ganglia gmond.

http://developer.nvidia.com/nvidia-management-library-nvml

# GPU Management: Finding Problems

- Monitoring though nvidia-smi or NVML
- Watching for Xid errors in syslog
- PCIe parity via EDAC

```
modprobe    edac_mc
echo 1 > /sys/devices/system/edac/pci/check_pci_parity
```

- User feedback and testing

# Handling Bad Devices

- Three different enumeration systems:
  - PCIe
  - CUDA runtime
  - nvidia-smi
- Do not assume that the three enumerations are consistent!
- PCIe device ID, serial number, and UUID are consitent

Always have operators check serial number of pulled HW

# GPU Management: Error Containment

Tesla 20-series:

- ECC Protection on DRAM and on-chip memories
- ECC can be turned on/off via `nvidia-smi` or NVML (reboot)
- Volatile and Persistent ECC counters
- Counts available via `nvidia-smi`, NVML, and sometime SMBus
- Correctable errors are logged but not scrubbed
- Uncorrectable errors cause user and Xid system error
- Uncorrectable errors cause GPU to reject work until reboot
- Uncorrectable errors do not cause MCE or reboot

Full ECC containment without requiring system halt

# GPU Management: Job Scheduling

For time-sharing a node use **$CUDA_VISIBLE_DEVICES:**

```
$ ./deviceQuery -noprompt | egrep "^Device"
 Device 0: "Tesla C2050"
 Device 1: "Tesla C1060"
 Device 2: "Quadro FX 3800"

$ export CUDA_VISIBLE_DEVICES="0,2"

$ ./deviceQuery -noprompt | egrep "^Device"
 Device 0: "Tesla C2050"
 Device 1: "Quadro FX 3800"
```

**Several batch systems and resource mangers support GPUs as independent consumables using $CUDA_VISIBLE_DEVICES**

# GPU Management: Resource Limits

- UVA depends on allocating virtual address space
- Virtual address space != physical ram consumption
- Use cgroups, not ulimit

Several batch systems and resource mangers support cgroups directly or via plugins.

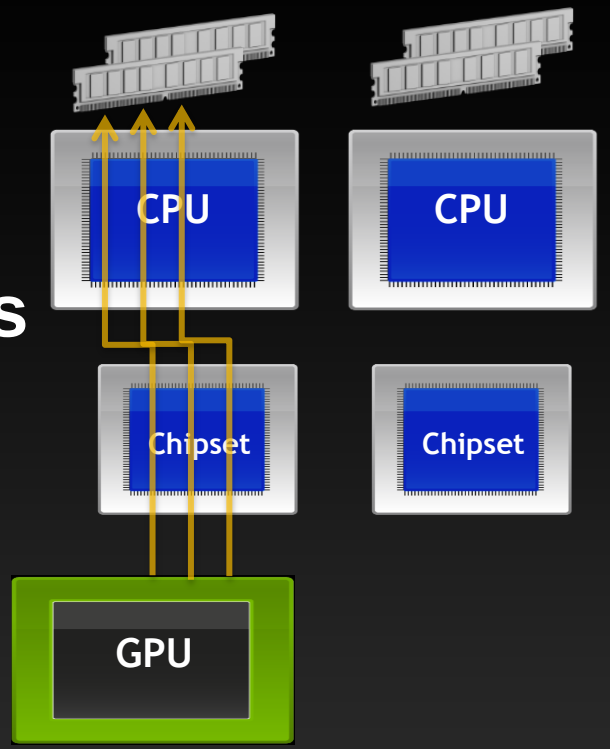
# GPU Utilization: GPUDirect v1

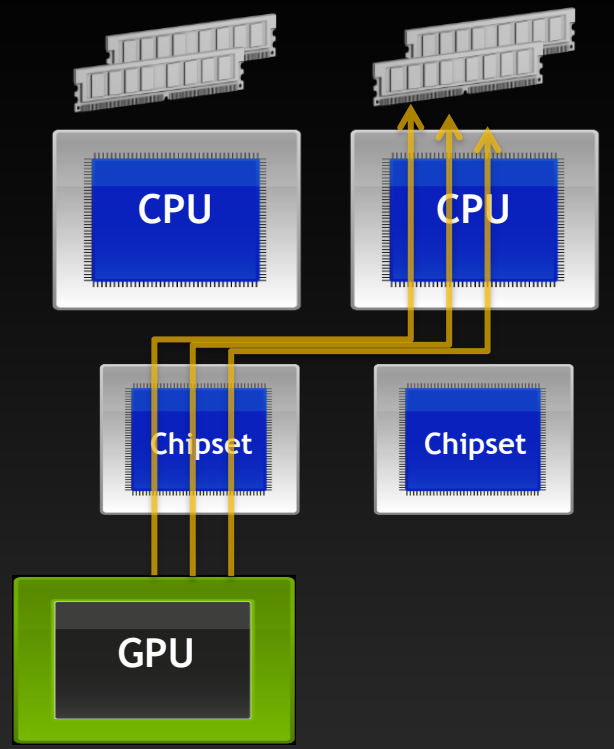
- With CUDA 4.0 use `$CUDA_NIC_INTEROP`
  - Setting envar to 1 forces the cuda allocation of pinned host memory down an alternate path which allows interoperation with other devices.
- With CUDA 4.1 the default allocation path allows interoperation.
- `cudaHostRegister()`
  - Pining and registering of memory allocated outside the CUDA API.

**GPUDirect is no longer a kernel patch**

# GPU Utilization: Topology Matters (again)

6.5 GB/s ☺




4.34 GB/s ☹

MPI rank (process) mapping impacts performance

# GPU Utilization: Detecting Local Rank

Easy on OpenMPI and MVAPICH, modular arithmetic everywhere else

```
if [[ -n ${OMPI_COMM_WORLD_LOCAL_RANK} ]]
  then
  lrank=${OMPI_COMM_WORLD_LOCAL_RANK}
elif [[ -n ${MV2_COMM_WORLD_LOCAL_RANK} ]]
  then
  lrank= ${MV2_COMM_WORLD_LOCAL_RANK}
elif [[ -n ${PMI_ID} && -n ${MPISPAWN_LOCAL_NPROCS} ]]
  then
  let lrank=${PMI_ID}%${PERHOST}
else
  echo could not determine local rank
fi
```

# GPU Utilization: NUMA Placement

Once you have the local rank, use it to bind the process to NUMA

```
case ${lrank} in
[0-1])
  numactl --cpunodebind=0 ${@}
  ;;
[2-3])
  numactl --cpunodebind=2 ${@}
  ;;
[4-5])
  numactl --cpunodebind=3 ${@}
  ;;
esac
```

# GPU Utilization: Wrapping for NUMA

Glue the previous two fragments into a wrapper script (`numawrap`)

- Use as `mpirun numawrap rest -of -command`
- The `${@}` will pick up the command at runtime
- Only need to figure out the topology once
- May require MV2_USE_AFFINITY=0 ; MV2_ENABLE_AFFINITY=0

- Eventually portable hardware locality (hwloc) may do this for us.

# Summary

# Things to Take Home

- Pick the correct HW
- Topology Matters
- Lots of hooks for management
- Topology Matters

# Questions