Pontificia Universidad JAVERIANA Cali

Facultad de Ingeniería y Ciencias
Carrera de Ingeniería de
Sistemas y Computación

PROGRAMACIÓN PARALELA

Installing the CUDA toolkit

The NVIDIA CUDA Toolkit provides a development environment for creating high-performance GPU-accelerated applications. It includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler, and a runtime library.

1. Pre-requisites:

We are assuming that the machine we'll use have Ubuntu 18.04 Bionic Beaver Linux installed and that we have privileged access to them. Besides, that the machine has a CUDA-enabled Graphics Processing Unit (GPU).

To find out if the machine has an NVIDIA GPU enter:

```
sudo lspci | grep -i nvidia
```

In the case of the machines in our cluster, they have an EVGA GeForce GT 730 2GB, and these are its features:

- Base Clock: 902 MHZ
- Memory Clock: 1800 MHz Effective
- CUDA Cores: 384
- Bus Type: PCI-E 2.0
- Memory Detail: 2048MB DDR3
- Memory Bit Width: 64 Bit
- Memory Speed: 1.1ns
- Memory Bandwidth: 14.4 GB/s

Then, we need to determine its CUDA Compute Capability and Google to find out which toolkit supports this board. In this case, CUDA 8.0 is the latest version that supports it. We can download the CUDA toolkit from the CUDA Toolkit Archive.

In our case, we need to download two files:
```
cuda_8.0.61_375.26_linux.run
cuda_8.0.61_patch2_linux.run
```

As always, before installing any drivers it is better to update the installed packages in the machine:

```
sudo apt-get update
sudo apt-get upgrade
```

It may be necessary to reboot the system and execute the first command to make sure that there are no additional packages that need to be installed. If need arises, we need to re-run the second command and reboot the machine again.

Linux distributions use free/libre software drivers for NVIDIA cards via the nouveau project. Unfortunately, sometimes these drivers are not compatible with the CUDA toolkit… and, this is our case.

Therefore, we need to install the proprietary NVIDIA driver. Even though the CUDA toolkit has video drivers, it is better to use the recommended video drivers for the current Linux version. This can be easily done once the packages have been updated: Usually we will see a notification and/or an icon, reminding us that restricted drivers are available; by clicking the icon we will be taken to a dialogue where we can choose which version we want to install. We will choose the recommended driver.

If for any reason, we can not use the graphical dialogue, we can use the command-line version:

```
sudo ubuntu-drivers devices
```

The output should be something like (excerpt):

```
vendor   : NVIDIA Corporation
model    : GK208B [GeForce GT 730]
    :
driver   : nvidia-driver-470 - distro non-free recommended
    :
driver   : xserver-xorg-video-nouveau - distro free builtin
```

As we can see above, the recommended driver is the `nvidia-driver-470`. Therefore, to install it enter:

```
sudo apt install nvidia-driver-470
```

2. CUDA Toolkit

Now we can proceed with the CUDA toolkit installation.

CUDA 8.0 comes with the following libraries (for compilation & runtime, in alphabetical order):

- cuBLAS – CUDA Basic Linear Algebra Subroutines library
- CUDART – CUDA Runtime library
- cuFFT – CUDA Fast Fourier Transform library
- cuRAND – CUDA Random Number Generation library
- cuSOLVER – CUDA based collection of dense and sparse direct solvers
- cuSPARSE – CUDA Sparse Matrix library
- NPP – NVIDIA Performance Primitives library
- nvGRAPH – NVIDIA Graph Analytics library
- NVML – NVIDIA Management Library
- NVRTC – NVIDIA Runtime Compilation library for CUDA C++

CUDA 8.0 comes with these other software components:

- nView – NVIDIA nView Desktop Management Software
- NVWMI – NVIDIA Enterprise Management Toolkit

First, we need to change the file attributes and then. unpack the installers:

```
sudo mkdir CUDA8
sudo cp cuda_8.0.61_375.26_linux.run CUDA8
sudo cd CUDA8
sudo chmod 755 cuda_8.0.61_375.26_linux.run
sudo cuda_8.0.61_375.26_linux.run --tar mxvf
```

Then, we need to copy a provided file to proceed with the installation:

```
sudo cp installUtils.pm /usr/lib/x86_64-linux-gnu/perl-base
```

We are now ready to install the toolkit:

```
sudo cd run_files
sudo cuda_linux64-rel-8.0.61-21551265.run
```

Proceed according to the instructions. Then, install the patch file:

```
sudo cuda_8.0.61_patch2_linux.run
```

We need to update the `.profile` file in our home directory.

Now, reboot the system.

Once logged in, execute:

```
nvidia-smi
```

Its output should reflect that a CUDA-enabled GPU has been detected.

3. Install required gcc and g++ versions

The gcc/g++ toolchains that come with Ubuntu 18.04 Bionic Beaver Linux are not supported by the CUDA 8.0 toolkit. We need to install the 4.8 version.

```
sudo apt install gcc-4.8 g++-4.8
```

Now we have two different versions (4.8 and 7) of both toolchains (gcc and g++). We will use the 7 version to compile MPI and OpenMP, just as we have been doing until now. But, we need to use the 4.8 version to compile CUDA programs.

For those cases, Linux provides a tool that allows us to switch between different versions of a given program using symbolic links: update-alternatives.

First, we will create the symlinks for gcc:

```
sudo update-alternatives --install /usr/local/bin/gcc gcc /usr/bin/gcc-7 7
sudo update-alternatives --install /usr/local/bin/gcc gcc /usr/bin/gcc-4.8 4
```

The last number in the previous command lines corresponds to the priority; the version with the higher number, takes precedence, and thus, is the default (preferred) version to run.

We can check which options are available for gcc:

```
sudo update-alternatives --query gcc
```

We can check which is the default version of gcc by typing:

```
gcc --version
```

When we need to switch to the version with the lower priority, we can execute:

```
sudo update-alternatives --config gcc
```

and select the appropriate version.

When we need to use the preferred version, we just type

```
sudo update-alternatives --auto gcc
```

Now, we need to create the symlinks for g++:

```
sudo update-alternatives --install /usr/local/bin/g++ g++ /usr/bin/g++-7 7
sudo update-alternatives --install /usr/local/bin/g++ g++ /usr/bin/g++-4.8 4
```

Check that both options are available.

4. Compiling and running CUDA programs

Remember that CUDA 8.0 requires using the gcc/g++ toolchains version 4.8. Then, before compiling any CUDA program, we need to use the corresponding `update-alternatives` command lines.

To execute a CUDA binary we just need to use its name in the command line.