Setup Guide for Compiling CUDA MEX Codes

The purpose of this guide is to allow you to set up CUDA with MATLAB without having to spend days sifting through forums trying to find a path entry or link. This document will not cover good CUDA programming, but rather just how to set up CUDA, set up `nvmex`, and be able to compile CUDA programs in MATLAB.

As of MATLAB 2010b, CUDA support is built into MATLAB. This is good and bad. On the good side, it allows you to use your GPU simply within MATLAB. On the bad side, code is often faster when compiled using your own CUDA/C codes for use with MATLAB. The new MATLAB lets you do this in a roundabout way by creating GPU variables and passing them to your kernels. However, many people would rather leave MATLAB out of it and instead use a pure C-code with CUDA compiled in a MATLAB compatible MEX file. This ensures better portability as well. This guide is a "how-to" to do this. The setup info here has been tested on 32-bit Windows 7/Vista for MATLAB 2008b, and 64-bit Windows 7 with MATLAB 2010a.

Most of the information in the guide is taken from NVIDIA's manuals and various forums.

Website for this guide with all setup files `http://www.nlsemagic.com/files/cudamatlab.zip`

1) Buy your CUDA card

The first step to setting up CUDA is to buy and install a CUDA-enabled GPU card. If you are interested in just developing codes and not running huge computations, I suggest you search Amazon for the lower-end cards that support CUDA, in which case you can find cards for $50. If you want to run serious codes, then you will want to spend more money (the more money, the more speedup). For scientific computing, you also definitely want to get a card with Fermi architecture (400 series and above).

For a list of CUDA-enabled cards go to: `http://developer.nvidia.com/cuda-gpus`

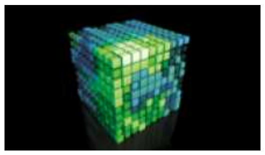
Caution! Most CUDA cards require a PCI-Express slot and a large power supply on your PC. If you have an older PC, you may only be able to use the older cards unless you buy a new power supply.

2) Download and Install Most Recent Drivers

Once your card is installed, windows will automatically install a driver for it.

Even so, it is best to go to `http://www.nvidia.com/drivers` to get the newest drivers.

3) Download and Install CUDA Toolkit



Now you need to install the CUDA Toolkit by going to `http://developer.nvidia.com/cuda-downloads` and downloading the newest version. (As of this writing, the newest toolkit is version 4.1).

Download the toolkit and any other items you want (but not the developers drivers).

There should not be much of a problem here, as these are self-extracting/installing exe files.

4) Get MATLAB

Most likely you already have MATLAB, but if not, go get it and install it.

`http://www.mathworks.com/products/matlab/`

If you are a student, the student edition is all you need and it costs $99.

I would suggest buying it from your University bookstore because it makes the student-authentication process during installation easier.

5) Get and Install Visual C++ Express 2008/2010

Microsoft® Visual Studio® 2010 Express

Microsoft® Visual Studio® 2008 Express

In order for the `nvmex` compiler to work with MATLABs MEX compiler, it needs to use the C compiler that comes with Visual C++ (others have been mentioned to work, but I have not tested them). Microsoft provides a free version of all the Visual Studios called Express. The newest full version of Visual C++ Express is 2010, but older versions of MATLAB MEX do not support it. Therefore the best plan is to get the older 2008 version of Visual C++ Express.

To get Visual C++ Express 2010, go to:

`http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express`

To get Visual C++ Express 2008, go to:

`http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express`

Simply run the downloaded exe file to install it and then restart your computer.

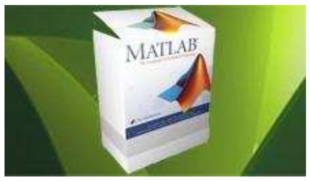
6) Get and Install Microsoft Windows SDK

Microsoft Windows SDK for Windows 7 and .NET Framework 3.5 SP1

Usually the Microsoft SDK is already installed on the system, but many times (especially for 64-bit) it is not.

To install the SDK, go to
`http://www.microsoft.com/download/en/details.aspx?id=8279`
and download the SDK and run the installer. If this link fails, simply search microsoft.com for the Windows SDK.

7) Get NVMEX Compiler Package and Install



Next you need the CUDA enabled MEX compiler called `nvmex`. This is included in the Legacy MATLAB plug-in on the NVIDIA site (see `http://developer.nvidia.com/matlab-cuda`). However, NVIDIA stopped updating the plugin as soon as MATLAB integrated GPU support in their newer versions. Therefore, the online user community has taken over and after combing through forums on multiple sites, I have collected an up-to-date `nvmex` package. The new package is geared for MATLAB 2008a and above, so if your MATLAB is older than that, you should still use the updated plugin bat files, but replace the other files with those from the NVIDIA plug-in site.

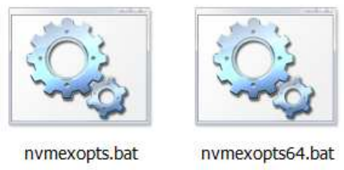
To download the most current and up-to-date nvmex package, go to `http://www.nlsemagic.com/files/cudamatlab.zip`.

To install the files, first rename the appropriate nvmex.pl file (32-bit or 64-bit) to "nvmex.pl" and put it in the MATLAB bin folder, for example:
`C:\Program Files\MATLAB\R2010a\bin\`

The other important files are **nvmex.m**, **nvmex_helper.m**, and **nvmexopts.bat**. These files must be in the directory that you want to compile CUDA MEX files. There are two **nvmexopts.bat** files provided, one for 32-bit and one for 64-bit (**nvmexopts64.bat**).

For 64-bit Windows, an additional included file is necessary: **vcvarsamd64.bat**.
This file must be placed in the directory (or equivalent):
`C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\bin\amd64\`

8) Edit nvmexopts(64).bat PATHs



This is the most annoying part of the installation. I have tried to make it as easy as possible by arranging the bat files to show which lines need to be altered as follows:

In the 32-bit bat file:

```
rem *****These two lines are all that should need to be tweaked (and target GPU below):
rem *****(Make sure you have installed the MS SDK and VS express 8)

set VSINSTALLDIR= C:\Program Files\Microsoft Visual Studio 9.0
set SDKDIR= C:\Program Files\Microsoft SDKs\Windows\v6.0A
```

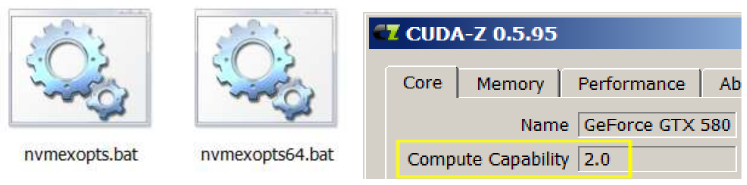
In the 64-bit bat file:

```
rem *****These two lines are all that should need to be tweaked (and target GPU below):
rem *****(Make sure you have installed the MS SDK and VS express 8)

set VSINSTALLDIR= C:\Program Files (x86)\Microsoft Visual Studio 9.0
set SDKDIR= C:\Program Files\Microsoft SDKs\Windows\v7.0
```

These directories need to be set to the correct path. As of this writing, the paths above should work.

9) Set GPU Compute Capability in nvmexopts(64).bat



Name: GeForce GTX 580
Compute Capability: 2.0

In order to use double precision with cards that support it, and to get faster codes in general, it is a good idea to compile the codes specifically for the GPU version you have installed. First, find out what compute capability your card has (either look it up online, download and run CUDA-Z, or run the CUDA SDK example “deviceQuery.exe”).

Next, add to the compile options in the .bat file as follows:

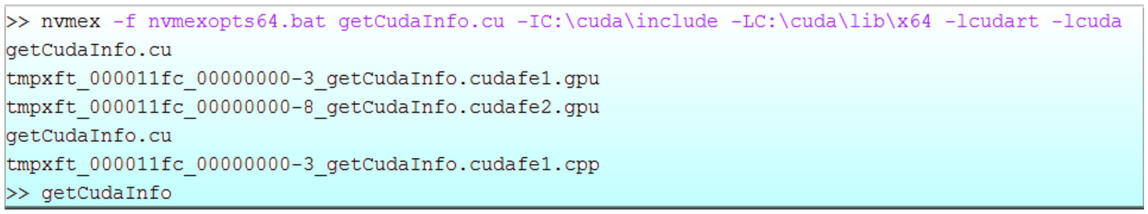
set COMPFLAGS=-gencode=arch=compute_20,core=sm_20 -gencode=arch=compute_20, code=compute_20 ***(rest of original line here)***

The **code=sm_xx** generates CUBIN files that will only work on your specific compute capability, while the **code=compute_xx** generates PTX files that are compiled at run time and are forward-compatible within the same major compute capability. The codes will run a bit faster if you include native CUBIN, but are more portable if you use PTX, so I use both. The _xx can be _12, _13, _20, etc. where the number is the compute capability (i.e. _20 is for compute capability 2.0 etc.

In order to use double precision in your code, you must have a card with compute capability

1.3 or above, and compile the codes with at least _13.

10) Compile CUDA Code from MATLAB



```
>> nvmex -f nvmexopts64.bat getCudaInfo.cu -IC:\cuda\include -LC:\cuda\lib\x64 -lcudart -lcuda
getCudaInfo.cu
tmpxft_000011fc_00000000-3_getCudaInfo.cudafe1.gpu
tmpxft_000011fc_00000000-8_getCudaInfo.cudafe2.gpu
getCudaInfo.cu
tmpxft_000011fc_00000000-3_getCudaInfo.cudafe1.cpp
>> getCudaInfo
```

Now you want to make sure everything is working.

The command in MATLAB to compile your .cu CUDA code is typically:

```
nvmex -f nvmexopts.bat yourcudacode.cu -IC:\cuda\include -LC:\cuda\lib -lcufft lcudart
lcuda
```

On 64-bit Windows, you must use

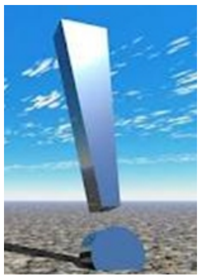
```
nvmex -f nvmexopts64.bat yourcudacode.cu -IC:\cuda\include -LC:\cuda\lib\x64 -lcufft
-lcudart -lcuda
```

The newest NVIDIA SDK does NOT install the CUDA library and include files into `C:\CUDA\` like it used to. Therefore one must use the full path in the compiler line above. On Windows, this presents a difficulty since there are often spaces in the directory names, which the compiler does not like.

The way around the problem (and what I do) is to create a folder called CUDA on the C drive (`C:\CUDA\`) and copy the folders **/lib/x64** (or **/lib/Win32**) and **/include** from the CUDA Toolkit directory (for example, **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v4.1**) into **C:\CUDA\**. If you do this, the compile lines as shown above should work fine.

Important! Don't forget to update the files in your `C:\CUDA\` directory when updating the CUDA toolkit!

That concludes the basic setup!



ONE LAST THING: When running CUDA MEX codes through MATLAB, occasionally with heavy use, crashes, NANs, programming errors, etc, MATLAB can cause a problem where the CUDA codes stop working and return garbage or crash MATLAB. This is a problem with MEX not CUDA so newer versions of MATLAB may not have this issue. If this starts happening, simply close MATLAB and restart it and everything should work fine (assuming your CUDA code has no bugs!). If that does not work, shut down the machine, unplug it, wait 30 seconds, replug it, and reboot to try again (doing this clears the RAM, which can solve a lot of problems).

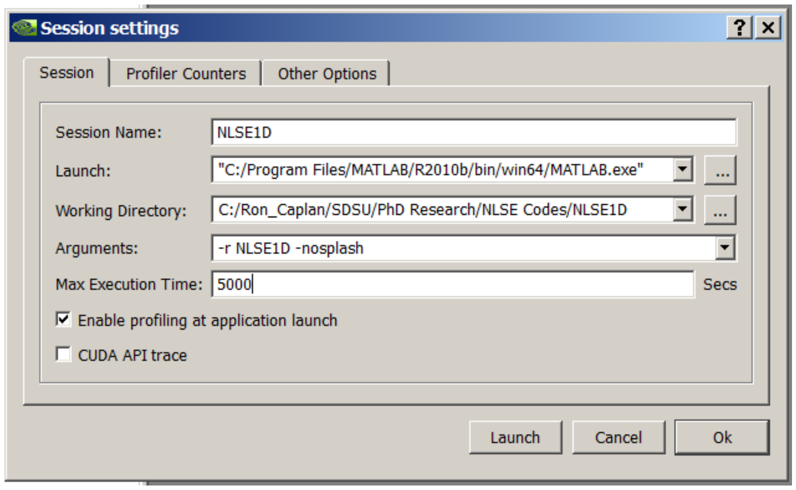
ADDITIONAL SETUP: The following are additional tips that are very useful when running CUDA codes with MATLAB on Windows:

CUDA Visual Profiler Setup

To use the profiler, you must set up the profiler to run the MATLAB executable. It is vital to NOT point to the **/MATLAB/matlab.exe**, but rather to the specific **/MATLAB/win32/matlab.exe** (or **/MATLAB/win64/matlab.exe,** or, on new versions of MATLAB, **/MATLAB/bin/win64/matlab.exe**)

Also, be sure to put an "exit;" command at the end of your MATLAB script, since the profiler will run your code multiple times and you do not want a bunch of MATLAB windows eating up RAM and desktop space.

Therefore, the setup window for visual profiler should look like this:



Session Name: NLSE1D
Launch: "C:/Program Files/MATLAB/R2010b/bin/win64/MATLAB.exe"
Working Directory: C:/Ron_Caplan/SDSU/PhD Research/NLSE Codes/NLSE1D
Arguments: -r NLSE1D -nosplash
Max Execution Time: 5000 Secs
[✓] Enable profiling at application launch
[ ] CUDA API trace

In order to profile your CUDA code, you will want to use the CUDA Visual Profiler.

A couple important things to note:
1) Notice that the working directory is where my MATLAB code is, not the folder where the MATLAB exe is located.
2) I use the -r tag to run the specific MATLAB script file I want, and the -nosplash to not have to see the MATLAB logo display each run.

3) I set the max execution time to 5000 so that there will definitely be enough time to finish the simulation.

For information on using the profiler results, see the CUDA documentation.

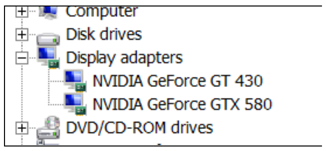
Setting up a GeForce/Quadro GPU Card as a Compute-Only Device

One drawback (besides the 25% double-precision performance and no ECC support) to using a GeForce or Quadro card for computing is that it is also typically being used for running the graphics of the OS which limits its capabilities to run code. Also, it makes the system pretty much unusable while the CUDA codes are running. An easy solution is to buy a Tesla model compute-only card, but those cost about 4 or 5 times as much as a GeForce.

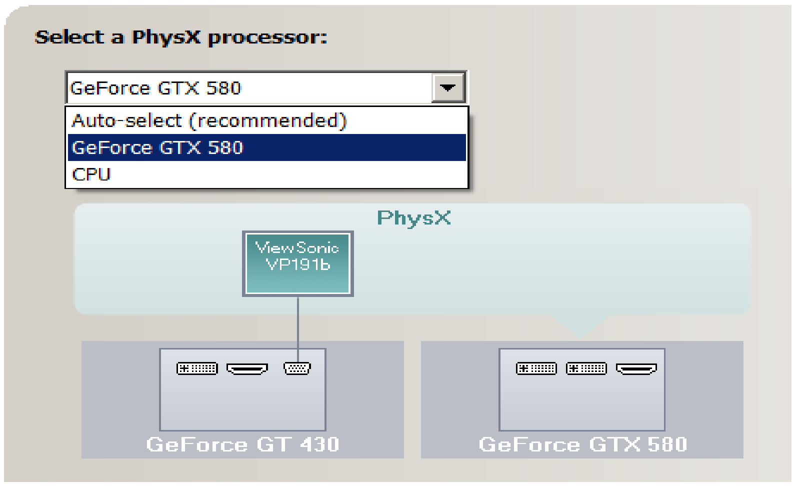
To fix this problem in a cheaper way, you can set up two GPUs on the same PC, one being used to run the display, and the other only used for CUDA computations. I have only done this on 64-bit Windows 7 so I do not know if it works on other setups.

Assuming you have a motherboard with two PCI-E slots, you can set up a GeForce card as a compute-only device as follows:

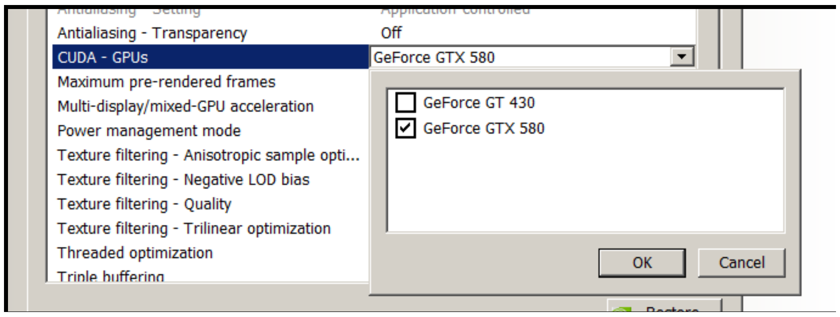
First, you will need a second (probably cheaper unless you are a gamer) NVIDIA GPU card, which should be set up on the primary PCI-E slot. Then, run the system and install the drivers for this card. Next, shut down and install the GPU you want to compute with on the second slot, and run the system and install its drivers. If all worked out, you should see two NVIDIA cards listed in the Windows device manager like this:



Once this is done, go to the PhysX setup in the NVIDIA control panel. Select the option to set the compute GPU as the only PhysX device (I do not know if this is absolutely necessary but I did it and it works so I am including this step).



Now, if the compute-only GPU is better than the primary it should automatically be set to be the CUDA device 0 so all your CUDA codes will run on it by default without having to add device selection code. To ensure this is the case, you can manually disable the CUDA capability of the GPU being used for the display by going to the 3D options in the NVIDIA control panel and un-checking the display GPU from the CUDA selection boxes as shown here:



Antialiasing - Transparency — Off
CUDA - GPUs — GeForce GTX 580

☐ GeForce GT 430
☑ GeForce GTX 580

Maximum pre-rendered frames
Multi-display/mixed-GPU acceleration
Power management mode
Texture filtering - Anisotropic sample opti...
Texture filtering - Negative LOD bias
Texture filtering - Quality
Texture filtering - Trilinear optimization
Threaded optimization
Triple buffering
OK
Cancel

Important! These settings automatically get reset when installing a new driver. Therefore, be sure to change these settings back after applying a driver update.