GPU TECHNOLOGY CONFERENCE

TOOLS AND TIPS FOR MANAGING A GPU CLUSTER

Adam DeConinck

HPC Systems Engineer, NVIDIA

Steps for configuring a GPU cluster

- ❑ Select compute node hardware
- ❑ Configure your compute nodes
- ❑ Set up your cluster for GPU jobs
- ❑ Monitor and test your cluster

NVML and nvidia-smi

Primary management tools mentioned throughout this talk will be **NVML** and **`nvidia-smi`**

NVML: NVIDIA Management Library

- Query state and configure GPU
- C, Perl, and Python API

`nvidia-smi`: Command-line client for NVML

GPU Deployment Kit: includes NVML headers, docs, and `nvidia-healthmon`

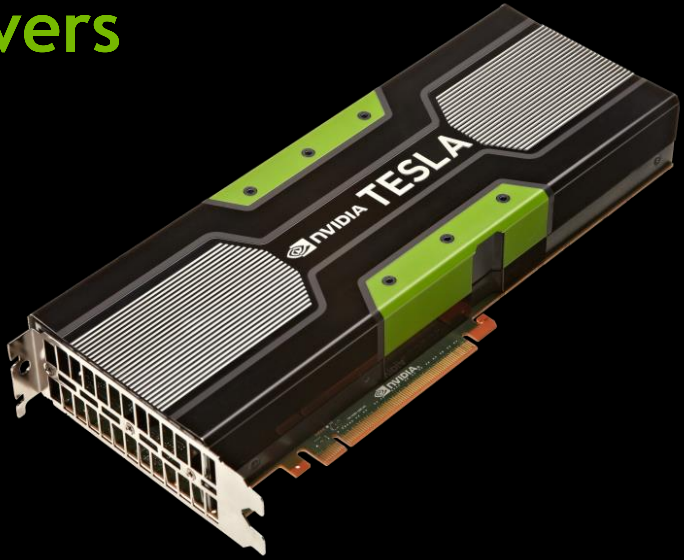
Select compute node hardware

- ❑ Choose the correct GPU
- ❑ Select server hardware
- ❑ Consider compatibility with networking hardware

What GPU should I use?

Tesla M-series is designed for servers

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

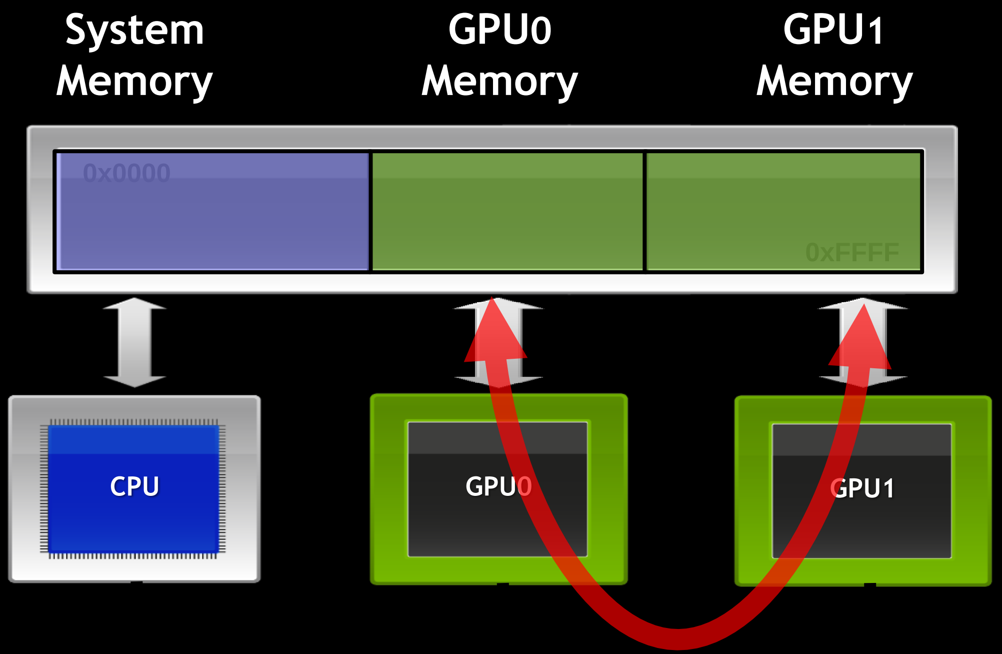


GPU TECHNOLOGY CONFERENCE

PCIe Topology Matters

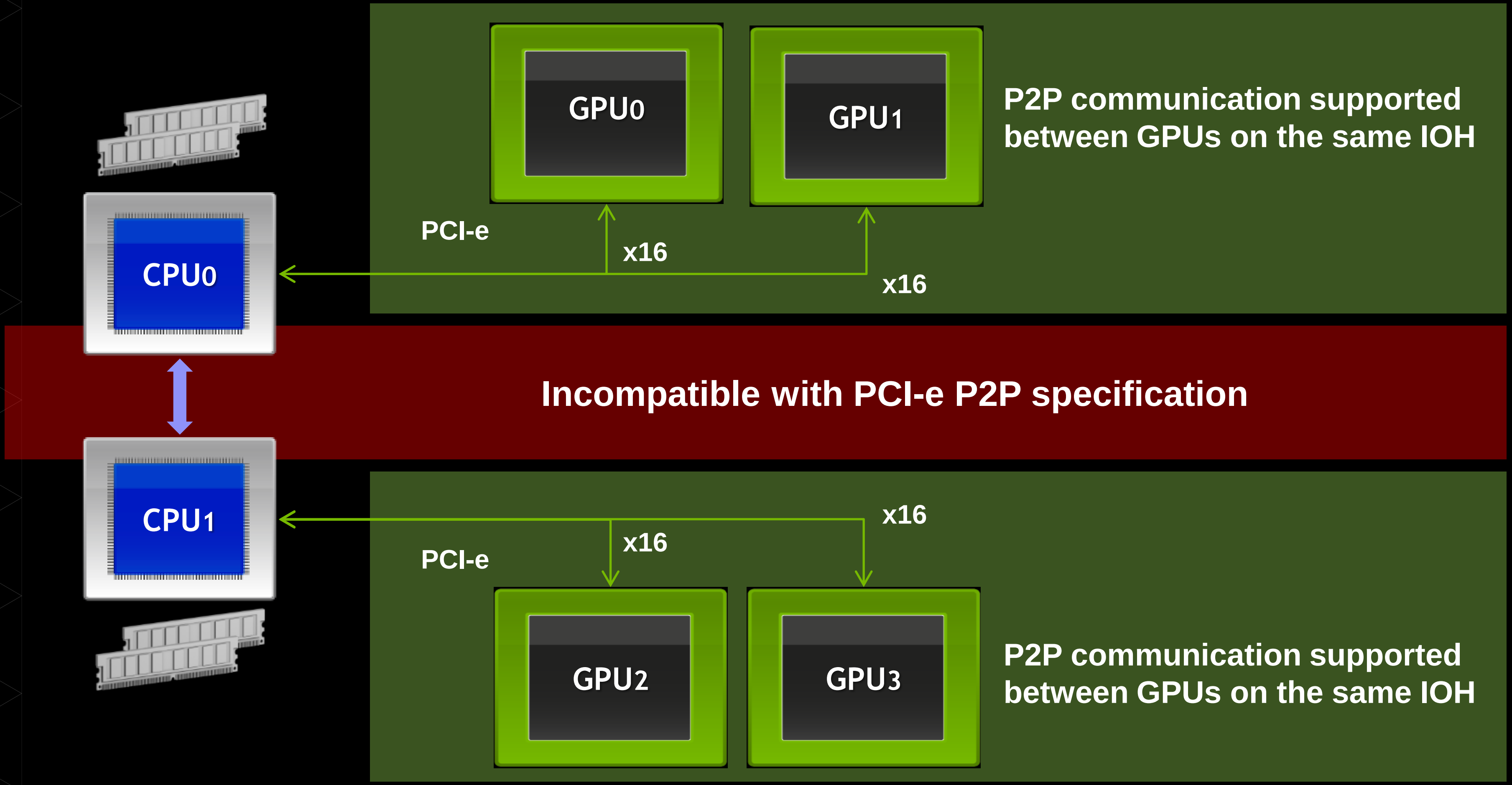
Biggest factor right now in server selection is **PCIe topology**

- Direct memory access between devices w/ P2P transfers
- Unified addressing for system and GPUs

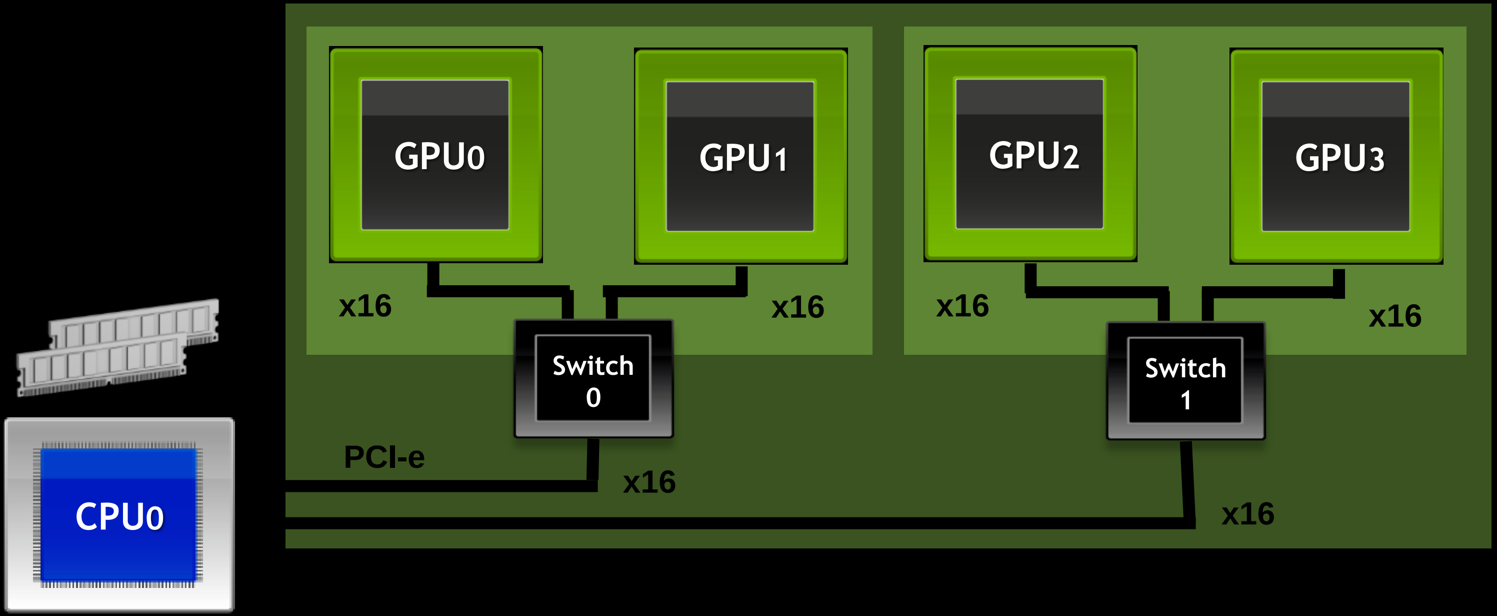


- Works best when all devices are on same PCIe root or switch

P2P on dual-socket servers



For many GPUs, use PCIe switches

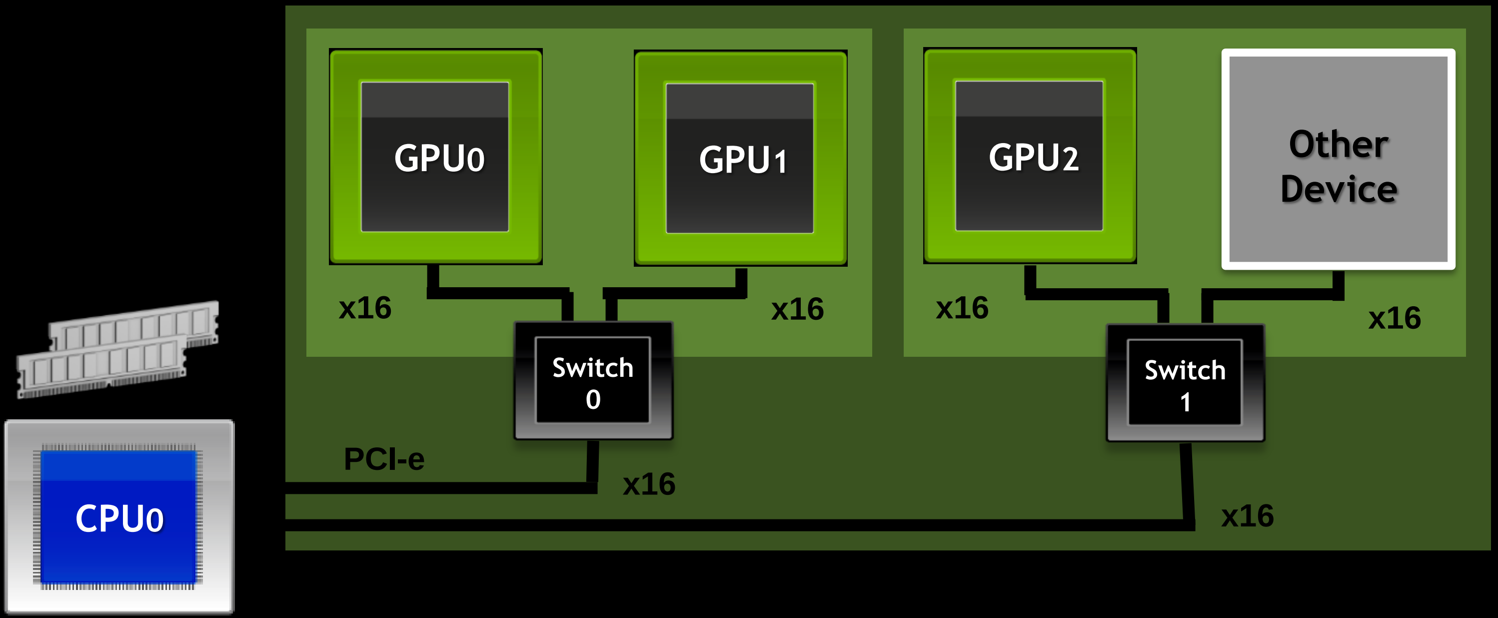


- PCIe switches fully supported
- Best P2P performance between devices on same switch

How many GPUs per server?

- If apps use P2P heavily:
  - More GPUs per node are better
  - Choose servers with appropriate PCIe topology
  - Tune application to do transfers within PCIe complex
- If apps don't use P2P:
  - May be dominated by host <-> device data transfers
  - More servers with fewer GPUs/server

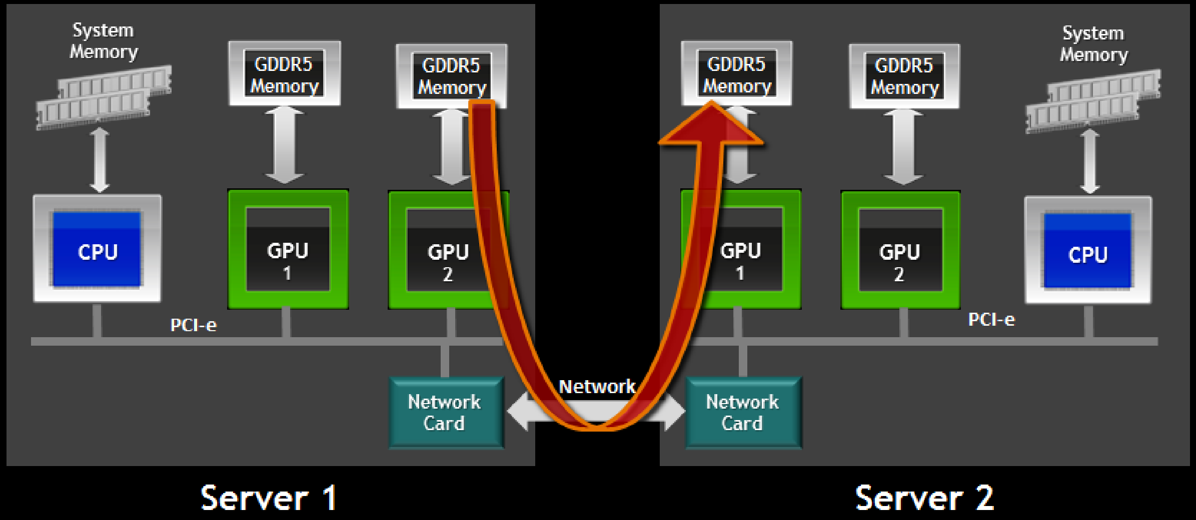
For many devices, use PCIe switches



- PCIe switches fully supported for all operations
- Best P2P performance between devices on same switch

- P2P also supported with other devices such as NIC via **GPUDirect RDMA**

GPUDirect RDMA on the network



- PCIe P2P between NIC and GPU without touching host memory
- Greatly improved performance
- Currently supported on Cray (XK7 and XC-30) and Mellanox FDR Infiniband
- Some MPI implementations support GPUDirect RDMA

Configure your compute nodes

- ❑ Configure system BIOS
- ❑ Install and configure device drivers
- ❑ Configure GPU devices
- ❑ Set GPU power limits

Configure system BIOS

- Configure large PCIe address space
  - Many servers ship with 64-bit PCIe addressing turned off
  - Needs to be turned on for Tesla K40 or systems with many GPUs
  - Might be called “Enable 4G Decoding” or similar
- Configure for cooling passive GPUs
  - Tesla M-series has passive cooling – relies on system fans
  - Communicates thermals to BMC to manage fan speed
  - Make sure BMC firmware is up to date, fans are configured correctly
- Make sure remote console uses onboard VGA, not “offboard” NVIDIA GPU

Disable the nouveau driver

nouveau does not support CUDA and will conflict with NVIDIA driver

Two steps to disable:

1. Edit `/etc/modprobe.d/disable-nouveau.conf:`

```
blacklist nouveau
nouveau modeset=0
```

2. Rebuild initial ramdisk:

RHEL: `dracut --force`

SUSE: `mkinitrd`

Deb: `update-initramfs -u`

Install the NVIDIA driver

Two ways to install the driver

- Command-line installer
  - Bundled with CUDA toolkit – developer.nvidia.com/cuda
  - Stand-alone – www.nvidia.com/drivers
- RPM/DEB
  - Provided by NVIDIA (major versions only)
  - Provided by Linux distros (other release schedule)
- Not easy to switch between these methods

Initializing a GPU in runlevel 3

Most clusters operate at runlevel 3 so you should initialize the GPU explicitly in an init script

- At minimum:
  - Load kernel modules – nvidia + nvidia_uvm (in CUDA 6)
  - Create devices with mknod

- Optional steps:
  - Configure compute mode
  - Set driver persistence
  - Set power limits

Install GPUDirect RDMA network drivers
(if available)

- Mellanox OFED 2.1 (beta) has support for GPUDirect RDMA
  - Should also be supported on Cray systems for CLE <…>

- HW required: Mellanox FDR HCAs, Tesla K10/K20/K20X/K40
- SW required: NVIDIA driver 331.20 or better, CUDA 5.5 or better, GPUDirect plugin from Mellanox

- Enables an additional kernel driver, nv_peer_mem

Configure driver persistence

By default, driver unloads when GPU is idle

- Driver must re-load when job starts, slowing startup
- If ECC is on, memory is cleared between jobs

Persistence daemon keeps driver loaded when GPUs idle:

```
# /usr/bin/nvidia-persistenced --persistence-mode \
[--user <username>]
```

- Faster job startup time
- Slightly lower idle power

GPU TECHNOLOGY CONFERENCE

Configure ECC

- Tesla and Quadro GPUs support ECC memory
  - Correctable errors are logged but not scrubbed
  - Uncorrectable errors cause error at user and system level
  - GPU rejects new work after uncorrectable error, until reboot

- ECC can be turned off – makes more GPU memory available at cost of error correction/detection
  - Configured using NVML or nvidia-smi

```
# nvidia-smi -e 0
```

  - Requires reboot to take effect

Set GPU power limits

- Power consumption limits can be set with NVML/nvidia-smi
- Set on a per-GPU basis
- Useful in power-constrained environments

```
nvidia-smi –pl <power in watts>
```

- Settings don’t persist across reboots – set this in your init script
- Requires driver persistence

Set up your cluster for GPU jobs

- ❑ Enable GPU integration in resource manager and MPI
- ❑ Set up GPU process accounting to measure usage
- ❑ Configure GPU Boost clocks (or allow users to do so)
- ❑ Managing job topology on GPU compute nodes

Resource manager integration

Most popular resource managers have some NVIDIA integration features available: SLURM, Torque, PBS Pro, Univa Grid Engine, LSF

- GPU status monitoring:
  - Report current config, load sensor for utilization
- Managing process topology:
  - GPUs as consumables, assignment using CUDA_VISIBLE_DEVICES
  - Set GPU configuration on a per-job basis
- Health checks:
  - Run nvidia-healthmon or integrate with monitoring system

NVIDIA integration usually configured at compile time (open source) or as a plugin

GPU process accounting

- Provides per-process accounting of GPU usage using Linux PID
- Accessible via NVML or nvidia-smi (in comma-separated format)
- Requires driver be continuously loaded (i.e. persistence mode)
- No RM integration yet, use site scripts i.e. prologue/epilogue

Enable accounting mode:

```
$ sudo nvidia-smi –am 1
```

Human-readable accounting output:

```
$ nvidia-smi –q –d ACCOUNTING
```

Output comma-separated fields:

```
$ nvidia-smi --query-accounted-
apps=gpu_name,gpu_util –
format=csv
```

Clear current accounting logs:

```
$ sudo nvidia-smi -caa
```

MPI integration with CUDA

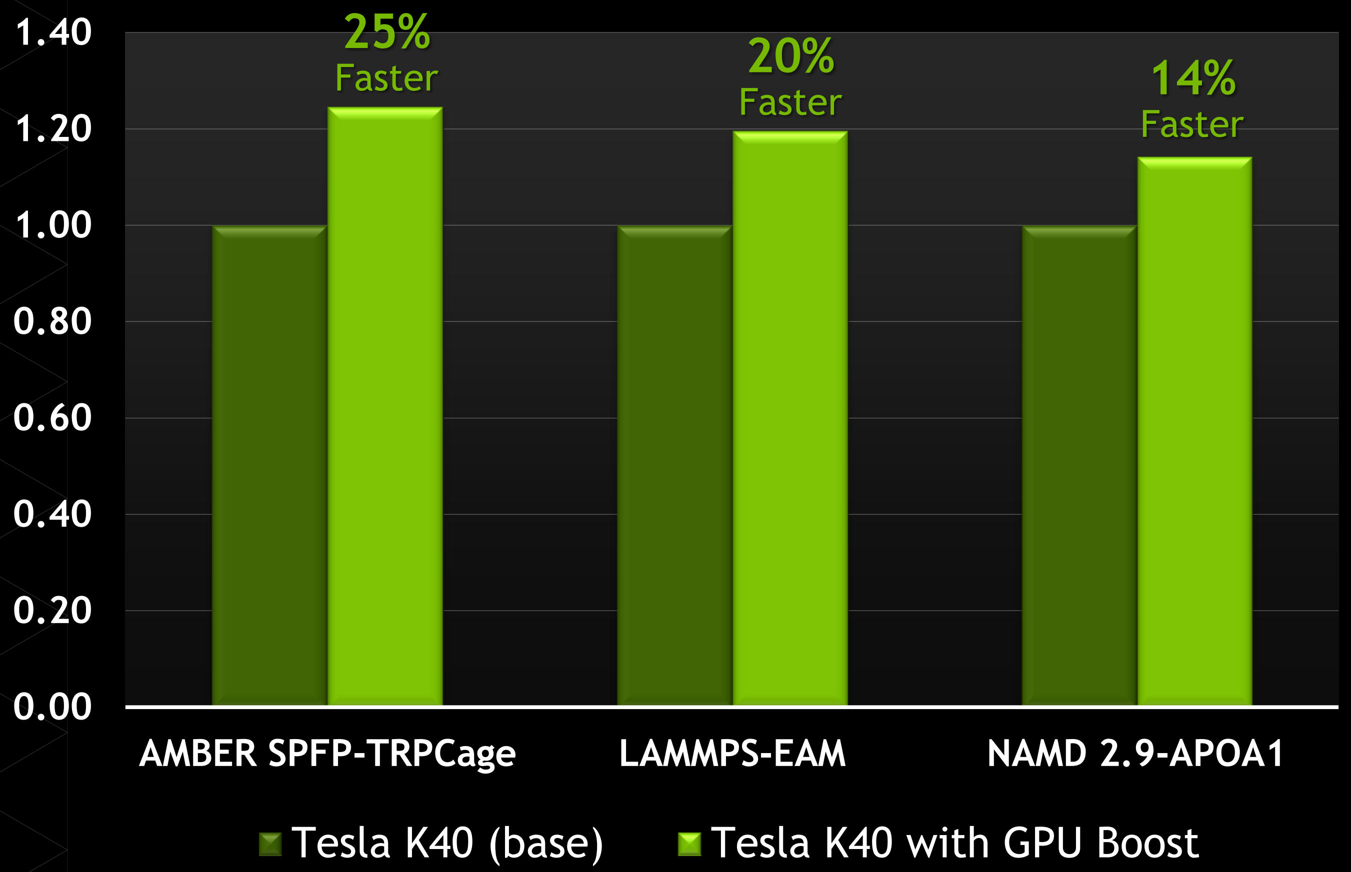
Most recent versions of most MPI libraries support sending/receiving directly from CUDA device memory

- OpenMPI 1.7+, mvapich2 1.8+, Platform MPI, Cray MPT
- Typically needs to be enabled for the MPI at compile time
- Depending on version and system topology, may also support GPUDirect RDMA
- Non-CUDA apps can use the same MPI without problems (but might link libcuda.so even if not needed)

Enable this in MPI modules provided for users

GPU Boost (user-defined clocks)

Use Power Headroom to Run at Higher Clocks



Paradigm — 17% Faster

OpenEye — 13% Faster

ANSYS — 11% Faster

GPU Boost (user-defined clocks)

- Configure with nvidia-smi:

```
nvidia-smi -q -d SUPPORTED_CLOCKS
nvidia-smi -ac <MEM clock, Graphics clock>
nvidia-smi -q -d CLOCK   shows current mode
nvidia-smi -rac   resets all clocks
nvidia-smi -acp 0    allows non-root to change clocks
```

- Changing clocks doesn’t affect power cap; configure separately
- Requires driver persistence
- Currently supported on K20, K20X and K40

Managing CUDA contexts with compute mode

Compute mode: determines how GPUs manage multiple CUDA contexts

- 0/DEFAULT: Accept simultaneous contexts.
- 1/EXCLUSIVE_THREAD: Single context allowed, from a single thread.
- 2/PROHIBITED: No CUDA contexts allowed.
- 3/EXCLUSIVE_PROCESS: Single context allowed, multiple threads OK. Most common setting in clusters.

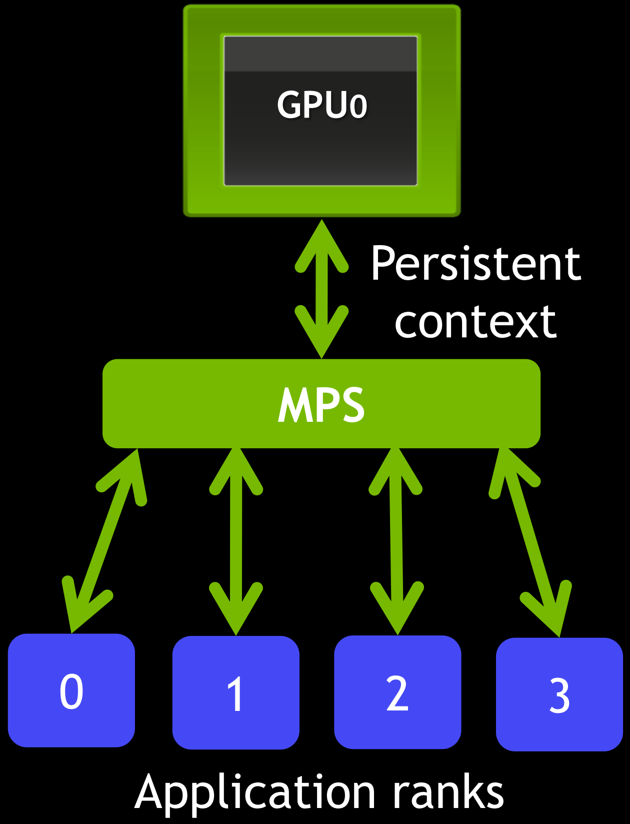
- Changing this setting requires root access, but it sometimes makes sense to make this user-configurable.

N processes on 1 GPU: MPS

- Multi-Process Server allows multiple processes to share a single CUDA context
- Improved performance where multiple processes share GPU (vs multiple open contexts)
- Easier porting of MPI apps: can continue to use one rank per CPU, but all ranks can access the GPU

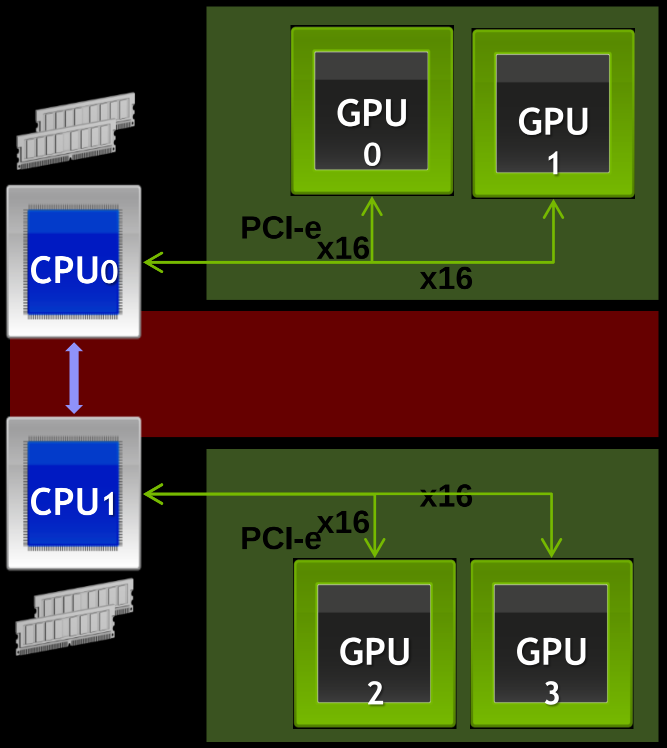
Server process: nvidia-cuda-mps-server

Control daemon: nvidia-cuda-mps-control



PCIe-aware process affinity

To get good performance, CPU processes should be scheduled on cores "local" to the GPUs they use



No good "out of box" tools for this yet!

- hwloc can be help identify CPU <-> GPU locality
- Can use PCIe dev ID with NVML to get CUDA rank
- Set process affinity with MPI or numactl

Possible admin actions:

- Documentation: node toplogy & how to set affinity
- Wrapper scripts using numactl to set "recommended" affinity

Multiple user jobs on a multi-GPU node

CUDA_VISIBLE_DEVICES environment variable controls which GPUs are visible to a process

Comma-separated list of devices

```
export CUDA_VISIBLE_DEVICES="0,2"
```

Tooling and resource manager support exists but limited

- Example: configure SLURM with CPU<->GPU mappings
- SLURM will use cgroups and CUDA_VISIBLE_DEVICES to assign resources
- Limited ability to manage process affinity this way

- Where possible, assign all a job's resources on same PCIe root complex

Monitor and test your cluster

- ❑ Use nvidia-healthmon to do GPU health checks on each job
- ❑ Use a cluster monitoring system to watch GPU behavior
- ❑ Stress test the cluster

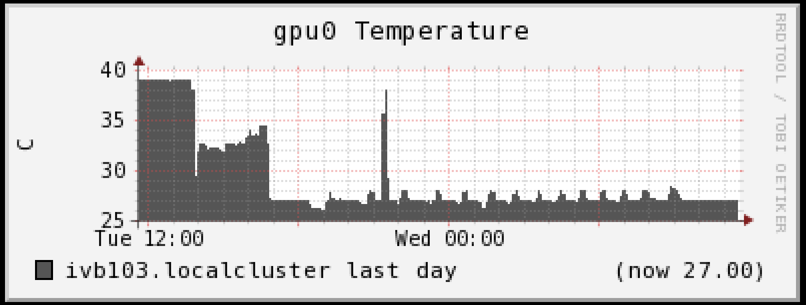
Automatic health checks: nvidia-healthmon

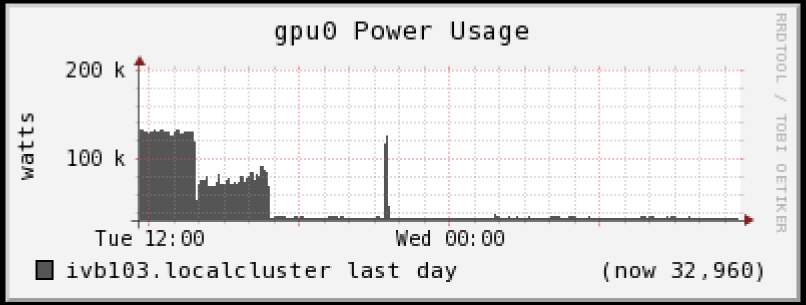
- Runs a set of fast sanity checks against each GPU in system
  - Basic sanity checks
  - PCIe link config and bandwith between host and peers
  - GPU temperature

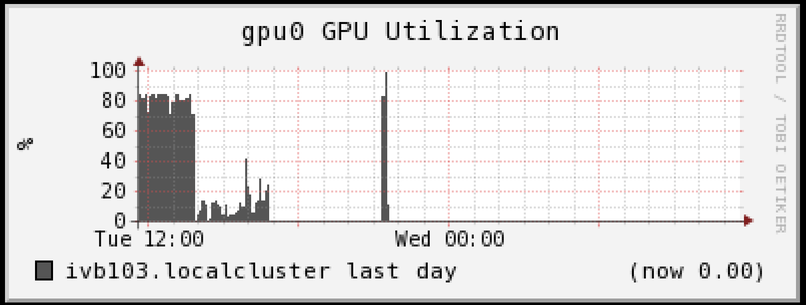
- All checks are configurable – set them up based on your system's expected values

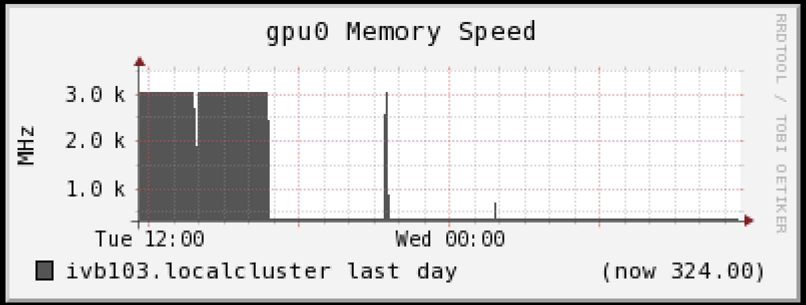
- **Use cluster health checker to run this for every job**
  - Single command to run all checks
  - Returns 0 if successful, non-zero if a test fails
  - Does not require root to run

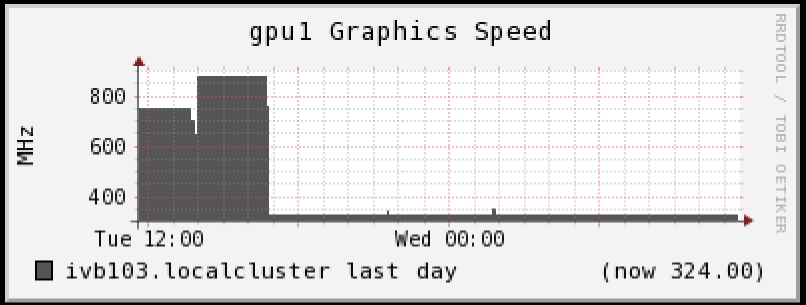
Use a monitoring system with NVML support











Examples: Ganglia, Nagios, Bright Cluster Manager, Platform HPC

Or write your own plugins using NVML

Good things to monitor

- GPU Temperature
  - Check for hot spots
  - Monitor w/ NVML or OOB via system BMC
- GPU Power Usage
  - Higher than expected power usage => possible HW issues
- Current clock speeds
  - Lower than expected => power capping or HW problems
  - Check “Clocks Throttle Reasons” in nvidia-smi
- ECC error counts

Good things to monitor

- Xid errors in syslog
  - May indicate HW error or programming error
  - Common non-HW causes: out-of-bounds memory access (13), illegal access (31), bad termination of program (45)

- Turn on PCIe parity checking with EDAC

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

  - Monitor value of /sys/devices/<pci-address>/broken_parity_status

Stress-test your cluster

- Best workload for testing is the user application
- Alternatively use CUDA Samples or benchmarks (like HPL)
- Stress entire system, not just GPUs

- Do repeated runs in succession to stress the system
- Things to watch for:
  - Inconsistent perf between nodes: config errors on some nodes
  - Inconsistent perf between runs: cooling issues, check GPU Temps
  - Slow GPUs / PCIe transfers: misconfigured SBIOS, seating issues

- Get “pilot” users with stressful workloads, monitor during their runs
- **Use successful test data for stricter bounds on monitoring and healthmon**

Always use serial number to identify bad boards

Multiple possible ways to enumerate GPUs:

- PCIe
- NVML
- CUDA runtime

These may not be consistent with each other or between boots!

Serial number will always map to the physical board and is printed on the board.

UUID will always map to the individual GPU.

(I.e., 2 UUIDs and 1 SN if a board has 2 GPUs.)

Key take-aways

- Topology matters!
  - For both HW selection and job configuration
  - You should provide tools which expose this to your users
- Use NVML-enabled tools for GPU cofiguration and monitoring (or write your own!)
- Lots of hooks exist for cluster integration and management, and third-party tools

Where to find more information

- docs.nvidia.com
- developer.nvidia.com/cluster-management
- Documentation in GPU Deployment Kit
- man pages for the tools (`nvidia-smi`, `nvidia-healthmon`, etc)

- Other talks in the "Clusters and GPU Management" tag here at GTC

GPU TECHNOLOGY CONFERENCE

NVIDIA

QUESTIONS?

@ajdecon

#GTC14