
py3nvml Documentation

Release 0.2.1

Fergal Cotter

Oct 23, 2018

Contents:

1	py3nvml	1
1.1	Requires	1
1.2	Installation	1
1.3	Utils	2
1.4	Regular Usage	4
1.5	Release Notes (for pynvml)	6
1.6	COPYRIGHT	7
1.7	LICENSE	7
2	Indices and tables	9

CHAPTER 1

py3nvml

Python 3 compatible bindings to the NVIDIA Management Library. Can be used to query the state of the GPUs on your system. This was ported from the NVIDIA provided python bindings [nvidia-ml-py](#), which only supported python 2. I have forked from version 7.352.0. The old library was itself a wrapper around the [NVIDIA Management Library](#).

In addition to these NVIDIA functions to query the state of the GPU, I have written a couple functions/tools to help in using gpus (particularly for a shared gpu server). These are:

- A function to ‘restrict’ the available GPUs by setting the `CUDA_VISIBLE_DEVICES` environment variable.
- A script for displaying a differently formatted nvidia-smi.

See the Utils section below for more info.

1.1 Requires

Python 3.5+.

1.2 Installation

From PyPi:

```
$ pip install py3nvml
```

From GitHub:

```
$ pip install -e git+https://github.com/fbcotter/py3nvml#egg=py3nvml
```

Or, download and pip install:

```
$ git clone https://github.com/fbcotter/py3nvml
$ cd py3nvml
$ pip install .
```

1.3 Utils

(Added by me - not ported from NVIDIA library)

1.3.1 grab_gpus

You can call the `grab_gpus(num_gpus, gpu_select, gpu_fraction=.95)` function to check the available gpus and set the `CUDA_VISIBLE_DEVICES` environment variable as need be. It determines if a GPU is available by checking if the amount of free memory is below memory-usage is above/equal to the `gpu_fraction` value. The default of .95 allows for some small amount of memory to be taken before it deems the gpu as being 'used'.

I have found this useful as I have a shared gpu server and like to use tensorflow which is very greedy and calls to `tf.Session()` grabs all available gpus.

E.g.

```
import py3nvm1
import tensorflow as tf
py3nvm1.grab_gpus(3)
sess = tf.Session() # now we only grab 3 gpus!
```

Or the following will grab 2 gpus from the first 4 (and leave any higher gpus untouched)

```
py3nvm1.grab_gpus(num_gpus=2, gpu_select=[0,1,2,3])
sess = tf.Session()
```

This will look for 3 available gpus in the range of gpus from 0 to 3. The range option is not necessary, and it only serves to restrict the search space for the `grab_gpus`.

You can adjust the memory threshold for determining if a GPU is free/used with the `gpu_fraction` parameter (default is 1):

```
# Will allocate a GPU if less than 20% of its memory is being used
py3nvm1.grab_gpus(num_gpus=2, gpu_fraction=0.8)
sess = tf.Session()
```

This function has no return codes but may raise some warnings/exceptions:

- If the method could not connect to any NVIDIA gpus, it will raise a `RuntimeWarning`.
- If it could connect to the GPUs, but there were none available, it will raise a `ValueError`.
- If it could connect to the GPUs but not enough were available (i.e. more than 1 was requested), it will take everything it can and raise a `RuntimeWarning`.

1.3.2 get_free_gpus

This tool can query the gpu status. Unlike the default for `grab_gpus`, which checks the memory usage of a gpu, this function checks if a process is running on a gpu. For a system with N gpus, returns a list of N booleans, where the nth value is True if no process was found running on gpu n. An example use is:

```
import py3nvm1
free_gpus = py3nvm1.get_free_gpus()
if True not in free_gpus:
    print('No free gpus found')
```

1.3.3 py3smi

I found the default *nvidia-smi* output was missing some useful info, so made use of the *py3nvml/nvidia_smi.py* module to query the device and get info on the GPUs, and then defined my own printout. I have included this as a script in *scripts/py3smi*. The print code is horribly messy but the query code is very simple and should be understandable.

Running `pip install` will now put this script in your python's bin, and you'll be able to run it from the command line. Here is a comparison of the two outputs:

```
phdnets fbc23@mario ~/r/f/d/tests develop nvidia-smi
Thu May 17 11:10:51 2018
```

NVIDIA-SMI 390.25					Driver Version: 390.25				

GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.		
=====									
0	GeForce GTX 1080	Off	00000000:04:00.0	Off			N/A		
26%	41C	P2	37W / 180W	7885MiB / 8119MiB	0%		Default		

1	GeForce GTX 1080	Off	00000000:05:00.0	Off			N/A		
28%	43C	P2	40W / 180W	7457MiB / 8119MiB	0%		Default		

2	GeForce GTX 1080	Off	00000000:08:00.0	Off			N/A		
26%	37C	P2	46W / 180W	2646MiB / 8119MiB	24%		Default		

3	GeForce GTX 1080	Off	00000000:09:00.0	Off			N/A		
26%	42C	P2	40W / 180W	2245MiB / 8119MiB	17%		Default		

4	GeForce GTX 1080	Off	00000000:83:00.0	Off			N/A		
30%	46C	P2	52W / 180W	2245MiB / 8119MiB	24%		Default		

5	GeForce GTX 1080	Off	00000000:84:00.0	Off			N/A		
29%	44C	P2	47W / 180W	2245MiB / 8119MiB	24%		Default		

6	GeForce GTX 1080	Off	00000000:87:00.0	Off			N/A		
26%	35C	P0	39W / 180W	0MiB / 8119MiB	0%		Default		

7	GeForce GTX 1080	Off	00000000:88:00.0	Off			N/A		
26%	34C	P0	39W / 180W	0MiB / 8119MiB	0%		Default		

Processes:								GPU Memory	
GPU	PID	Type	Process name	Usage					
=====									
2	22808	C	python	2235MiB					
3	23066	C	python	2235MiB					
4	4954	C	python	2235MiB					
5	5459	C	python	2235MiB					

```
phdnets fbc23@mario ~/r/f/d/tests develop py3smi -l 4
Thu May 17 11:16:46 2018
```

NVIDIA-SMI										Driver Version: 390.25	
GPU	Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage		GPU-Util	Compute	M.		
0	26%	40C	P2	37W / 180W	7885MiB /	8120MiB	0%	Default			
1	27%	43C	P2	40W / 180W	7457MiB /	8120MiB	0%	Default			
2	26%	36C	P2	47W / 180W	2646MiB /	8120MiB	21%	Default			
3	26%	41C	P2	45W / 180W	2245MiB /	8120MiB	26%	Default			
4	29%	46C	P2	50W / 180W	2245MiB /	8120MiB	26%	Default			
5	28%	43C	P2	46W / 180W	2245MiB /	8120MiB	24%	Default			
6	26%	34C	P0	40W / 180W	0MiB /	8120MiB	0%	Default			
7	26%	33C	P0	39W / 180W	0MiB /	8120MiB	0%	Default			

Processes:						GPU Memory
GPU	Owner	PID	Uptime	Process Name		Usage
2	jv365	22808	53:35	python		2235MiB
3	jv365	23066	53:17	python		2235MiB
4	jv365	4954	01:21:22	python		2235MiB
5	jv365	5459	01:20:39	python		2235MiB

For py3smi, you can specify an update period so it will refresh the feed every few seconds. I.e., similar to `watch -n5 nvidia-smi`, you can run `py3smi -l 5`.

You can also get the full output (very similar to `nvidia-smi`) by running `py3smi -f` (this shows a slightly modified process info pane below).

1.4 Regular Usage

Visit [NVML reference](#) for a list of the functions available and their help. Also the script `py3smi` is a bit hacky but shows examples of me querying the GPUs for info.

(below here is everything ported from `pynvml`)

```
from py3nvml.py3nvml import *
nvmlInit()
print("Driver Version: {}".format(nvmlSystemGetDriverVersion()))
# e.g. will print:
#   Driver Version: 352.00
deviceCount = nvmlDeviceGetCount()
for i in range(deviceCount):
    handle = nvmlDeviceGetHandleByIndex(i)
    print("Device {}: {}".format(i, nvmlDeviceGetName(handle)))
# e.g. will print:
#   Device 0 : Tesla K40c
#   Device 1 : Tesla K40c
```

(continues on next page)

(continued from previous page)

```
nvmlShutdown()
```

Additionally, see `py3nvml.nvidia_smi.py`. This does the equivalent of the `nvidia-smi` command:

```
nvidia-smi -q -x
```

With

```
import py3nvml.nvidia_smi as smi
print(smi.XmlDeviceQuery())
```

1.4.1 Differences from NVML

The `py3nvml` library consists of python methods which wrap several NVML functions, implemented in a C shared library. Each function's use is the same with the following exceptions:

1. Instead of returning error codes, failing error codes are raised as Python exceptions. I.e. They should be wrapped with exception handlers.

```
try:
    nvmlDeviceGetCount()
except NVMLError as error:
    print(error)
```

2. C function output parameters are returned from the corresponding Python function as tuples, rather than requiring pointers. Eg the C function:

```
nvmlReturn_t nvmlDeviceGetEccMode(nvmlDevice_t device,
                                   nvmlEnableState_t *current,
                                   nvmlEnableState_t *pending);
```

Becomes

```
nvmlInit()
handle = nvmlDeviceGetHandleByIndex(0)
(current, pending) = nvmlDeviceGetEccMode(handle)
```

3. C structs are converted into Python classes. E.g. the C struct:

```
nvmlReturn_t DECLDIR nvmlDeviceGetMemoryInfo(nvmlDevice_t device,
                                               nvmlMemory_t *memory);

typedef struct nvmlMemory_st {
    unsigned long long total;
    unsigned long long free;
    unsigned long long used;
} nvmlMemory_t;
```

Becomes:

```
info = nvmlDeviceGetMemoryInfo(handle)
print("Total memory: {}MiB".format(info.total >> 20))
# will print:
#   Total memory: 5375MiB
print("Free memory: {}MiB".format(info.free >> 20))
```

(continues on next page)

(continued from previous page)

```
# will print:
#   Free memory: 5319MiB
print("Used memory: {}".format(info.used >> 20))
# will print:
#   Used memory: 55MiB
```

4. Python handles string buffer creation. E.g. the C function:

```
nvmlReturn_t nvmlSystemGetDriverVersion(char* version,
                                         unsigned int length);
```

Can be called like so:

```
version = nvmlSystemGetDriverVersion()
nvmlShutdown()
```

5. All meaningful NVML constants and enums are exposed in Python. E.g. the constant `NVML_TEMPERATURE_GPU` is available under `py3nvml.NVML_TEMPERATURE_GPU`

The `NVML_VALUE_NOT_AVAILABLE` constant is not used. Instead `None` is mapped to the field.

1.5 Release Notes (for pynvml)

Version 2.285.0

- Added new functions for NVML 2.285. See NVML documentation for more information.
- Ported to support Python 3.0 and Python 2.0 syntax.
- Added `nvidia_smi.py` tool as a sample app.

Version 3.295.0

- Added new functions for NVML 3.295. See NVML documentation for more information.
- Updated `nvidia_smi.py` tool - Includes additional error handling

Version 4.304.0

- Added new functions for NVML 4.304. See NVML documentation for more information.
- Updated `nvidia_smi.py` tool

Version 4.304.3

- Fixing `nvmlUnitGetDeviceCount` bug

Version 5.319.0

- Added new functions for NVML 5.319. See NVML documentation for more information.

Version 6.340.0

- Added new functions for NVML 6.340. See NVML documentation for more information.

Version 7.346.0

- Added new functions for NVML 7.346. See NVML documentation for more information.

Version 7.352.0

- Added new functions for NVML 7.352. See NVML documentation for more information.

1.6 COPYRIGHT

Copyright (c) 2011-2015, NVIDIA Corporation. All rights reserved.

1.7 LICENSE

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the NVIDIA Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 2

Indices and tables

- `genindex`
- `search`