**Cuda Compilation Utilizing**
**the NVIDIA GPU**
**Oct 19, 2017**

This document will essentially provide the reader with the understanding on how to use the CUDA 7.0 environment within the Electrical and Computer engineering network. Please note that the only room that presently has NVIDA GPU cards installed is ENG412. Yet this doesn't mean that students only have to work ENG412 to compile code to work with a GPU. Any workstation within the department will allow the students to compile code to work with a GPU. The final compiled application though must be executed on a system in ENG412 to access a GPU. Running the compiled application on any other system will fail. Students can still use the ssh command to connect from any workstation within the department to any workstation in ENG412 if they wish to execute their code.

There are two ways to setup your environment to use the cuda compiler. If you are logged into any workstation within the department you can select the Cuda 7 Compiler via the Applications menu as shown in Figure 1.0.

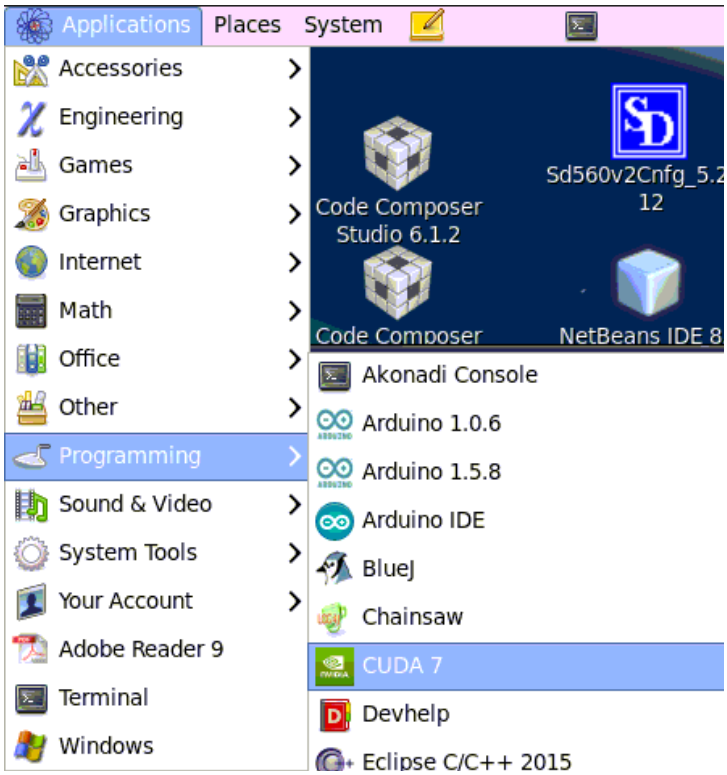


**Figure 1.0:** Accessing Cuda 7 via the Applications Menu

By clicking on the “CUDA 7” Applications menu, a terminal window will pop up providing you a configured environment and folder where you can compile C/C++ code within this window. The second option to access the same environment would be to execute the following command in a terminal window:

```
cuda_shell
```

Either way will get a terminal up and running and configured to work with the CUDA compiler. The first time you execute the environment (either through the applications menu or via the command terminal) the system will have to create a folder in your home directory called:

cuda-7.0

For example the first time I issued the cuda_shell command I got the following output:

```
jnaughto@slinux: ~$ cuda_shell

       *************** CUDA INITIALIZATION  *********************

     Welcome to the CUDA environment.  Please wait while we create a few
folders in your home directory.

In the future you will be directed to your CUDA work folder called:

            /home/faculty/jnaughto/cuda-7.0

 where you will compile cuda supported files.

Creating cuda work folder: /home/faculty/jnaughto/cuda-7.0 ... Done
Creating library and include links ... Done
Creating sample folders:
    0_Simple, 1_Utilities, 2_Graphics, 3_Imaging, 4_Finance,
    5_Simulations, 6_Advanced,  ... Done

   ************************* WARNING ****************************
   *     CUDA supported hardware installed on this computer...    *
   * You may be able to compile cuda code but you are not working *
   * working on a computer with a CUDA supported GPU.  Only       *
   * systems in ENG412 have CUDA supported GPUs.                  *
   ************************* WARNING ****************************


[jnaughto@slinux:~/cuda-7.0/samples]$
```

Notice here the warning. The cuda_shell was run on the workstation slinux which does not have a NVIDA GPU. If the system that you were working on had a NVIDA GPU the warning block above would not appear. Yet the folder was created and the student account would be left in the

~/cuda-7.0/samples folder. Once the folder has been created any future execution of the cuda_shell will produce:

```
****************** CUDA Work Enviornment  ********************

 Welcome to the CUDA environment.  The environment has
been setup for you to compile CUDA supported GPU programs.

Changing to existing cuda folder: /home/faculty/jnaughto/cuda-7.0 ...Done

   ************************* WARNING ****************************
   *     CUDA supported hardware installed on this computer...    *
   * You may be able to compile cuda code but you are not working *
   * working on a computer with a CUDA supported GPU.  Only       *
   * systems in ENG412 have CUDA supported GPUs.                  *
   ************************* WARNING ****************************


[jnaughto@slinux:~/cuda-7.0/samples]$
```

Inside the samples folder there are a number of CUDA sample programs that can be compiled.

```
[jnaughto@slinux:~/cuda-7.0/samples]$ ls
0_Simple     2_Graphics  4_Finance 6_Advanced  common
1_Utilities  3_Imaging      5_Simulations
```

For example under 0_Simiple/vectorAdd, this example can be compiled as follows:

```
[jnaughto@slinux:~/cuda-7.0/samples]$ cd 0_Simple/
[jnaughto@slinux:~/cuda-7.0/samples/0_Simple]$ cd vectorAdd

[jnaughto@slinux:~/cuda-7.0/samples/0_Simple/vectorAdd]$ ls
Makefile  NsightEclipse.xml  readme.txt  vectorAdd.cu

[jnaughto@slinux:~/cuda-7.0/samples/0_Simple/vectorAdd]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode
arch=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30 -gencode
arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode
arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode
arch=compute_52,code=compute_52 -o vectorAdd.o -c vectorAdd.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20
-gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode
arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o vectorAdd
vectorAdd.o
mkdir -p ../../bin/x86_64/linux/release
cp vectorAdd ../../bin/x86_64/linux/release
```

The last line will show that the vectorAdd binary exists in the present working directory and inside:

```
~/cuda7-0/samples/bin/x86_64/release
```

The student can test running this program by executing the command vectorAdd. In the case that the program is run on a system that does not support NVIDA GPUs the output will appear as:

```
[jnaughto@slinux:~/cuda-7.0/samples/0_Simple/vectorAdd]$ vectorAdd
[Vector addition of 50000 elements]
Failed to allocate device vector A (error code CUDA driver version is insufficient for
CUDA runtime version)!
```

This is because the system that we ran the vectorAdd program did not have a NVIDA GPU. Yet if the student ran the same program in one of the workstations in ENG412 the output would be the following:

```
[jnaughto@oshawa:~/cuda-7.0/samples/0_Simple/vectorAdd]$ vectorAdd
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

Under the sample folder 1_Utilities/deviceQuery there is an example program called **deviceQuery** there is a Makefile in this folder that the student can use to compile this program similar to the vectorAdd program that was compiled above. Once compiled the output of this program will provide specific information about the NVIDA GPU that can be useful:

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro 600"
  CUDA Driver Version / Runtime Version          8.0 / 7.0
  CUDA Capability Major/Minor version number:    2.1
  Total amount of global memory:                 964 MBytes (1010958336 bytes)
  ( 2) Multiprocessors, ( 48) CUDA Cores/MP:     96 CUDA Cores
  GPU Max Clock rate:                            1280 MHz (1.28 GHz)
  Memory Clock rate:                             800 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 131072 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65535), 3D=(2048,
2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
```

```
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (65535, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 7.0,
NumDevs = 1, Device0 = Quadro 600
Result = PASS
```

This would be a typical execution response.