This is the manual for coding in CUDA and make DLL files. The making DLL part applies to code not using CUDA as well.

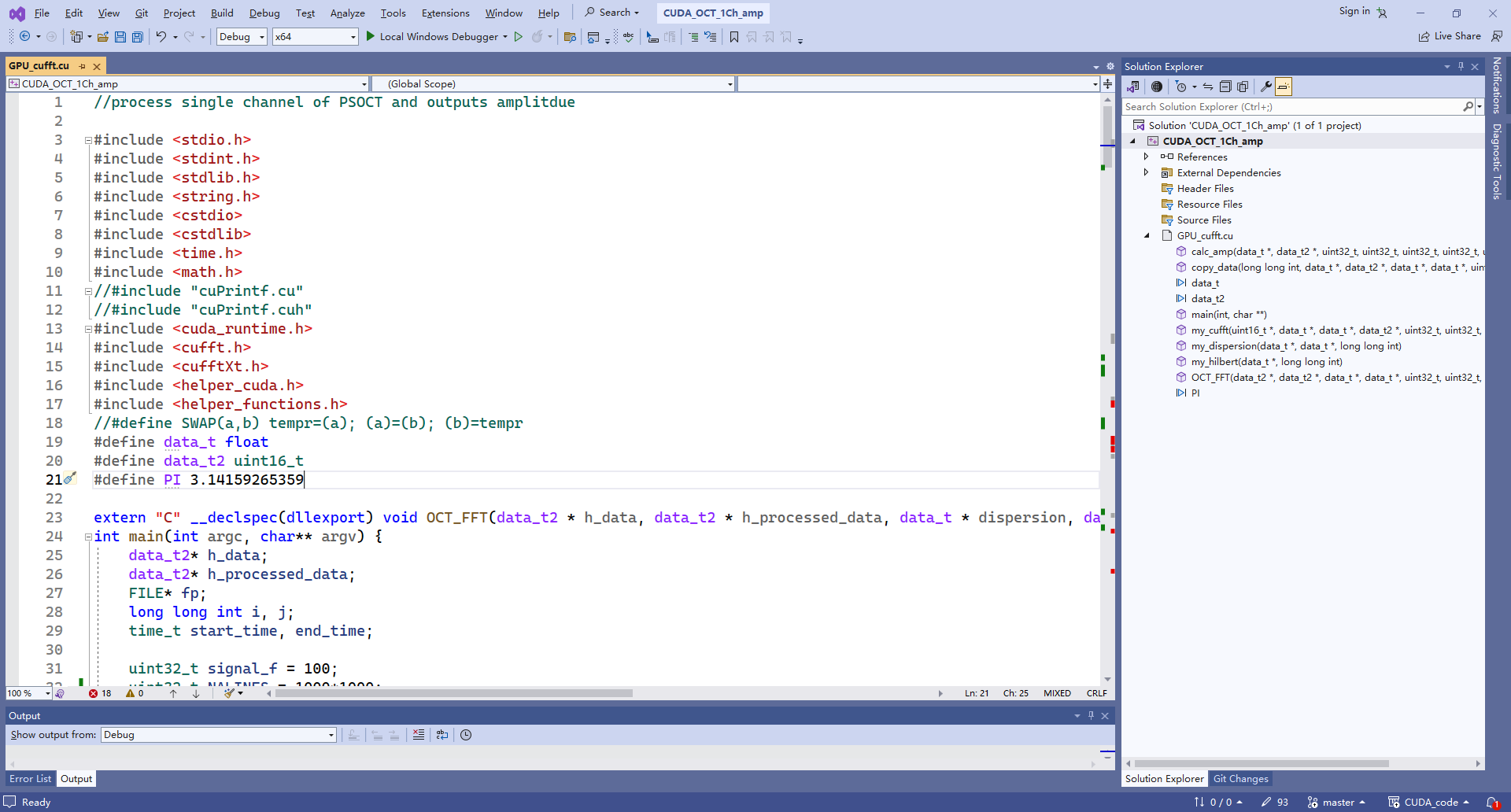
When it comes to CUDA, NEVER start from scratch. Instead, always start from a working example and modify it to suit your application. You can copy CUDA\_OCT\_TEMPLATE and start modifying from it.

This project was tested to successfully run on Visual studio 2017(changed to VS2022) community version, 64bit system. The GPU used was NVIDIA Quadro RTX4000 (NVIDIA GeForce RTX3060 tested). GPU driver was from CUDA Toolkit 10.1. (changed to 12.3, 64bit version I suppose)

Update in 2023: re-configured visual studio environment with CUDA toolkit in SUSTECH. Used visual studio 2022 and CUDA Toolkit 12.3, some modifications added.

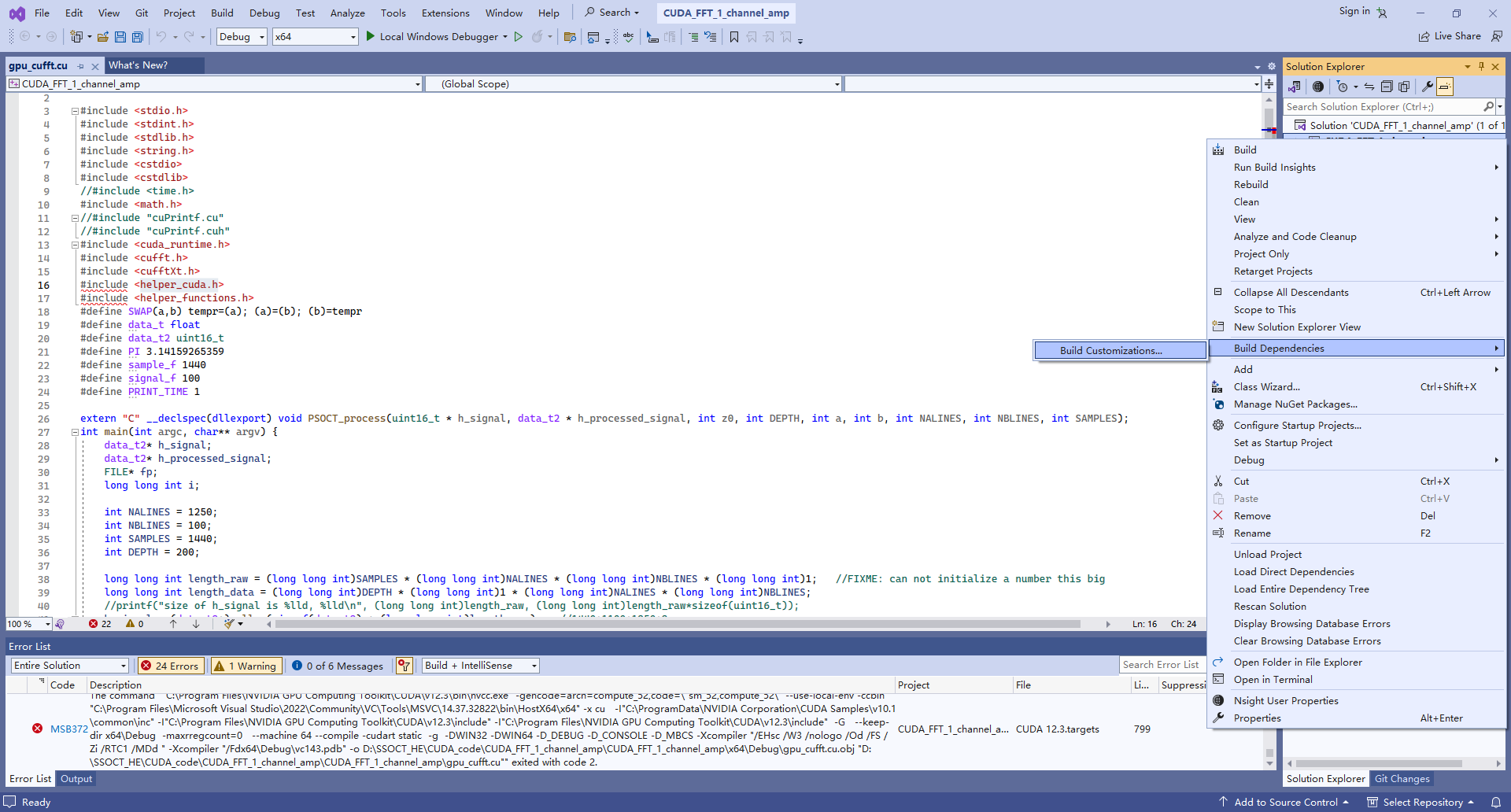
This example code should be ready to run. If it doesn’t, check the following configurations:

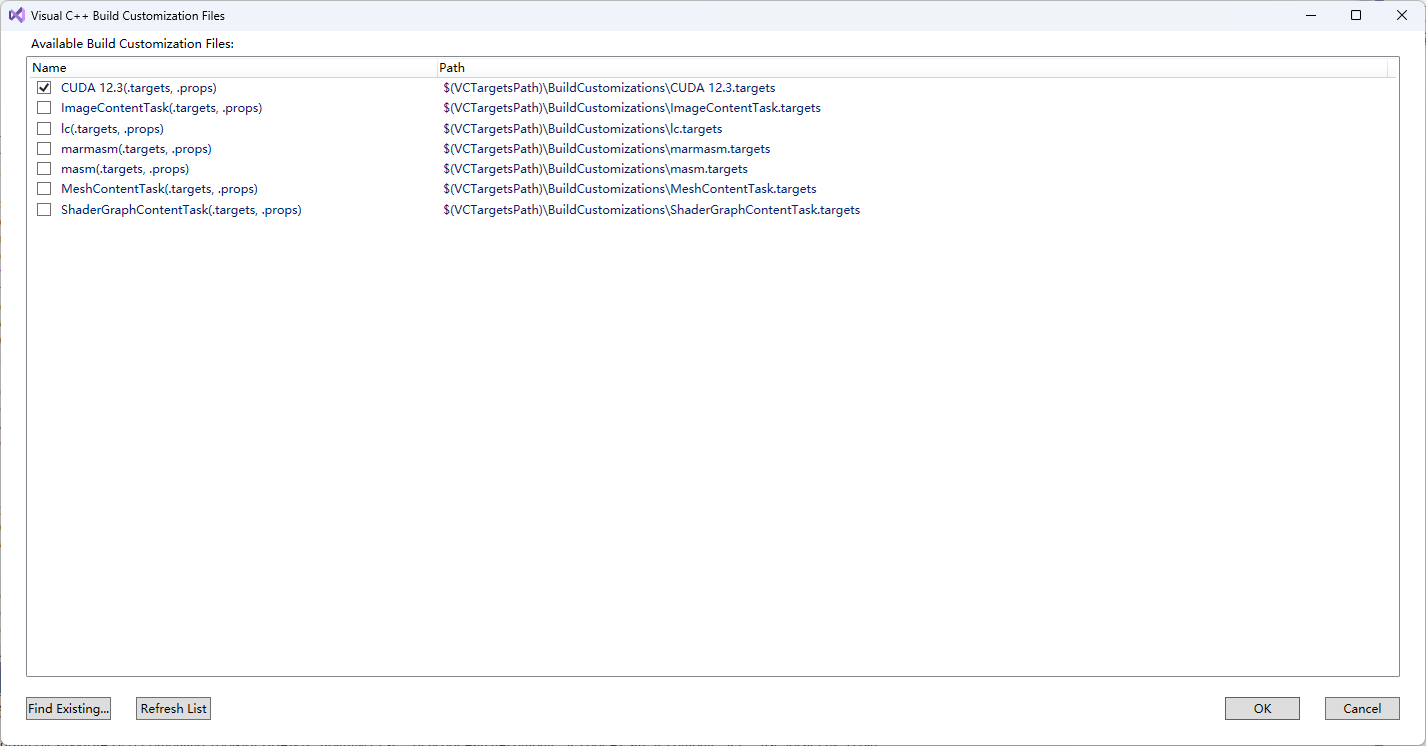
1. If you want to generate DLL file, put extern "C" \_\_declspec(dllexport) in front of the function declaration that you want to export to a DLL file. Make sure you put the declaration at the very top of the script, right after all includes



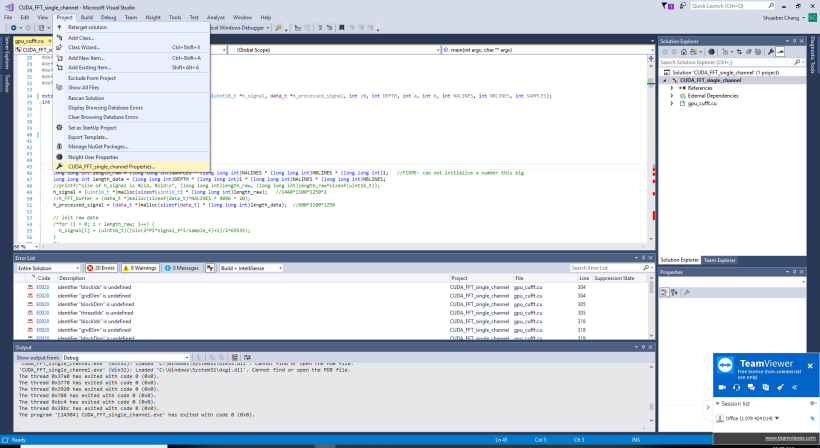
Steps 2,5,6,7 are for CUDA configurations, skip them if you do not need to use CUDA

1. Right click on project name in Solution Explorer ,Go to build customization and select CUDA, this adds CUDA toolkit to the project environment.

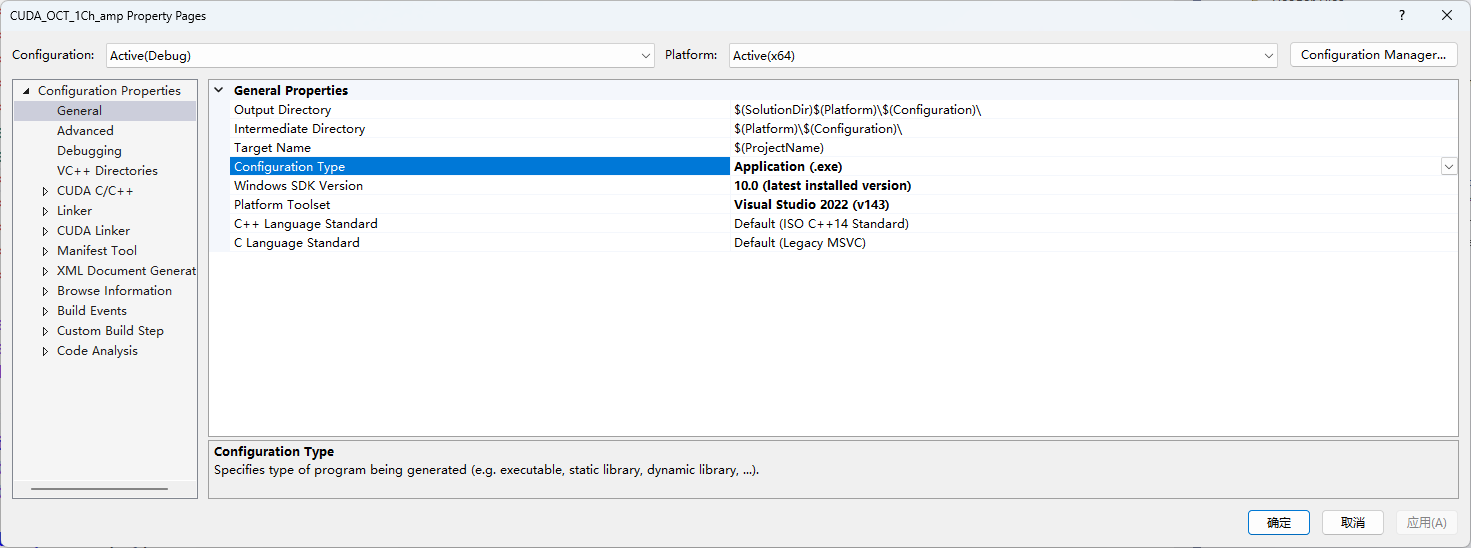




1. Left Click anywhere in the .cu file, then go to toolbar: Project-> <project name> Properties to make following changes. If you do not click the .cu file, there will be no <project name> Properties.

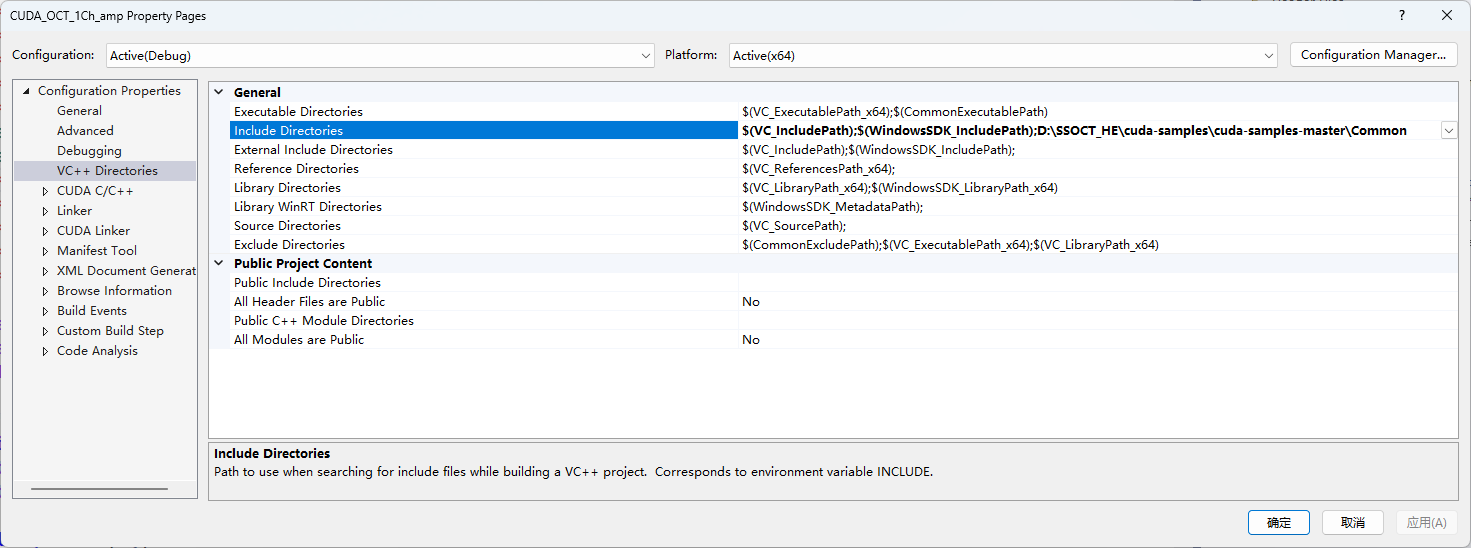


1. Go to Configuration Properties-> General-> Configuration Type, choose DLL if you want build a DLL file, or stay with .exe if you are coding in C/C++

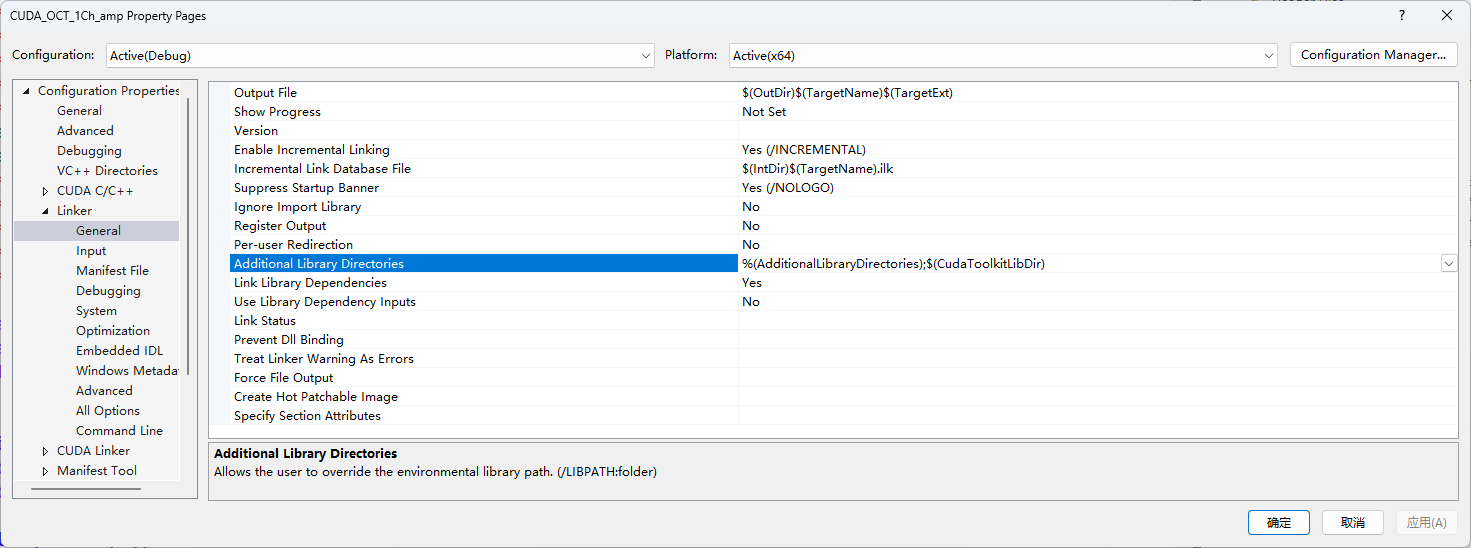


1. Go to Configuration Properties-> VC++ Directories-> Include Directories, add the CUDA example directory, use ; to separate directories

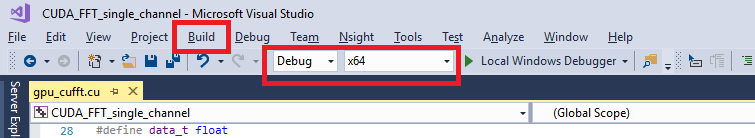
Since CUDA 12, CUDA samples are not included during installation, you need to go to github <https://github.com/nvidia/cuda-samples> to download it yourself.



1. Go to Configuration Properties-> Linker-> General-> Additional Library Directories, check if **$(CudaToolkitLibDir)** is there, if not, add it, use ; to separate different items



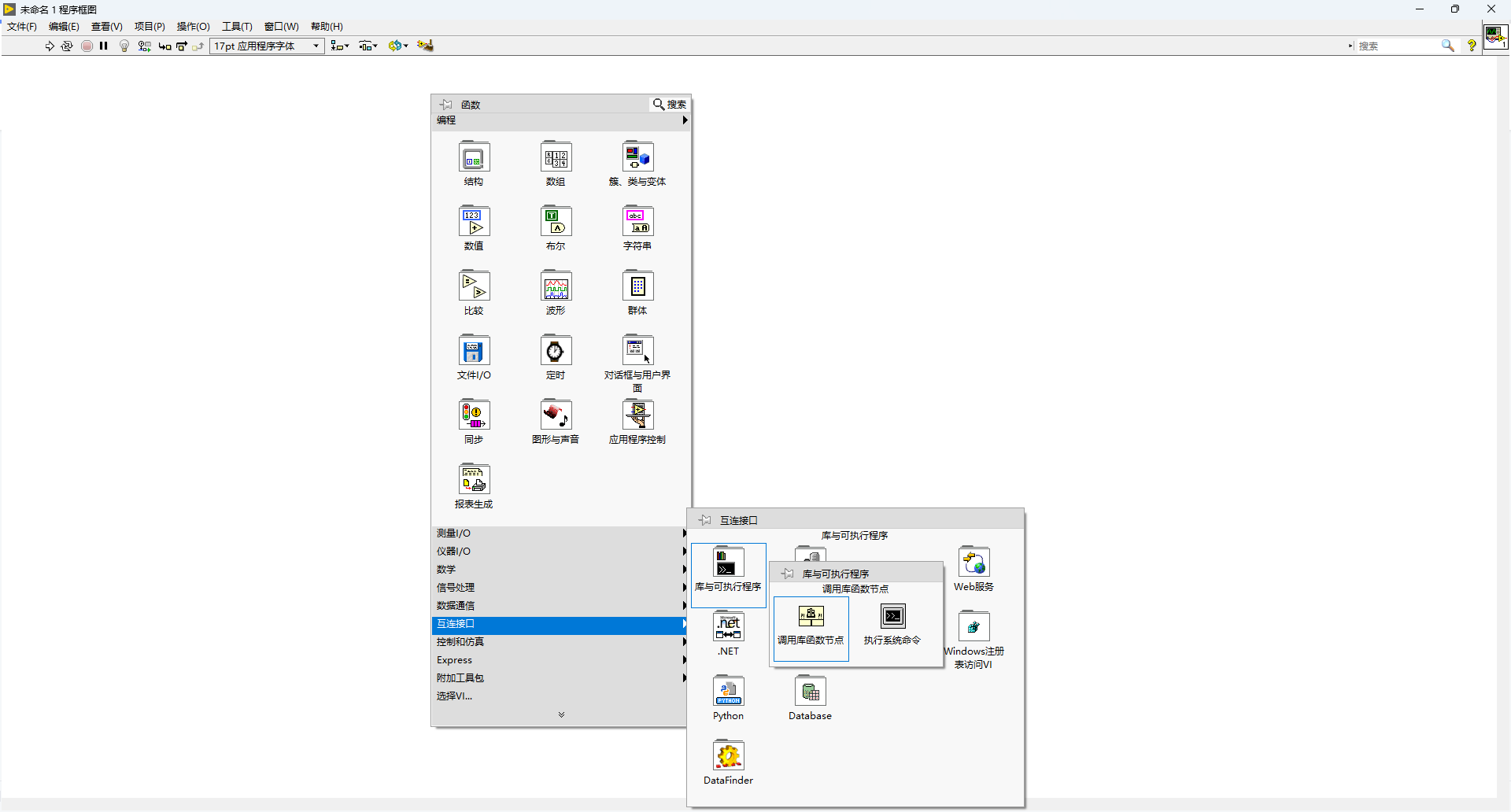
1. Go to Configuration Properties-> Linker-> Input-> Additional Dependencies, add cu**fft.lib** there if you will use cuda FFT functions, use ; to separate different libraries.
2. Click apply
3. Sometimes Release has problems finding some CUDA functions, use Debug instead. Make sure you are choosing x64 for 64bit application and x86 for 32bit application. When building DLL files, use Build-> build/rebuild solution, DO NOT use the local windows debugger as DLL can not directly execute as .exe files

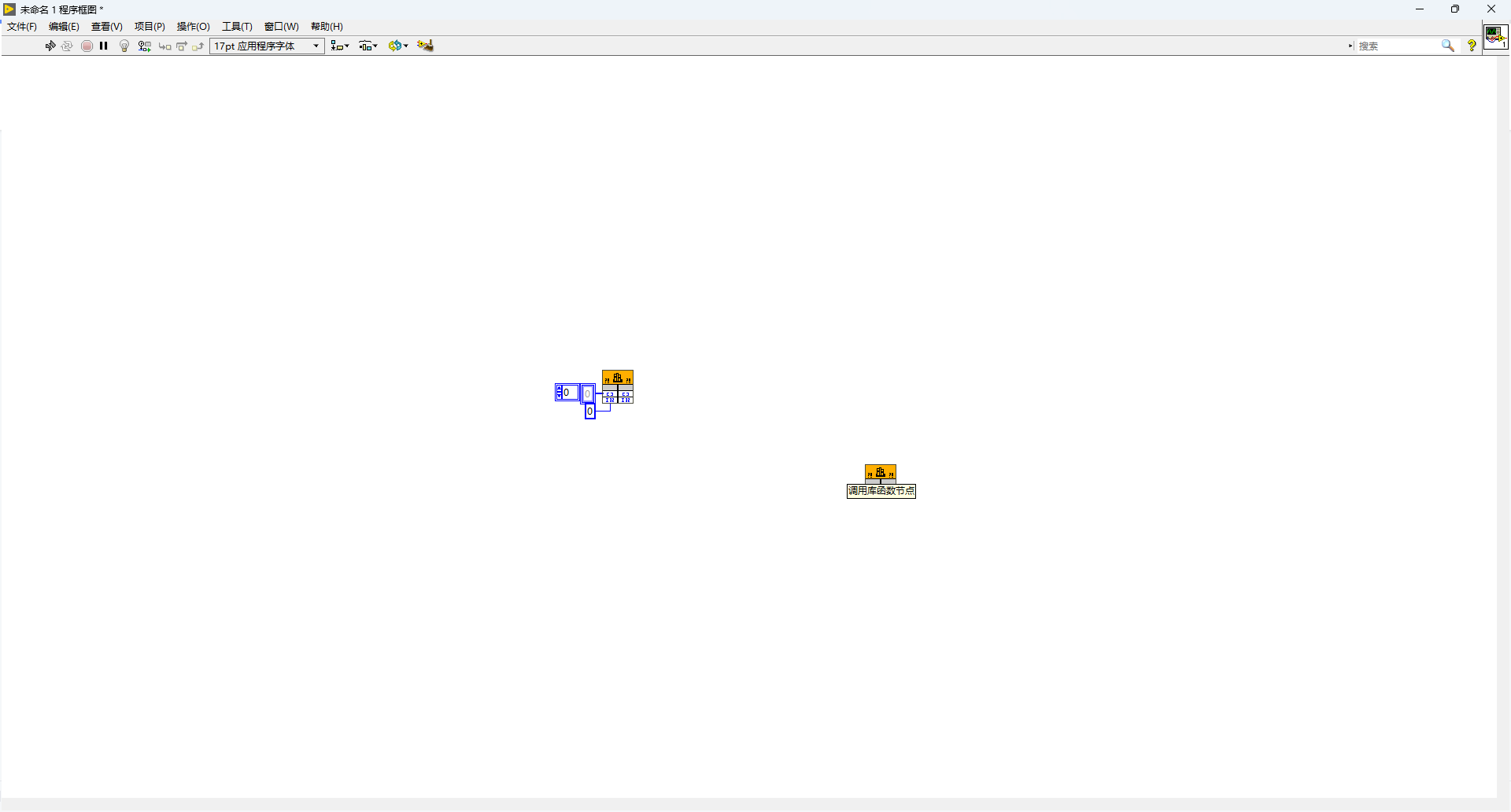


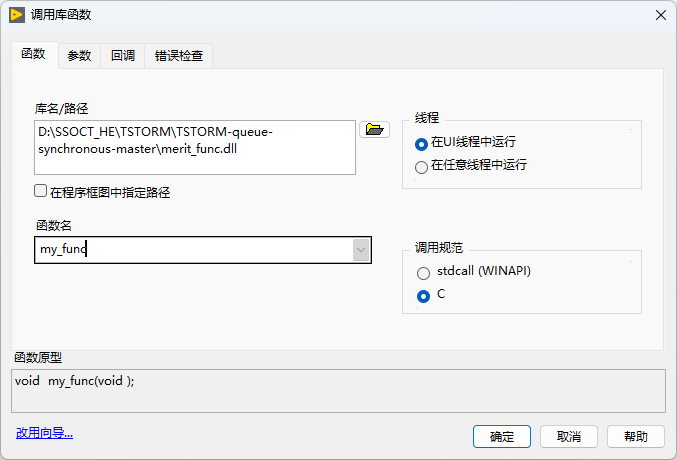
The generated DLL or .exe file can be found in <project folder>/x64/Debug. If you are building DLL file, you can use it in other programming languages.

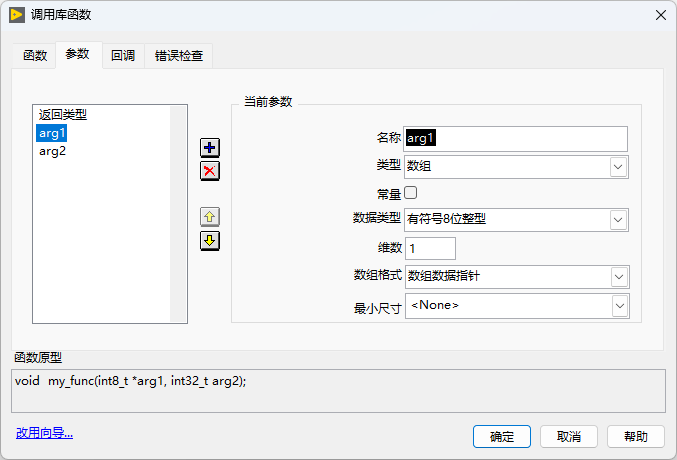
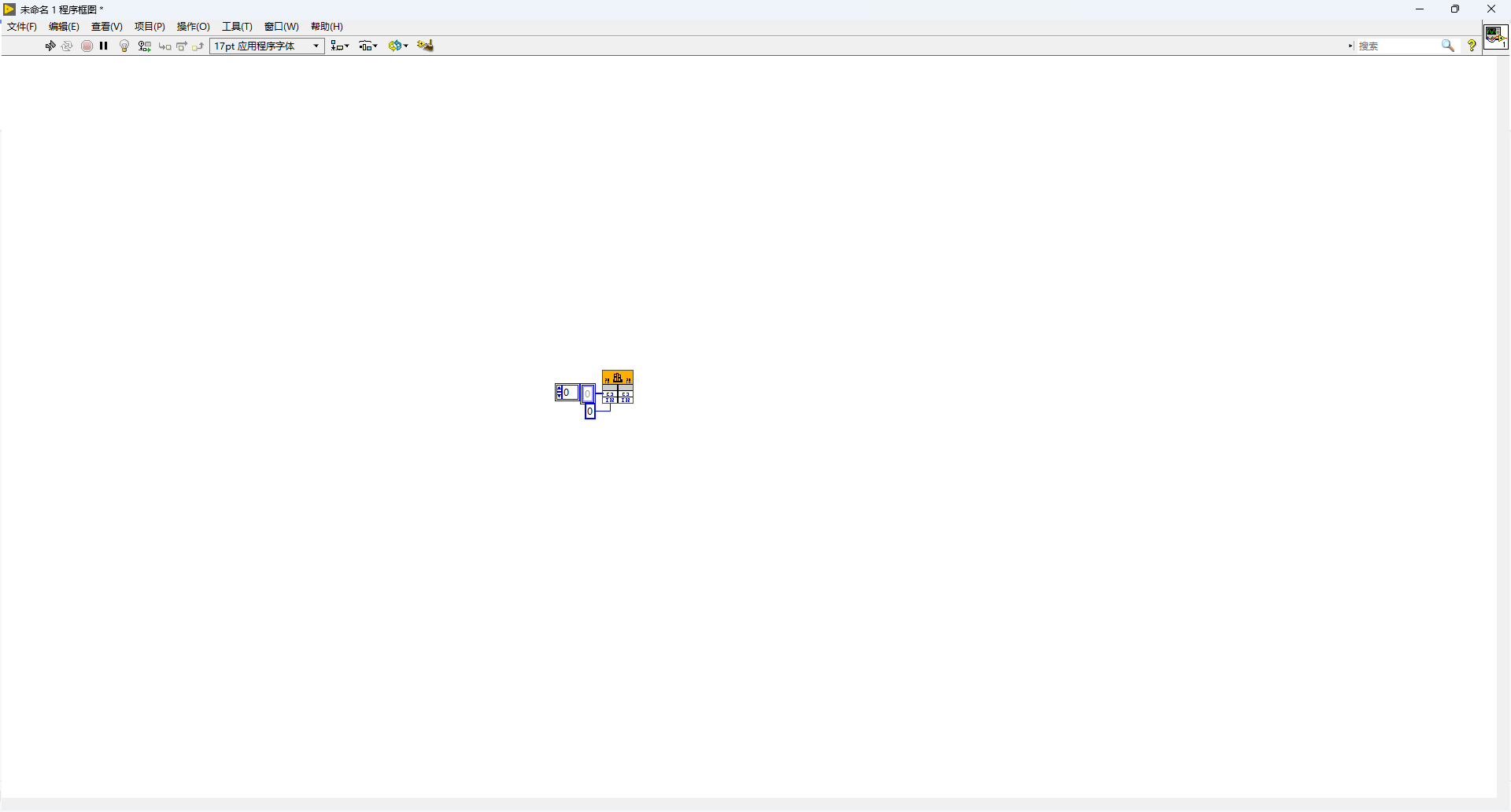
For example, you can load this DLL in LabVIEW to call some external C functions for faster execution.

Make sure your labview is 64bit version if your DLL is for 64bit system, or vice versa. I found myself unable to build 32bit DLL or .exe file in visual studio, maybe it’s because I installed 64bit CUDA toolkit. The following example of loading DLL function in Labview applies to C functions with or withnot CUDA configurations

1. Right click in the empty space of labview VI, select 互联接口-> 库于可执行程序-> 调用库函数节点
2. Double click on 库函数，select the DLL file you just generated in 库名/路径，give it a function name





1. In the parameter panel, define the return and parameters as you did in the C code , if the parameter definition in Labview is inconsistent with C code , the calling will fail 
2. After that , you can successfully call C functions in Labview 

Note: for those C functions not using CUDA, and do not go through the CUDA configuration as we did in step 2,5,6,7, you can load the DLL in Python as well using ctypes, examples can be found online. But if you do use CUDA in the C code, I found it hard to call the DLL file in Python, it throws “one of its dependence not found” error when loading DLL in python. So for those who want to use CUDA in python, I recommend using cupy module, which is an interface wrapped for doing GPU computation in python

---------Shuaibin(Stephan) Chang

01/27/2020

Updated on 01/10/2024