Management and Monitoring of GPU Clusters

Axel Koehler
Sr. Solution Architect HPC

HPC Advisory Council Meeting, March 13-15 2013, Lugano
Agenda

- Introduction into Management and Monitoring of GPU Clusters
- Tools Overview
  - NVML, nvidia-smi, nvidia-healthmon
  - Out-of-Band Management
  - Third Party Management Tools
- GPU Management and Control
  - GPU Modes, Persistence Mode, GPU UUID, InfoROM
- GPU Power Management
  - Power Limits, Application Clocks, GOM Modes
- GPU Job Scheduling
  - Scheduling specific GPUs, Hardware Locality
- Summary
Management and Monitoring of GPU clusters

- **Installation**
  - Ease of installation
  - Integration into deployment tools

- **Scheduling GPU Jobs**
  - High Utilization
  - Topology aware scheduling

- **Power Management**
  - Set Power Limits
  - Query clock throttle reasons

- **Monitor GPU State**
  - GPU and memory utilization
  - ECC error events
  - Thermals

- **Ability to Change GPU State**
  - Change compute mode
  - Enable ECC support or clear ECC error counts

- **Systems Interoperability**
  - OOB Management
  - Integration with Third Party Management Tools
Tools Overview
NVML is available as part of the Tesla Deployment Kit (TDK)
http://developer.nvidia.com/tesla-deployment-kit
C-based interface for monitoring and managing various states within NVIDIA GPUs

Intended to be a platform for building 3rd party applications

Thread-safe to make simultaneous NVML calls from multiple threads

Different categories of calls:

- Support Methods (Initialization/Cleanup), Query Methods (System, Device), Control Methods (Device commands), Event Handling Methods, Error Reporting Methods

Supported on Tesla and Quadro product line
NVML Example (C Version)

```c
#include <stdio.h>
#include <nvml.h>

int main() {
    nvmlReturn_t result;
    unsigned int device_count, i;
    char version[80];
    result = nvmlInit();
    result = nvmlSystemGetDriverVersion(version, 80);
    printf("Driver version: %s \n\n", version);
    result = nvmlDeviceGetCount(&device_count);
    printf("Found %d device%s\n\n", device_count, device_count != 1 ? "s" : "");
    printf("Listing devices:\n");
```
for (i = 0; i < device_count; i++) {
    nvmlDevice_t device;
    char name[64];
    nvmlComputeMode_t compute_mode;
    result = nvmlDeviceGetHandleByIndex(i, &device);
    result = nvmlDeviceGetName(device, name, sizeof(name)/sizeof(name[0]));
    printf("%d. %s\n", i, name);
}
result = nvmlShutdown();

cc -o nvml_test nvml_test.c -lnvidia-ml -l.
$ ./nvml_test
Driver version: 304.64
Found 2 devices
Listing devices:
0. Tesla K20m    1. Tesla K20m
NVML Bindings

- Bindings expose the NVML feature set through the Perl and Python scripting languages
- Support the same environments as NVML
- Updated with each CUDA release and publicly available on
  - CPAN (http://search.cpan.org/~nvbinding/nvidia-ml-pl/ ) and
  - PYPI (http://pypi.python.org/pypi/nvidia-ml-py )
NVML Example (Python Version)

```python
#!/usr/bin/python
from pynvml import *

nvmlInit()

count = nvmlDeviceGetCount()
for index in range(count):
    h = nvmlDeviceGetHandleByIndex(index)
    print nvmlDeviceGetName(h)

gpu = nvmlDeviceGetHandleByIndex(0)

print "Current clock speed in MHz:" ,
nvmlDeviceGetClockInfo(gpu, NVML_CLOCK_SM)

print "Max SM Clock speed in MHz:" ,
nvmlDeviceGetMaxClockInfo(gpu, NVML_CLOCK_MEM)

print "Power Usage in milliwatts:" ,
nvmlDeviceGetPowerUsage(gpu)

nvmlShutdown()
```

$ ./nvml_test.py
Tesla K20m
Tesla K20m
Current clock speed in MHz: 324
Max Mem Clock speed in MHz: 2600
Power Usage in milliwatts: 15561
#!/usr/bin/perl -w
use nvidia::ml qw(:all);

nvmlInit();
($ret, $version) = nvmlSystemGetDriverVersion();
die nvmlErrorString($ret) unless $ret ==
    nvidia::ml::bindings::NVML_SUCCESS;
print "Driver version: " . $version . "\n";

($ret, $count) = nvmlDeviceGetCount();
die nvmlErrorString($ret) unless $ret ==
    nvidia::ml::bindings::NVML_SUCCESS;
print "Found " . $count . " devices\n";

for ($i=0; $i<$count; $i++) {
    ($ret, $handle) = nvmlDeviceGetHandleByIndex($i);
    next if $ret != nvidia::ml::bindings::NVML_SUCCESS;
    ($ret, $info) = nvmlDeviceGetMemoryInfo($handle);
    next if $ret != nvidia::ml::bindings::NVML_SUCCESS;
    $total = ($info->{"total"} / 1024 / 1024);
    print "Total Memory Device " . $i . ": $total MB\n";
}

nvmlShutdown();

$ ./nvml_test.pl
Driver version: 304.64
Found 2 devices
Total Memory Device 0: 4799.5625 MB
Total Memory Device 1: 4799.5625 MB
nvidia-smi is a cross-platform command line tool
Exposes the NVML feature set through easy-to-use interface
Intended for interactive use and, via XML output, for automation
Examples:
- `nvidia-smi -q` (Query attributes for all GPUs)
- `nvidia-smi -q -x` (Output in XML Format)
- `nvidia-smi --loop=120` (Continuously report query data)
Ganglia GPU Support

- NVIDIA GPU monitoring plugin for gmond available
  - https://github.com/ganglia/gmond_python_modules/tree/master/gpu/nvidia
Nagios / Icinga GPU Support

- GPU sensor monitoring plugin for Nagios / Icinga based on NVML Perl binding is developed by Georg Schoenberger
Other Third Party Tools

Bright Cluster Manager

HP Cluster Management Utility (CMU)
nvidia-healthmon

- Diagnostic tool for quick health check
- Suggest remedies to software and system configuration problems

Feature Set
- Basic CUDA and NVML sanity check
- Diagnosis of GPU failure-to-initialize problems
- Check for conflicting drivers (I.E. VESA)
- InfoROM validation
- Poorly seated GPU detection
- Check for disconnected power cables
- ECC error detection and reporting
- Bandwidth test

Coordination with job schedulers is needed as nvidia-healthmon creates a CUDA context (if GPUs are running in exclusive mode)

http://developer.nvidia.com/tesla-deployment-kit
nvidia-healthmon

- Tests are controllable via config files
- Can be configured to fail on cluster-wide inconsistencies
- Use cases
  - Cluster scheduler prologue / epilogue script
  - Designed to integrate into third party tools
  - After provisioning cluster nodes
  - Run directly, manually

Sample config.ini file:

```ini
[global]
devices.tesla.count = 1
drivers.blacklist = nouveau

[Tesla K20m]
bandwidth.warn = 5500
bandwidth.min = 4500
pci.gen = 2
pci.width = 16
```
$ ./nvidia-healthmon -v -e
Loading Config: SUCCESS
Global Tests
    Black-Listed Drivers: SUCCESS
    Load NVML: SUCCESS
    Load CUDA: SUCCESS
    NVML Sanity: SUCCESS
    Tesla Devices Count: SUCCESS
Global Test Results: 5 success, 0 errors, 0 warnings, 0 did not run
GPU 0000:02:00.0 #0 : Tesla K20m (Serial: 0333412010882)
    NVML Sanity: SUCCESS
    InfoROM: SUCCESS
    GEMINI InfoROM
        This GPU does not share a board with another GPU chip.
        Result: SKIPPED
    ECC: SUCCESS
CUDA Sanity
    GPU: Tesla K20m

Compute Capability: 3.5
    Amount of Memory: 5032706048 bytes
    ECC: Enabled
    Number of SMs: 13
    Core Clock: 705 MHz
    Watchdog Timeout: Disabled
    Compute Mode: Default
    Result: SUCCESS
PCIe Maximum Link Generation: SUCCESS
PCIe Maximum Link Width: SUCCESS
PCI Bandwidth
    Host-to-GPU pinned memory bandwidth: 5881.894531 MB/s
    GPU-to-host pinned memory bandwidth: 6368.273926 MB/s
    Bidirectional pinned memory bandwidth: 10947.803711 MB/s
    Result: SUCCESS
Memory
    Allocated 4900807791 bytes (97.3%)
    Result: SUCCESS
Device Results: 8 success, 0 errors, 0 warnings, 1 did not run
Out-of-Band API

- Out-of-band API provides an interface before OS boot or driver load
- Integration into Lights Out Management
- Minimizes performance jitter
- Provides a subset of in-band NVML functionality
  - ECC
  - Power Draw
  - Temperature
  - Static info – Serial number, UUID
- BMC can control and monitor GPU
  - Control system fans based on GPU temperature
- Requires system vendor integration
GPU Management and Control
GPU Compute Modes

Control whether individual or multiple compute applications may run on the GPU (nvidia-smi –c <n>)

- **DEFAULT** compute mode
  - Multiple host threads can use the device at the same time
- **EXCLUSIVE_THREAD** compute mode
  - Only one host thread can use the device at any given time
- **PROHIBITED** compute mode:
  - No host thread can use the device
- **EXCLUSIVE_PROCESS** compute mode:
  - Only one context is allowed per device, usable from multiple threads at a time

Note: nvidia-smi –c settings do not persist across reboots or driver installs; they must be set at every boot
Causes driver to maintain a persistent connection to the GPU
- Faster, more consistent job startup
- Not preserved between reboots
- Boot scripts to set persistence mode for all GPUs in a system
  nohup nvidia-smi -pm 1
- Default: Persistence mode is disabled
GPU UUID

- UUID is the NVIDIA preferred mechanism to identify a GPU
  - Board serial number is shared by all GPU chips on a single board
  - GPU index is not guaranteed to remain constant

- NVML (nvmlDeviceGetUUID) and nvidia-smi report UUIDs for all CUDA capable GPUs (R304 drivers and later)

```
$ nvidia-smi -L
GPU 0: Tesla K20m (UUID: GPU-89050949-9e07-beb6-8271-250d7a7341f7)
GPU 1: Tesla K20m (UUID: GPU-08e6a4d4-1cd6-0bf8-ae68-0893d7ce218)
```
InfoROM

- InfoROM is a small, persistent store of configuration and state data for the GPU

Configuration checksum
- Makes it easy to verify that two GPUs have the same configuration
- Does not cover OS settings like persistence mode
- Example:
  - GPU 0 has ECC mode set to off
  - GPU 1 has ECC mode set to on
  - The InfoROM Configuration Checksum for GPU 0 will not match the checksum for GPU 1

- InfoROM verification integrity with `nvmlDeviceValidateInforom()` (exposed in nvidia-healthmon)
GPU Power Management
GPU Power Management

- NVIDIA GPUs have the ability to regulate power draw and thermals via active clock/voltage management
- This is done automatically, but can be directed by users in some cases
- Kepler provides much enhanced support vs. Fermi
  - Set power limit
  - Set fixed maximum clocks
  - Query performance limiting factors
Set Power Limit (Kepler only)

- Limit the amount of power GPU can consume
- Set power budgets and power policies
- Exposed in NVML and nvidia-smi
  - Example: Limit power to 85 Watts
    - nvidia-smi –pl 85
Set Applications Clocks (Kepler only)

Set maximum clocks that compute and graphics applications

Examples:
- Query supported clocks: `nvidia-smi -q -d SUPPORTED_CLOCKS`
- Set clocks for applications: `nvidia-smi -ac 2000, 800` (requires root access)
- Reset clocks: `nvidia-smi -rac`

Overridden by out-of-spec events (power, temperature)

Fixed performance when multiple GPUs operate in lock step
- Equivalent Performance
- Reliable Performance
- Save Power
Query Clock Throttle Reasons

GPU clocks will adjust based on environment and may be lower than the maximum if:
- GPU is idle
- Limited by software defined clock limit (eg. set by nvidia-smi --applications--clocks)
- Limited by software power limit (eg. set by nvidia-smi --power-limit)
- Limited by hardware limiters (eg. temperature)

Useful to understand GPU performance

```
$ nvidia-smi -q
....
Clocks Throttle Reasons
   Idle                     : Active
   User Defined Clocks     : Not Active
   SW Power Cap            : Not Active
   HW Slowdown             : Not Active
   Unknown                 : Not Active
```
GPU Operation Mode

- Allows to reduce power usage and optimize GPU throughput by disabling GPU features
  - Only supported on Kepler GK110 based K20/K20X (not on C-Class)
- Requires a reboot to change (might be removed in the future)
- Modes:
  - All on – All features are on (including graphics capabilities)
  - Compute – Running only compute tasks
  - Low Double Precision – Running graphics applications that don’t require high bandwidth double precision
GPU Job Scheduling
GPU Job Scheduling and Resource Management

- Slurm
- IBM Platform LSF
- Altair PBS Professional
- Grid Engine
- Moab/Torque
- Open Grid Scheduler
Requirements for GPU Job Scheduling

- Maximize the utilization of the GPU resources in the Cluster
- Handle different GPU configurations (different types, number of GPUs in a node, …)
- Map the GPU resources dependent on the hardware topology to get better performance and scalability
  - Eg. CPU-GPU pinning, GPU peer-to-peer communication (GPUDirect)
- Integrate features like CUDA Proxy
- Allow prologue / epilogue scripts (eg. run nvidia-healthmon)
Scheduling specific GPUs

- The environment variable CUDA_VISIBLE_DEVICES can be used to select specific GPUs without changing the application code, e.g.
  - Setting CUDA_VISIBLE_DEVICES to 0 will expose the 1st physical device as the only device to an application (hide a second GPU)
  - Setting CUDA_VISIBLE_DEVICES to 1,0 will expose the first two physical devices but swap the order of their device indices: device 0 will become 1 and vice-versa

- Allows batch systems and resource manager control
hwloc

- hwloc utility discovers server topology
- Use API to choose CPU and GPU that are physically close
- Version 1.7 will add support for “nvml” OS devices such as “nvml0” and also improves the discovery of their PCIe link speed
  - Used pre-release version hwloc-1.7a1r5368
    - ./configure --prefix=$HOME --enable-libpci --enable-plugins=nvml
    - lstopo -v

```
........
PCI 10de:1028 (P#540672 busid=0000:84:00.0 class=0302(3D) link=8.00GB/s
PCIVendor="nVidia Corporation")
    GPU L#7 (Backend=NVML GPUVendor="NVIDIA Corporation" GPUModel="Tesla K20m"
NVIDIAUUID=GPU-08e6a4d4-1cd6-0bfb-ae68-0893d7cec218) "nvml1"
........
```
Machine (64GB)

```
NUMANode P#0 (32GB)
```

```
Socket P#0
L3 (15MB)
L2 (256KB)
L2 (256KB)
L2 (256KB)
L2 (256KB)
L2 (256KB)
L1d (32KB)
L1i (32KB)
L1i (32KB)
Core P#6
PU P#0
```

```
GPU 0 and GPU1
```

```
GPU 0 and GPU1
```

```
PCI bandwidth
```

```
lstopo --output-format png topo.png
```
Skeleton for a MPI application

/* Read environment variables for local MPI rank */
… = atoi (getenv( "xxx_COMM_WORLD_RANK" ) );
… = atoi ( getenv( "xxx_COMM_WORLD_LOCAL_RANK" ) );
/* Discover the hardware topology */
hwloc_topology_init(…);
/* Select CPU and GPU based on the MPI rank and hardware topology */
cudaSetDevice(…);
hwloc_set_cpubind(…);
/* Clean up hardware topology library */
hwloc_topology_destroy(…);
/* Initialize MPI */
MPI_Init(…);
/* Body of MPI computation */
/* Terminate MPI */
MPI_Finalize();
/* Delete the CUDA context */
cudaDeviceReset();
Summary

- With the increasing number of large GPU clusters the management and monitoring is getting more important.
- NVIDIA is providing a management and monitoring API (NVML) which is used by NVIDIA and Third Party Tools.
- Power management features provide reliable performance and power savings.
- Topology aware scheduling can increase the performance significantly.
Thank you.
Questions?

Axel Koehler
Sr. Solution Architect HPC
akoehler@nvidia.com