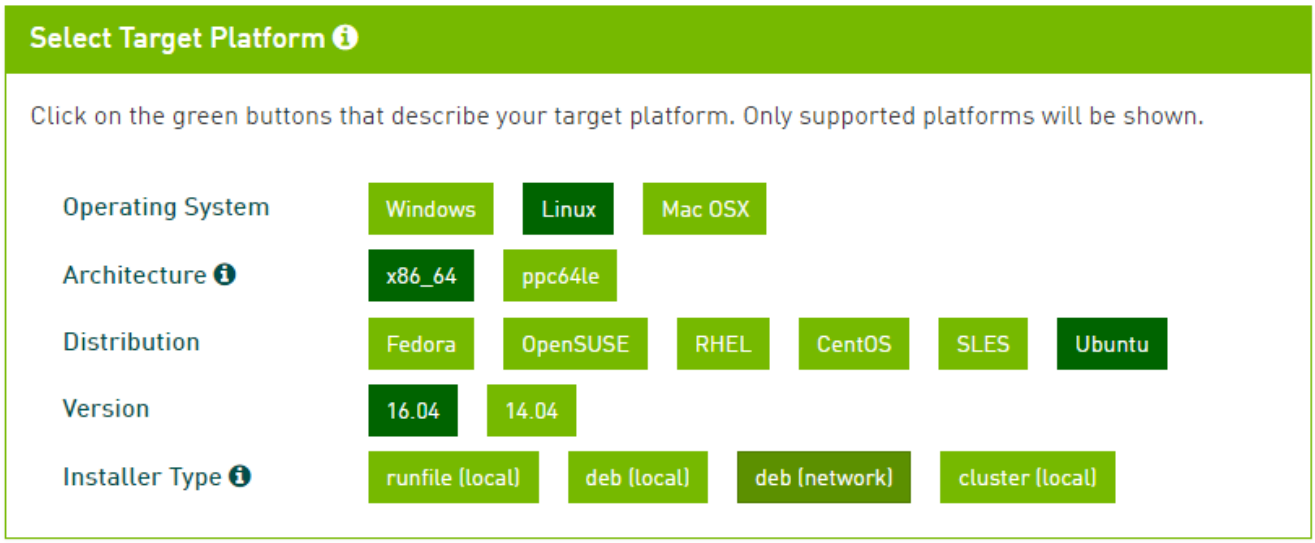
Guide to install NVIDIA CUDA 8.0 developer tools in Ubuntu 16.04 and execute a CUDA program in a remote GPU server

A. Install CUDA developer tools in your computer

1. Go to https://developer.nvidia.com/cuda-downloads and download the deb (network) installer for Ubuntu 16.04



Open a terminal, copy & paste the following commands, and accept the installation of the new software

```
$ sudo dpkg -i cuda-repo-ubuntu1604_*.deb
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install build-essential cuda nvidia-cuda-dev nvidia-cuda-toolkit nvidia-nsight
```

B. Create a new CUDA project in Nsight software for developing CUDA programs

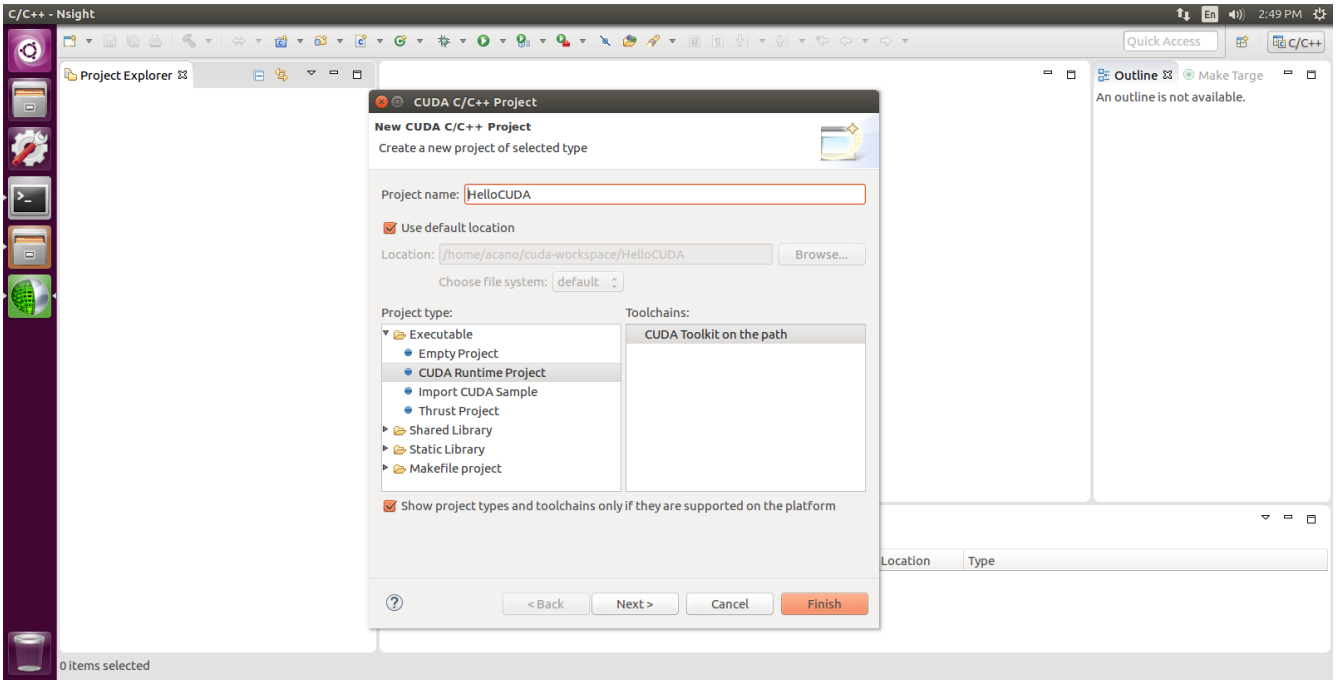
1. Open a terminal and open the Nsight software by typing

```
$ nsight
```

Use the default workspace (recommended), e.g. /home/username/cuda-workspace

2. Create a new project using File -> New -> CUDA C/C++ Project

3. Type a project name, and select CUDA Runtime Project as the project type, click Next

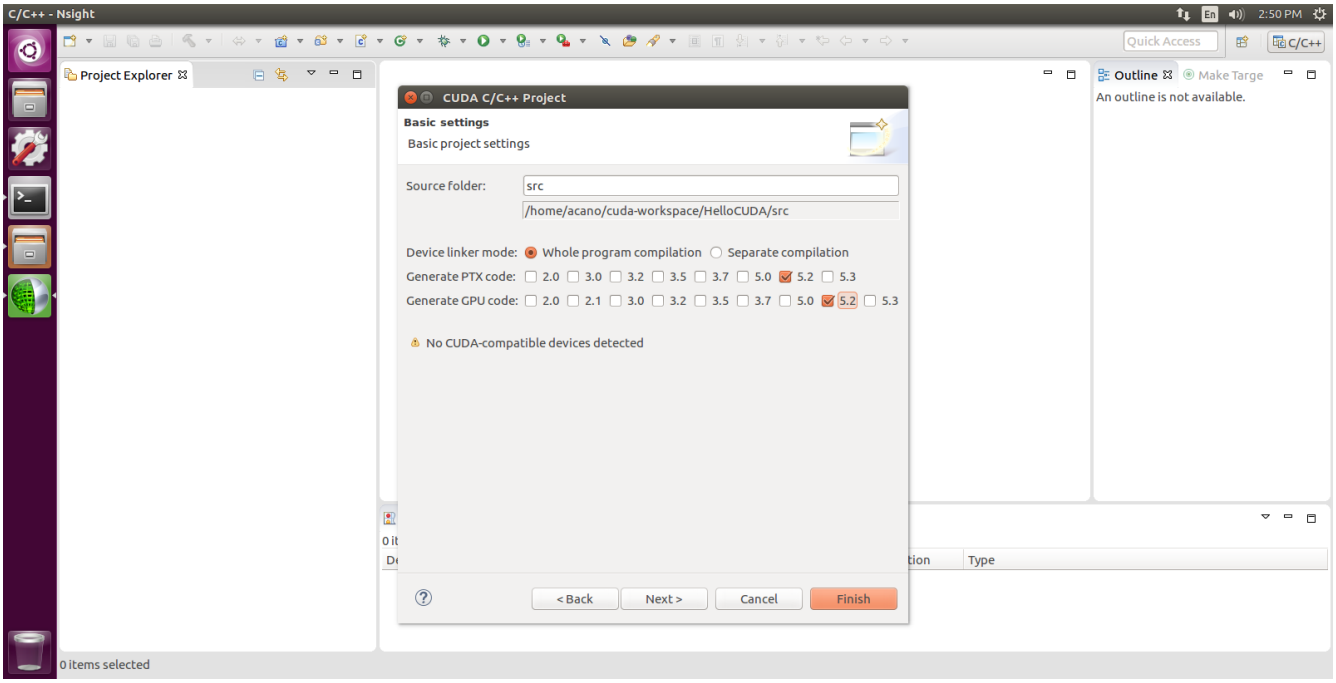


4. (Optional) Complete the information about author and copyright notice, click Next

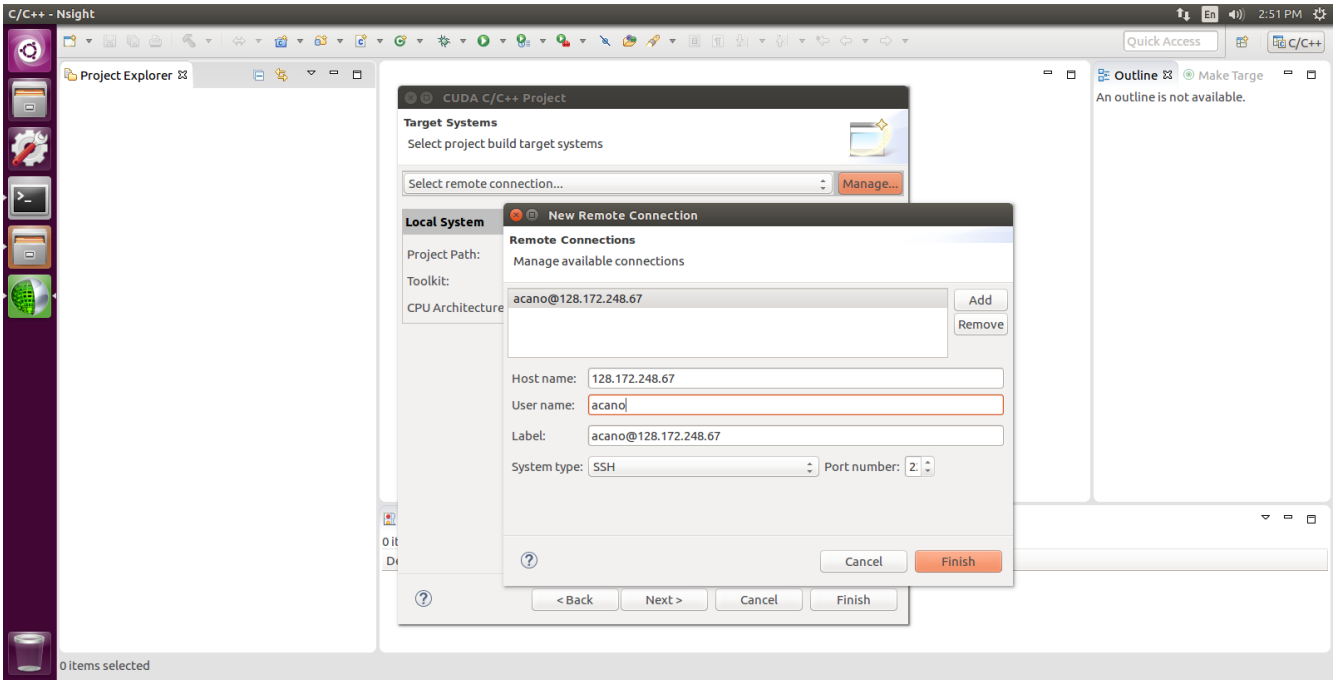
5. Select the project settings, tick 5.2 and untick 2.0 for both PTX and GPU code, click Next

If 5.2 is not available is because you are using another older installer for a previous release but it is still ok, then select the closest number < 5.2.

Note: this reduces the compilation time by forcing to compile only for the GPUs architecture we are using in the remote server (NVIDIA GTX 980s). Should you use your own GPUs make sure to select the proper compute capability according to the GPU architecture you have.



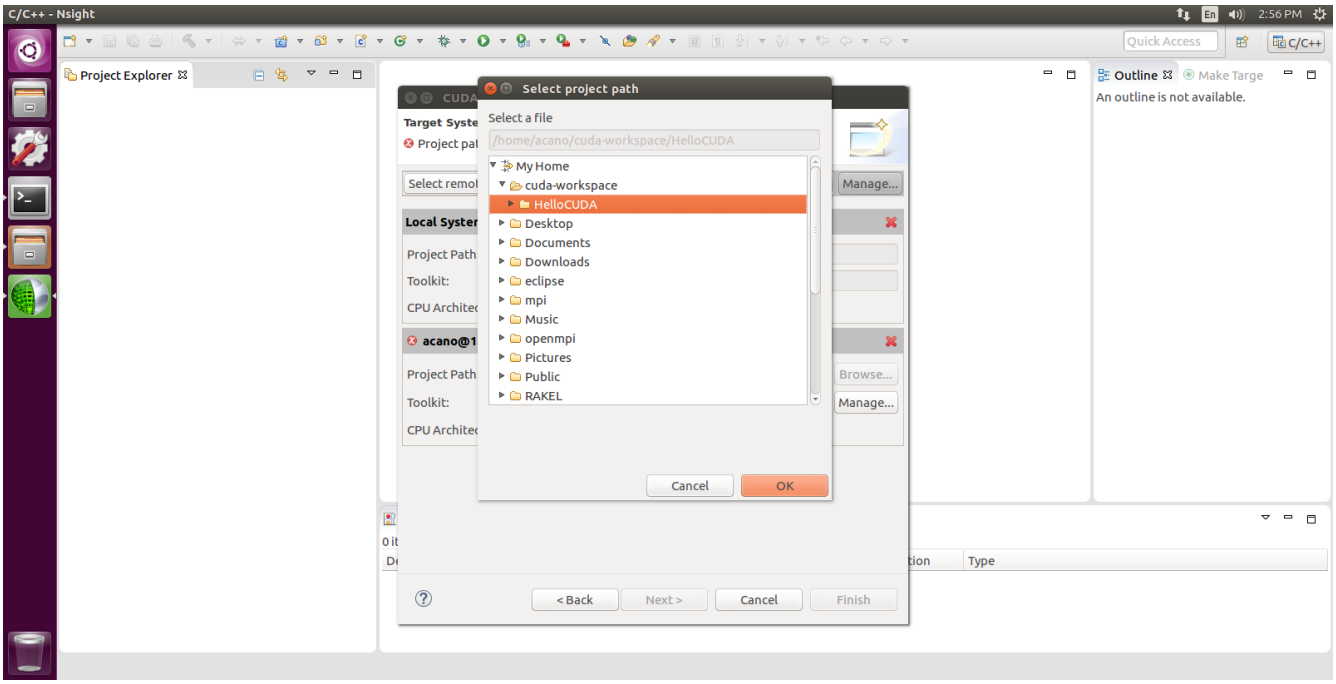
6. Create a remote connection, click Manage, and click Add to add a remote connection, use the server IP 172.18.199.200 as the hostname, and your VCU eID as your user name (same login as in ssh), click on Finish



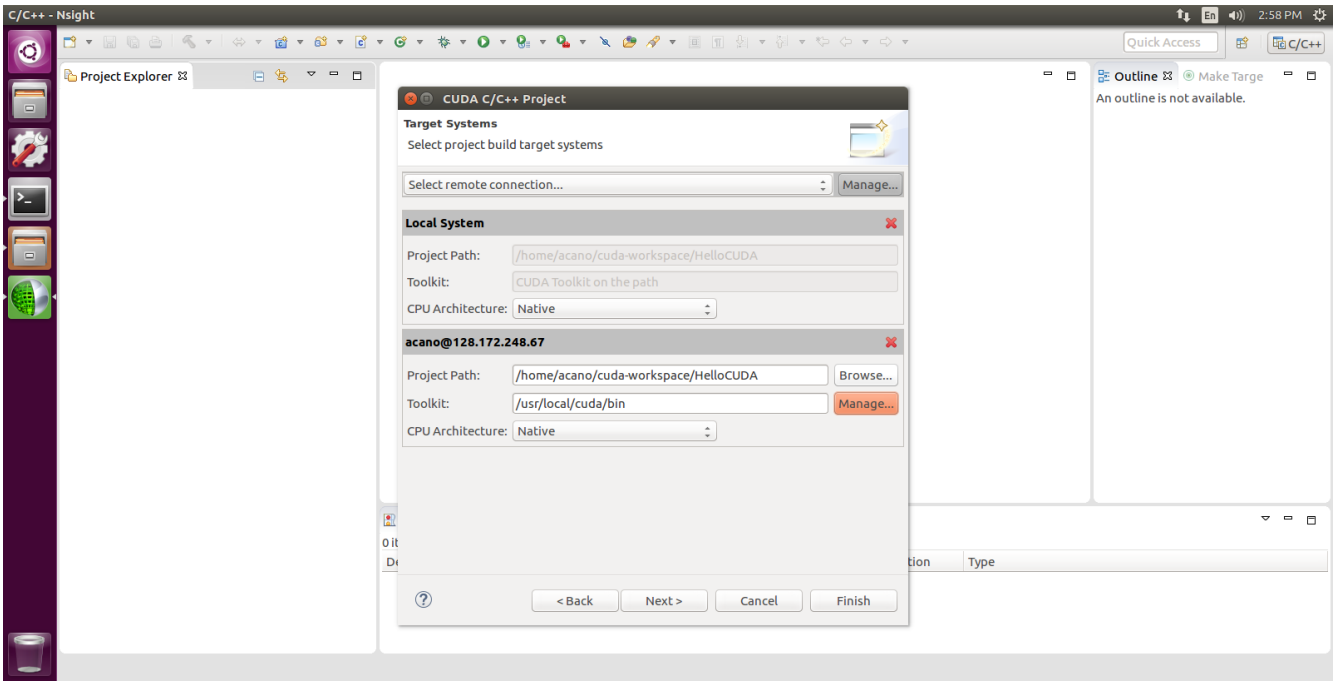
7. Open a terminal, connect to the GPU server using ssh, and make sure to create the paths for the project in the remote GPU server, e.g. using the HelloCUDA as project name

```
$ ssh 172.18.199.200 -l username
$ mkdir /home/username/cuda-workspace/
$ mkdir /home/username/cuda-workspace/HelloCUDA/
$ mkdir /home/username/cuda-workspace/HelloCUDA/Debug
$ mkdir /home/username/cuda-workspace/HelloCUDA/Release
```

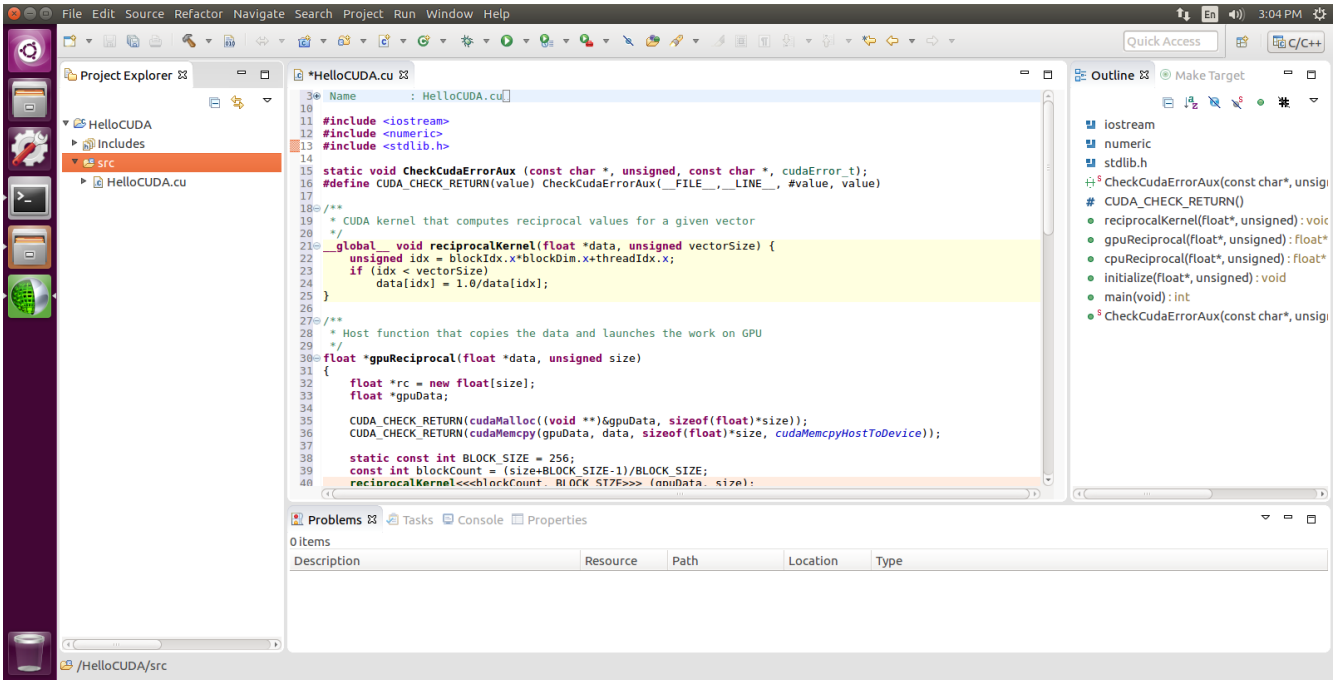
8. Back to Nsight, select the project path, click on Browse, type your password for your user in the server (VCU V number by default), double click to expand the tree, and point to the folders for the project you just created in the remote machine so that it is consistent with your localhost, for example My Home > cuda-workspace > HelloCUDA, will point to the path /home/username/cuda-workspace/HelloCUDA in the server and it is consistent to your project path in your local machine.



9. Select the Toolkit, click Manage, type in the toolkit path "/usr/local/cuda/bin/", click Finish



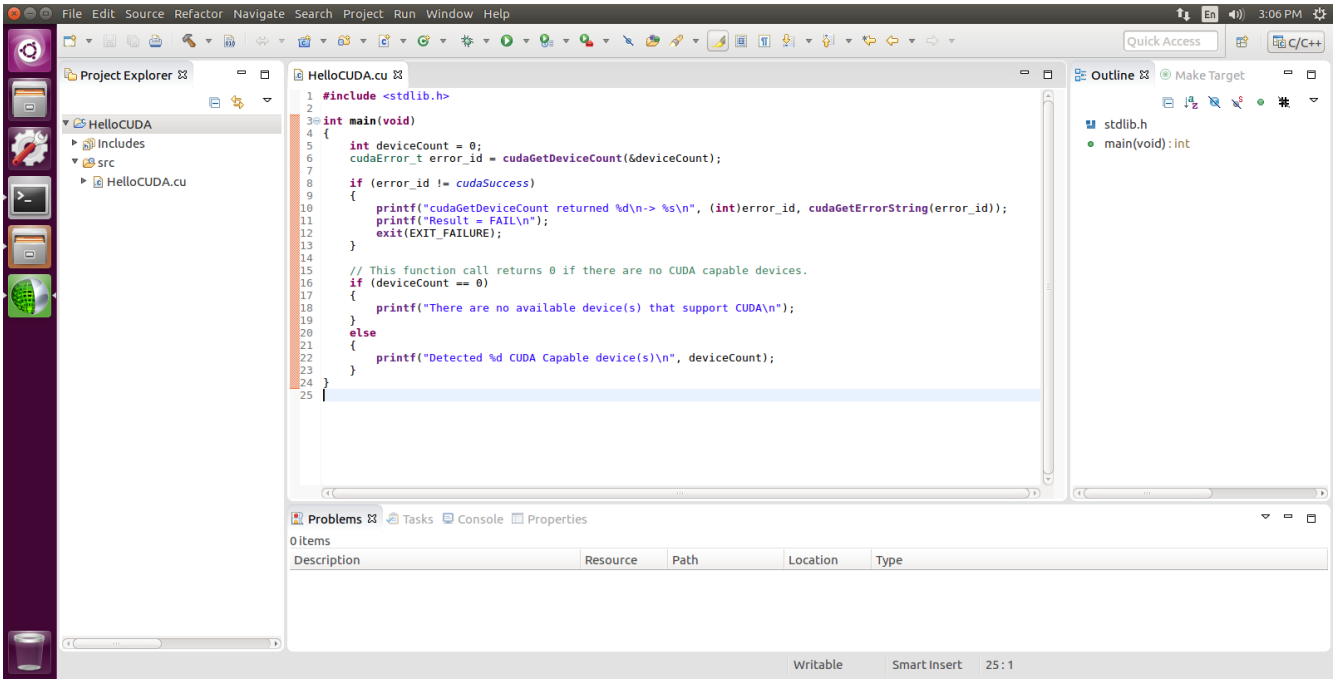
10. Click Finish to create the project, and this is how it looks. By default your code will be initialized with some example program.



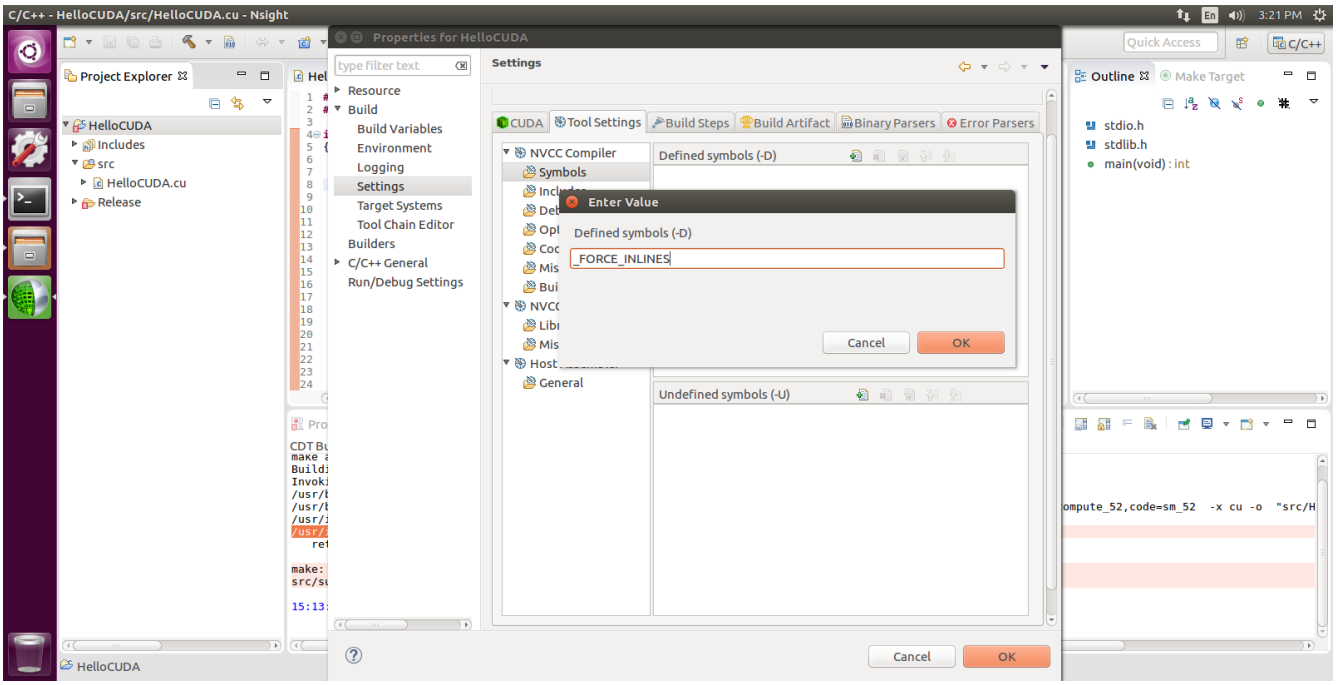
11. Delete the example code provided and copy & paste the following code to test the remote GPU

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int deviceCount = 0;
   cudaError_t error_id = cudaGetDeviceCount(&deviceCount);

   if (error_id != cudaSuccess)
   {
      printf("cudaGetDeviceCount returned %d\n-> %s\n", (int)error_id, cudaGetErrorString(error_id));
      printf("Result = FAIL\n");
      exit(EXIT_FAILURE);
   }

   // This function call returns 0 if there are no CUDA capable devices.
   if (deviceCount == 0)
   {
      printf("There are no available device(s) that support CUDA\n");
   }
   else
   {
      printf("Detected %d CUDA Capable device(s)\n", deviceCount);
   }
}
```

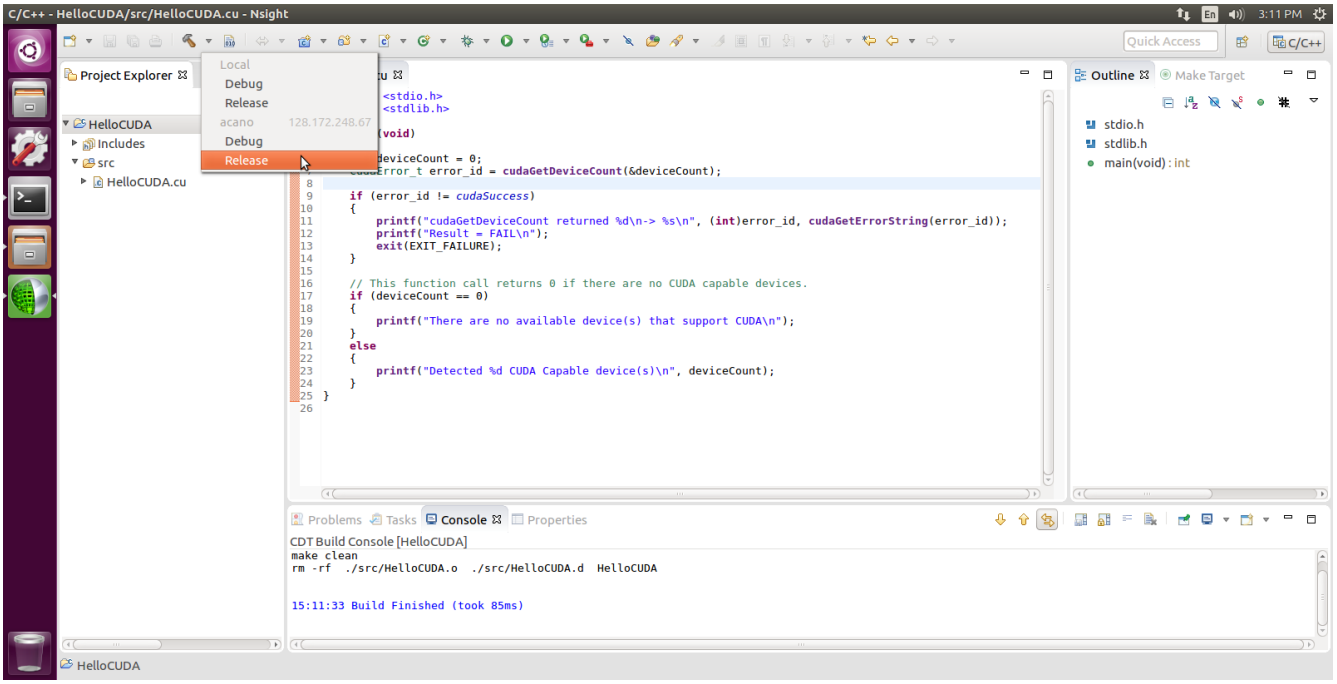
It now should something like this. Ok!, we're ready to compile the code.



12. (OPTIONAL) Users of Ubuntu 16.04 with CUDA 7.5 have to conduct an additional step to fix a compilation issue due to compiler versions incoherence. If you are installing CUDA 8.0 skip this step. To solve this, right click the project name in the explorer located left of the screen, select Properties to get the project properties, then go to Build -> Settings, then go to the Tool Settings tab, and for the NVCC compiler symbols, click on the green cross to add a Defined symbol, write _FORCE_INLINES and click OK for both windows to exit the project properties.

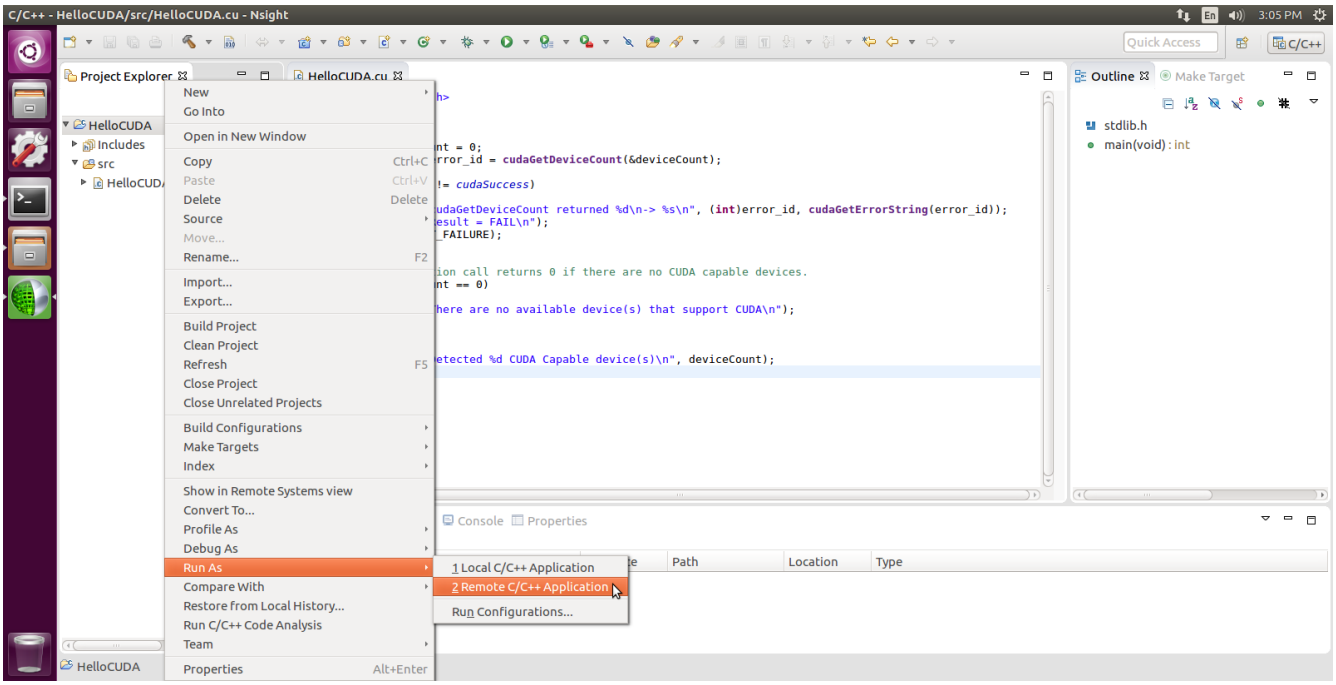


13. Compile the code for the remote GPU server, clicking in the triangle right of the hammer icon, then select the Release option for the remote server.

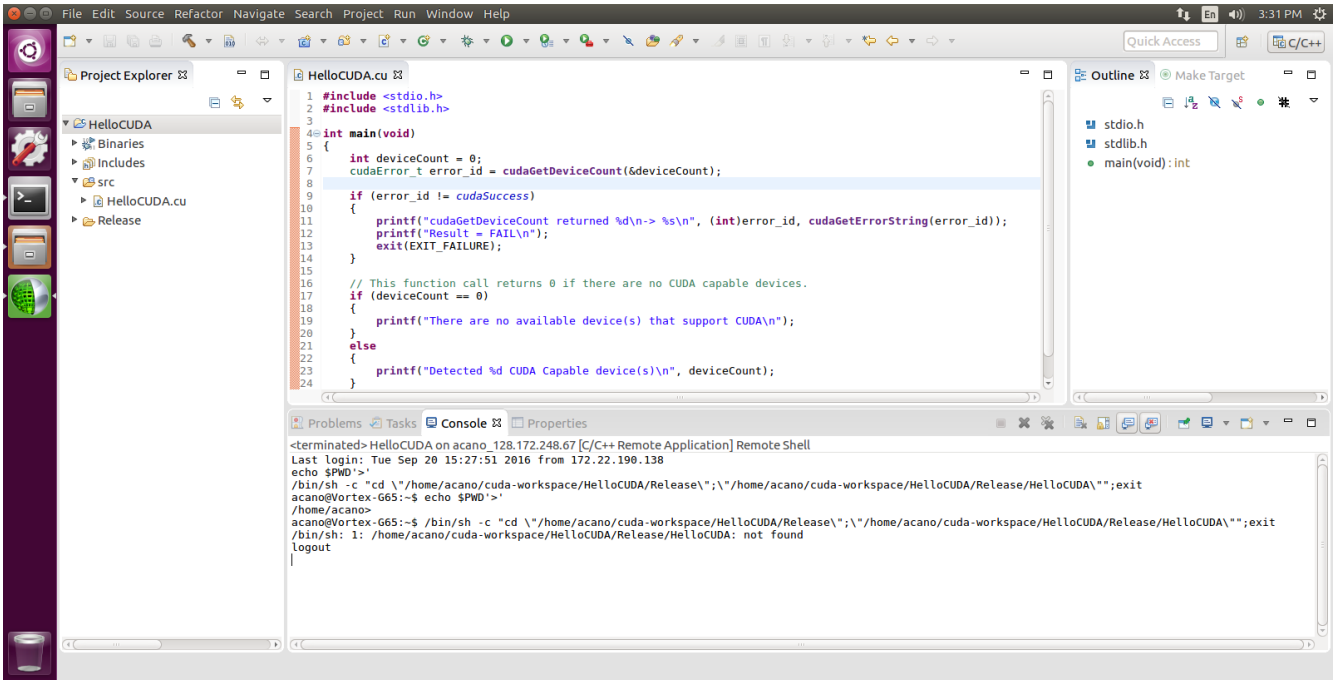


If you get a compilation error like */usr/include/string.h:652:42: error: 'memcpy' was not declared in this scope*, make sure to define the symbol as detail in step #12.

14. Now it is compiled, lets create the setup to run the program in the remote GPU server. Right click the project name in the explorer -> Run As -> Remote C/C++ Application and Select the remote connection for your user in the list, click Finish



You will get in the console something like this



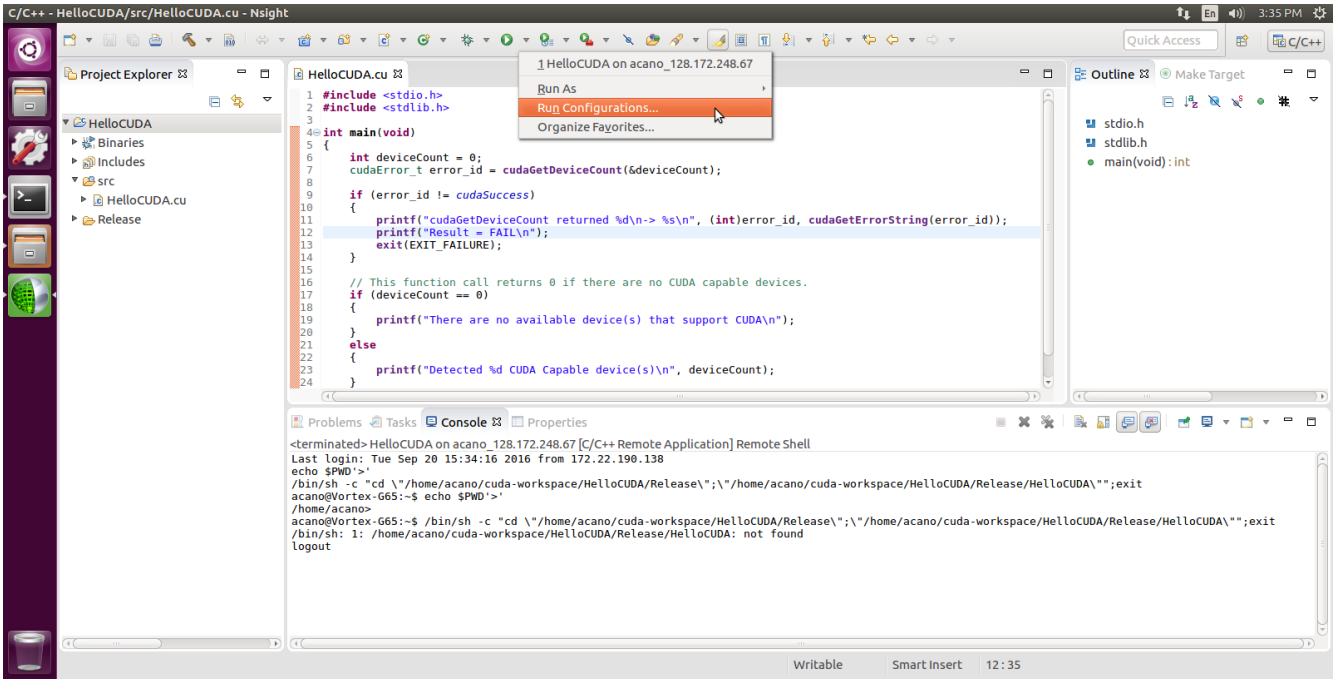
What it is trying to do is opening a ssh connection to the server and execute the remote compiled program, but you receive this error

"/bin/sh: 1: /home/acano/cuda-workspace/HelloCUDA/Release/HelloCUDA: not found"

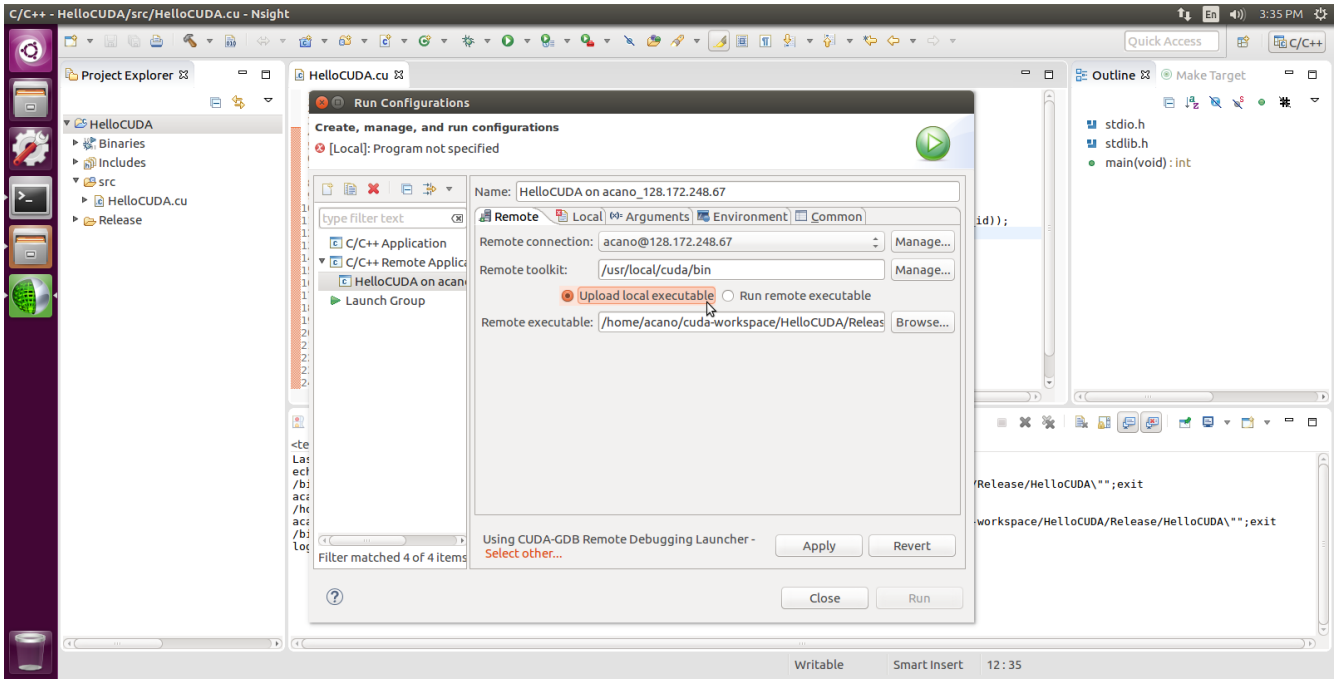
which simply means that the remote binary file does not exist, because we compiled the code in the localhost (your computer), but we still need to transfer the executable to the server! (captain obvious)

Rather than manually uploading to the server the binary file every time we edit the code, using e.g. filezilla or scp, (that would be terribly tedious!), we may force the automatic upload of the binary file every time we execute the program by editing the configuration of the runnable program. Keep reading.

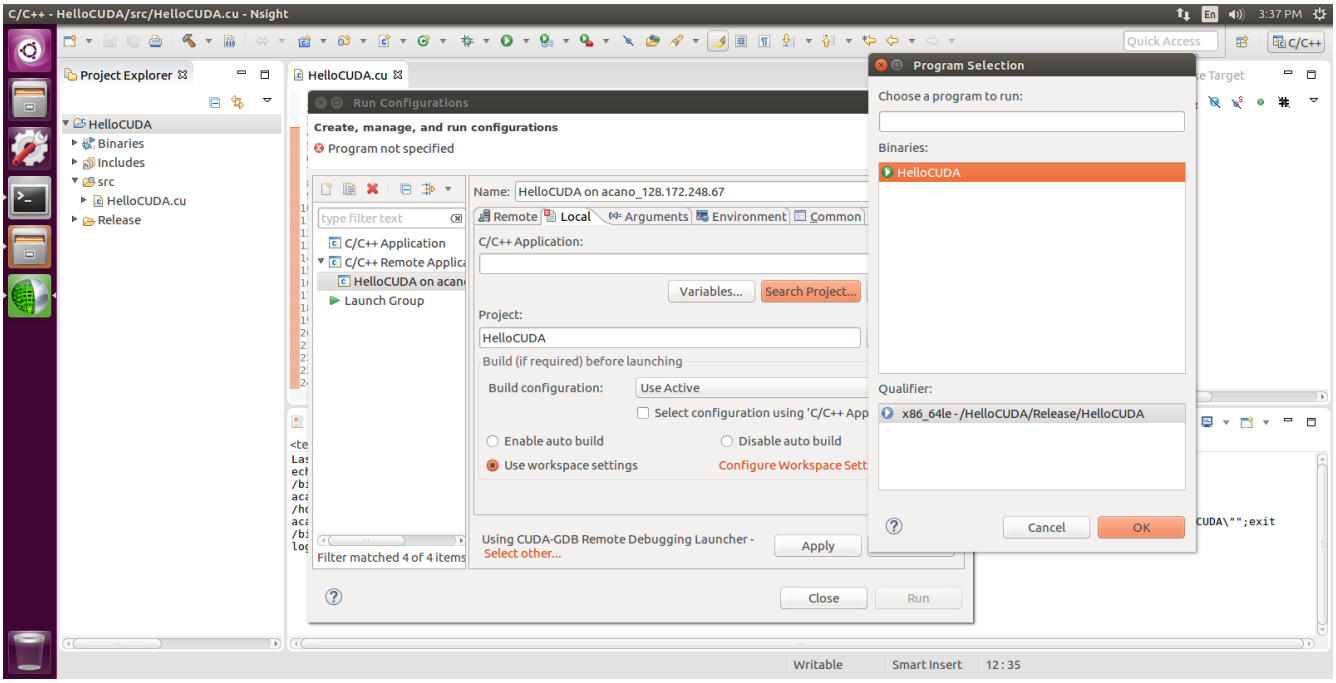
15. To edit the run configurations, click on the menu Run -> Run Configurations, or in the triangle next to the green Run icon.



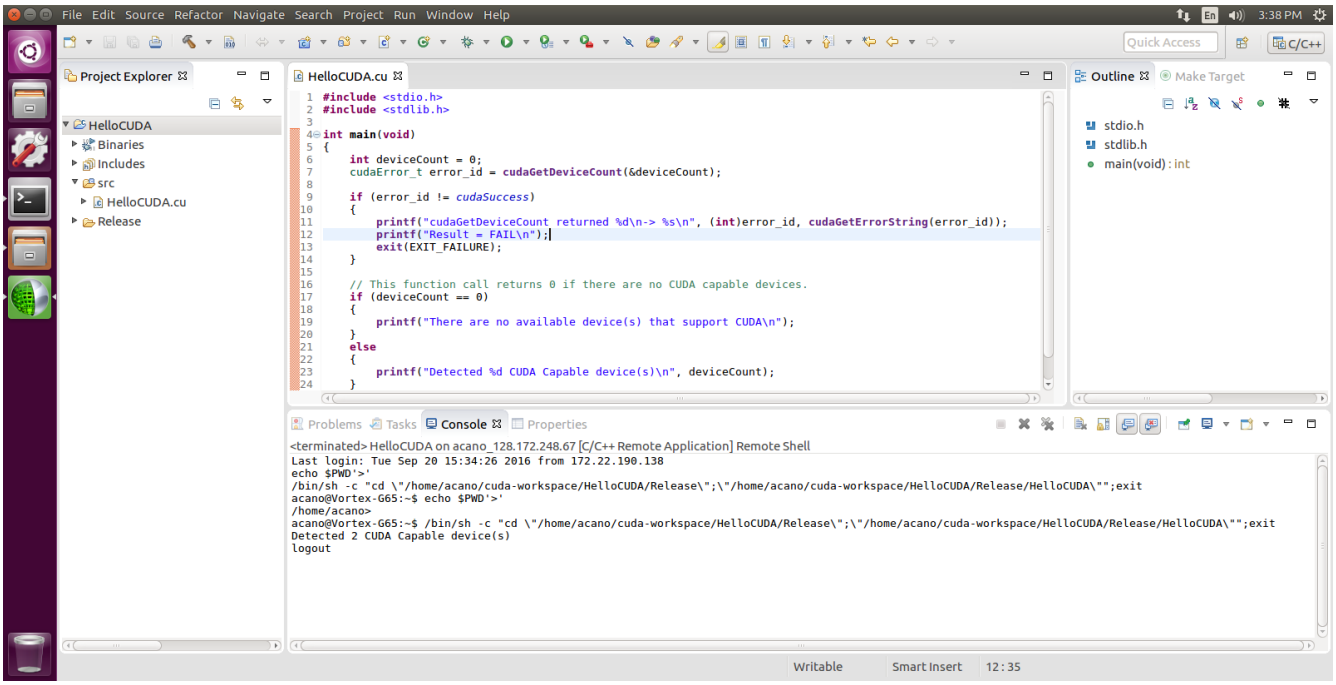
16. For the given project, Select "Upload local executable" in the Remote tab



17. Moreover, click to the "Local" tab, click "Search Project" in C/C++ Application, select the HelloCUDA program, and click OK. This way, we identify the executable file to be uploaded.



18. Finally, click "Apply", and then "Run"



It works!!

We received the message "Detected 2 CUDA Capable device(s)" from the remote GPU server. Awesome! It only took 11 pages! The good news is that this project setup need to be only done only once. Now, every time you edit the code and simply click on Run, it will be automatically compiled and transferred to the GPU server, and you will get the system output in your console! No one said developing code in localhost and executing remotely was easy!