Creating a new project with Visual Studio on Windows

Open Visual Studio 2017 from the Windows Start menu. You may have to go through a password challenge; if so, see our document on launching Visual Studio.

Visual Studio has the concept of a solution, which contains one or more projects. Each project produces one executable (with perhaps the usual multiple versions of the executable for debug, release, profiling, etc). In fact, we typically create a new VS solution for each lab, with the solution containing just one project.

If you click on File / New / Project, Visual Studio will pop up this:

You’ll be using the GPU, so you have to create a GPU-enabled project. Click on the “NVIDIA” in the top-left list of installed project types, and then select “CUDA 10.0”. Then click on the “CUDA 10.0 Runtime” icon that shows up as the project type. Finally, choose whatever names and locations you like for your project as shown here:
Click “OK” on the bottom right, and you now have a Visual Studio project, which will be part of a solution. At that point, you can create the files you need and add them to the project.

The next thing to do is add your files to the project. Usually I create the files first, with my favorite editor, and then add them to the project. To do that, right-click on the project name (not the solution name) as shown below, and then click on the pop-up menu item “add -> existing item.” Browse to find the file you want to add. Do this for all of the .cxx and .cu files, and you’re “almost” ready to go.
Visual Studio has the concept of a debug vs. release. You choose your configuration first; this affects any subsequent builds. You should set your platform to “x64”. My example below is in “release” mode. “Debug” mode will run quite slowly on the large examples.

In order to get reasonable GPU results, you must also have various parameters set correctly. Note that Visual Studio allows you to set these parameters different for debug vs. release builds; to avoid this confusion, you should set the parameters for “All releases” (shown below). Here are the parameters to set:

- Visual Studio does not initialize the “CUDA Toolkit Custom Directory.” To set it, work through the menus: Debug / “your-project-name properties” to pop up a Properties window. Then select Cuda C++, then Common. Then set “CUDA Toolkit Custom Directory” (which will likely not be set) to “C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0”. See the picture just below for details.
- Cuda categorizes different GPUs by their Cuda Compute Capability. To set this correctly for our GPUs, again work through the menus: Debug / “your-project-name properties” to pop up a
Properties window. Then select Cuda C++, then Device. You’ll see “Code generation;” make sure that it is “compute_50,sm_50” (sometimes it defaults to “compute_30,sm_30”).

- By default, Windows only gives you a 32-bit virtual-address space, which is not enough for our matrices. To get the full 64 bits, you need two steps. First, go to “Debug / your-project-name properties” again. Then, choose Linker / System, and set “EnableLargeAddresses = Yes (LargeAddressAware).” Second, using the top-level menus Build / Configuration Manager, set your configuration to x64.

Here’s an example of the property page for CUDA Toolkit Custom Directory, correctly set.

Once you are all done, you should exit Visual Studio and then enter again (this is sometimes necessary for the parameters to fully percolate through their system!). In particular, if you get the compile error “1>CUDACOMPILE : error : couldn't get arguments: The parameter is incorrect”, this is the problem.

A final note: for some reason, the “Build / Build solution” does not always correctly determine which files have been changed since the last build. In that case, “Build / Rebuild solution,” which recompiles everything, does work fine.