Parallel Computing Laboratory
IT-300
Fall-2019

By
Dr. B. Neelima
National Institute of Technology Karnataka
(NITK), Surathkal

Week: 9th October-2019

This week the students are expected to install CUDA on their machines. A bsic installation guidance is given here. For any further system specific arrors the students are requested to solve them and learn to install CUDA.

After CUDA is installed the students are given instructions on executing CUDA programs and samples, as explained later

CUDA installation requires, NVIDIA driver to be installed. The following are some excerpts from my successful installation of CUDA. Students are requested to check their system and use similar instructions suitably:

Steps:

1. First check whether is there a CUDA capable graphics card on your machine:
   ```
   $ lspci | grep -i nvidia
   ```

2. Verify you have a supported version of linux
   ```
   $ uname -m && cat /etc/*release
   ```

3. Verify the system has gcc installed
   ```
   $ gcc --version
   ```

4. Install the NVIDIA graphics driver either through apt-get or run-file,as follows:
   a. **Via apt-get**
      i. For ubuntu 14.04.5 LTS, the latest version is 352. To install the driver, excute `sudo apt-get nvidia-352 nvidia-modprobe`, and then reboot the machine.
      ii. For ubuntu 16.04.3 LTS, the latest version is 375. To install the driver, excute `sudo apt-get nvidia-375 nvidia-modprobe`, and then reboot the machine.
      iii. NVIDIA driver should match as per the linux kernel version. Find the kernel version and supported driver

      OR

   b. **Via run-file**
   c. Remove Previous Installations (Important)

      ```
      sudo apt-get purge nvidia*

      sudo apt-get autoremove
      sudo dpkg -P cuda-repo-ubuntu1404 (your ubuntu version)
      ```

   d. Download the Driver

      ```
      cd ~
      wget http://us.download.nvidia.com/XFree86/Linux-
x86_64/384.69/NVIDIA-Linux-x86_64-384.69.run
      ```

384.69…Find the compatible driver for the GPU and linux version of your machine

e. Install Dependencies

```
sudo apt-get install build-essential gcc-multilib dkms
```

Installs all dependencies (if you do not want all, chose the one you are interested)

f. Create Blacklist for Nouveau Driver

Create a file at `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:

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

Then,

1. for Ubuntu 14.04 LTS, reboot the computer;
2. for Ubuntu 16.04 LTS, excute `sudo update-initramfs -u` and reboot the computer;

g. Stop lightdm/gdm/kdm

For Ubuntu 14.04 / 16.04, excuting `sudo service lightdm stop` (or use `gdm` or `kdm` instead of `lightdm`)

For Ubuntu 16.04 / Fedora / CentOS, excuting `sudo systemctl stop lightdm` (or use `gdm` or `kdm` instead of `lightdm`)

h. Executing the Runfile

```
cd ~
chmod +x NVIDIA-Linux-x86_64-384.69.run
sudo ./NVIDIA-Linux-x86_64-384.69.run --dkms -s
```

i. Check the Installation

`nvidia-smi` command will report all your CUDA-capable devices in the system.

5. Install CUDA (choose your version of CUDA or 7.5 will also serve the purpose. Sometimes the GPU that you have may not support the latest CUDA version)

Scripts for installing CUDA Toolkit are summarized below.

```
cd ~
wget
http://developer.download.nvidia.com/compute/cuda/7.5/Prod/local_instal
lers/cuda_7.5.18_linux.run
chmod +x cuda_7.5.18_linux.run
./cuda_7.5.18_linux.run --extract=$HOME
sudo ./cuda-linux64-rel-7.5.18-19867135.run
```

After the installation finishes, configure runtime library.

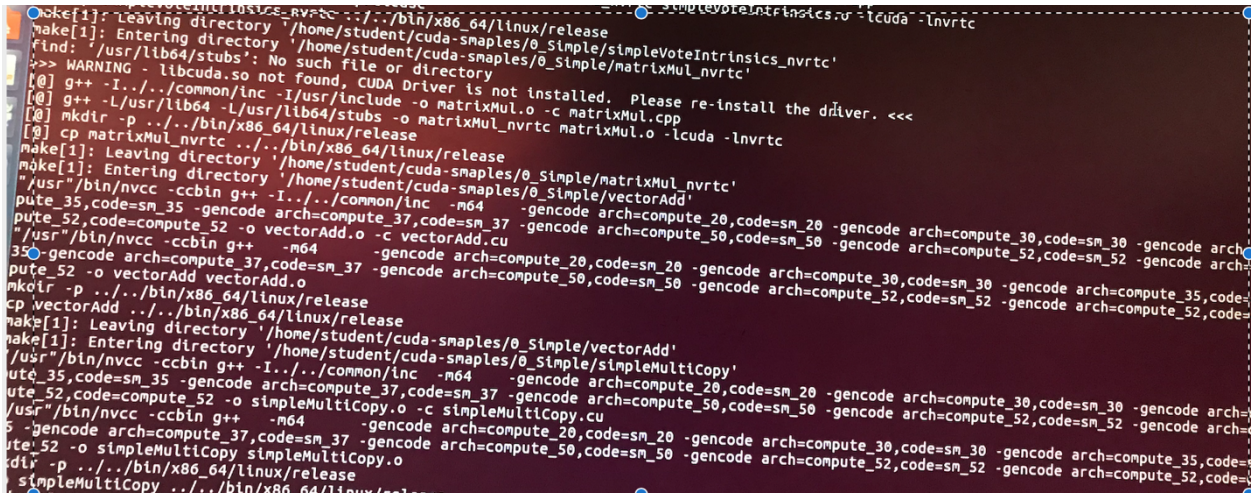
```
sudo bash -c "echo /usr/local/cuda/lib64/ >/etc/ld.so.conf.d/cuda.conf"

sudo ldconfig
```

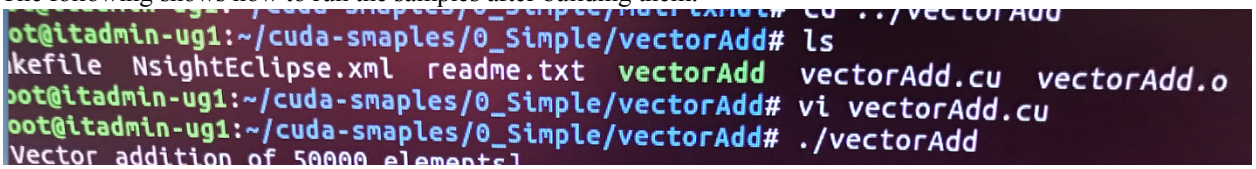
It is also recommended for Ubuntu users to append string `/usr/local/cuda/bin` to system file `/etc/environments` so that `nvcc` will be included in `$PATH`. This will take effect after reboot.

Install samples and make them so that you can run few examples. (try make -k)

The following figure shows an instance of building the samples:



The following shows how to run the samples after building them:



Run deviceQuery. From CUDA samples to know complete details of your device.

Similarly run the samples of your interest to get started with CUDA

Please report the status of your installation through an email to me and inform your evaluator on 9th October, 2019.

EXECUTING CUDA PROGRAMS:

After you finish installation and testing the samples, write your own codes in cuda and run them as per the following instructions:

Write the cuda program in c-style. The file is saved as: filename.cu

Compilation using nvidia: nvcc filename.cu. (use the required flags as per the requirement)
executing: ./a.out

Exercise 1: deviceQuery

1. Run the deviceQuery program from samples and send the screenshot of details of your device.

Exercise 2: Simple vector addition using CUDA

1. Write the following CUDA program and change the blocks and threads as per your device and see the variations in execution time. Report the same.

```
#define N 256
#include <stdio.h>

__global void vecAdd (int *a, int *b, int *c);

int main() {
 int a[N], b[N], c[N];
 int *dev_a, *dev_b, *dev_c;

 // initialize a and b with real values (NOT SHOWN)

 size = N * sizeof(int);

 cudaMalloc((void**)&dev_a, size);
 cudaMalloc((void**)&dev_b, size);
 cudaMalloc((void**)&dev_c, size);

 cudaMemcpy(dev_a, a, size,cudaMemcpyHostToDevice);

 cudaMemcpy(dev_b, b, size,cudaMemcpyHostToDevice);

 vecAdd<<<1,N>>>(dev_a,dev_b,dev_c);

 cudaMemcpy(c, dev_c, size,cudaMemcpyDeviceToHost);

 cudaFree(dev_a);
 cudaFree(dev_b);
 cudaFree(dev_c);

 exit (0);
}

__global void vecAdd (int *a, int *b, int *c) {
 int i = threadIdx.x;
 c[i] = a[i] + b[i];
}
```

2. Change the number of elements and/or number of threads. IF they are not exactly divisible, design your grid to pad the last block.

```
#define N 1618
#define T 1024 // max threads per block
#include <stdio.h>

__global void vecAdd (int *a, int *b, int *c);

int main() {
 int a[N], b[N], c[N];
 int *dev_a, *dev_b, *dev_c;

 // initialize a and b with real values (NOT SHOWN)

 size = N * sizeof(int);

 cudaMalloc((void**)&dev_a, size);
 cudaMalloc((void**)&dev_b, size);
 cudaMalloc((void**)&dev_c, size);

 cudaMemcpy(dev_a, a, size,cudaMemcpyHostToDevice);

 cudaMemcpy(dev_b, b, size,cudaMemcpyHostToDevice);

 vecAdd<<<(int)ceil(N/T),T>>>(dev_a,dev_b,dev_c);

 cudaMemcpy(c, dev_c, size,cudaMemcpyDeviceToHost);

 cudaFree(dev_a);
 cudaFree(dev_b);
 cudaFree(dev_c);

 exit (0);
}

__global void vecAdd (int *a, int *b, int *c) {
 int i = blockIdx.x * blockDim.x + threadIdx.x;
 if (i < N) {
  c[i] = a[i] + b[i];
 }
}
```