Begin of investigating how to make the Compatibility to work:

YouTube Videos Related to it:

[**Get Started with the Intel® DPC++ Compatibility Tool | oneAPI | Intel Software**](https://www.youtube.com/watch?v=H8Etp9FcHbQ)

It just explains how the compatibility tool “works”, not how to make it work.

Text

Description automatically generated

Diagram

Description automatically generated with medium confidence

Besides the complex commands on how to “use” the compatibility tool, it doesn’t seem to explain step by step how to set up everything and run it.

[CUDA Portability with HIPCL and Intel® DPC++ Compatibility Tool](https://www.intel.com/content/www/us/en/developer/videos/cuda-portability-hipcl-dpcpp-compatibility-tool.html)

Not useful for installation. It explains the technical aspects of using the Compatibility tool (just glanced at it rather than watching it, I don’t think it’s worth watching)

[Intel® DPC++ Compatibility Tool Developer Guide and Reference](https://www.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top.html)

Currently, it gives similar explanations from the YouTube video on how it works. Click the Migrate Project Button at the left bottom corner.

**Deviation to make sure the Linear\_Regression code can be compiled in the terminal using nvcc:**

I had to use the [Nvidia Documentation](https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html) to get a better view to the “extension commands”

I’m currently first compiling my Linear Regression code using the nvcc compiler. After failing inserting some commands:

nvcc -o Linear\_Regression.cu

I know used the following command to create an object file:

nvcc --compile Linear\_Regression.cu

However, I didn’t know how to move forward from that, so I decided to use the following command to create .exp, .lib., and a .exe files.

nvcc --link Linear\_Regression.cu

The issue with the previous is that the executable of the program only shows the results for a fraction of a second, most likely since the program doesn’t care about retaining the results on the screen. To avoid this, now I’m using the following command to do the same as before, but now it displays in the terminal where you insert the command:

nvcc --run Linear\_Regression.cu

The issue with the previous is that the executable of the program only shows the results for a fraction of a second, most likely since the program doesn’t care about retaining the results on the screen. To avoid this, now I’m using the following command to do the same as before, but now it displays in the terminal where you insert the command:

End of Deviation

\*Intel estimates as of September 2021\* 🡪This may mean that the guide is not updated up to date.

After clicking the Before You Begin button at the right bottom, I saw a new link that I’ll follow now to see if there’s anything new I missed before.

[Get Started with the Intel® DPC++ Compatibility Tool](https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-dpcpp-compatibility-tool/top.html)

I looked at this guide before. There’s a lot of references to Linux environments, but this doesn’t necessarily mean that it is useless in windows.

You can reference custom locations by pointing to them with the

--cuda-include-path=<path/to/cuda/include>

 option in the Intel® DPC++ Compatibility Tool command line.

I don’t understand what this NOTE means: The CUDA include path should not be the same as, or a child path of, the directory where the source code that needs to be migrated is located.

Then, I followed the following instruction:

To set up the Intel® DPC++ Compatibility Tool environment, run the following:

On Windows:  Drive:\Program Files (x86)\Intel\oneAPI\setvars.bat

Text

Description automatically generatedI run it as administrator, but it just shows something like this:

After looking at it, maybe Visual Studio command-line environment could be the key to it.

Ultimately, it was redirecting myself to use **the Intel oneAPI command prompt for Intel 64 for Visual Studio 2022** that we already used before. However, I’ll try it either way.

They now ask to use the dpct --help to look at the different “options” for the compiler. I’ll create a word document that will show the command display so I can reference it later (Intel doesn’t seem to have an actual website link to document what the options are in comparison to Nvidia.) I’m gonna take some time now to edit and format the options well enough, this may take 30 to 45 minutes to complete.

After finishing the document, I’m moving to the “**Migrate as Simple Test Project**”:

The Intel® DPC++ Compatibility Tool comes with several sample projects so you can explore the tool and familiarize yourself with how it functions:

Graphical user interface, text, application, email

Description automatically generatedWe can read the README file to know more about the functionality of this.

To access the samples:

1. Use the oneapi-cli utility to select a sample from the Intel® DPC++ Compatibility Tool category or

2. Download the samples from [GitHub](https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/Migration)

I’ll attempt to use the oneapi-cli first and see how it goes. However, they recommend to go through the [**Get Started with the Intel® oneAPI Base Toolkit for Windows**](https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-oneapi-base-windows/top.html)**:**

I click in the [**Configure Your System**](https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-oneapi-base-windows/top/before-you-begin.html):

The first explains how to install CMake if we plan to use Microsoft Visual Studio (MVS). The website has a link to each version of MVS on [how to install CMake](https://learn.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=msvc-170).

Here’s also a [Troubleshooting](https://software.intel.com/en-us/get-started-with-intel-oneapi-base-windows-troubleshooting) link to OneAPI issues that I hope may have some use if I get stuck in an part.

After making sure that my environment was set up fine, I move on to the Build and Run a sample tutorial. I’ll attempt the Command Line method first:

Command Line

After some comments on how to use OneAPI Visual Studio (which I will ignore it for now t focus only in command line implementation,) I started to read the “Download Samples using the oneAPI CLI Samples Browser”

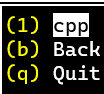
We will use the oneAPI CLI Samples Browser to browse the collection of online oneAPI samples. We can copy them to our local disk as buildable sample projects. Most oneAPI sample projects are built using Make or CMake, so the build instructions are included as part of the sample in a README file. The oneAPI CLI utility is a single-file, stand-alone executable that has no dependencies on dynamic runtime libraries

After that explanation, it provides a [list of components that support CMake](https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/oneapi-development-environment-setup/use-cmake-with-oneapi-applications.html). It also provides information on how to use this toolkit offline, see [Developing with Offline Systems in the Troubleshooting section](https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-intel-oneapi-base-windows/top/troubleshooting.html#troubleshooting_OFFLINEDEVELOPMENT). They also provide a video presentation of how to create a project with the command line: [Exploring Intel® oneAPI Samples from the Command Line](https://www.youtube.com/watch?v=de-hS_d37Hc&feature=youtu.be). However, I saw the video before and I don’t think it helped at all. Nevertheless, I’ll include a little section on what the video talks about:

“Video Instructions”

1. Run the **Intel oneAPI command prompt for Intel 64 for Visual Studio 2022** from the Start (this will automatically configure the OneAPI environment for you)

2. Launch the oneAPI CLI utility by running the following command: oneapi-cli

3. Choose to create a project. In the video, there is support for python, but mine shows the following choices:

Text

Description automatically generatedText

Description automatically generated4. After selection cpp, the video shows a different version of programs than me. May be because of the age of the tutorial, but I’ll leave a image comparison so it is easier to analyze.

Vs

Note: Apparently, a feature of oneAPI CLI utility is that it will give you a heads up if we’re missing any dependencies required to build a sample.

5. I’m choosing the Base: Vector Add Project to create

Graphical user interface, application

Description automatically generated6. The directory options in my version differ from the ones in the video. Here’s the example:

Graphical user interface, text

Description automatically generated

Vs

Note: My version could just combine both Directory and Project Name together.

Graphical user interface, application, website

Description automatically generated7. I created a Temp directory in C to match the tutorial on YouTube.

A screenshot of a computer

Description automatically generated with medium confidence

Vs

8. After entering Enter twice to create the file, we’ll get two options, we’ll select the “View Readme and Quit” one. If I want to access the README again, we can always go to the directory and open the README file with VSC or a text editor.

9. We will navigate to the project we created by typing the following: cd C:\Temp\vector-add

10. Apparently the project is built with nmake. The video shows the commands too fast, but here they are:

nmake -f Makefile.win

nmake -f Makefile.win run

11. After running the last command, we got the following error:

Vector size: 10000

**InvalidBuiltinSetName: Expects OpenCL.std. Actual is OpenCL.DebugInfo.100 [Src: ..\..\..\libSPIRV\SPIRVModule.cpp:594 SPIRVBuiltinSetNameMap::rfind(BuiltinSetName, &BuiltinSet) ]**

**NMAKE : fatal error U1077: '.\vector-add-buffers.exe' : return code '0xc0000005' Stop.**

WE WILL NEED TO INVESTIGATE FURTHER AFTER I FINISH MY ATTEMPTS AT USING THE COMPATIBILITY TOOL.

“End of Video Part”

After finishing the video, I move on to follow the instructions in the website:

1. Create a folder like C:\samples\vector-add

2. We will use PowerShell to plug in the following command (it doesn’t work if attempted in cmd):

cmd.exe "/K" '"C:\Program Files (x86)\Intel\oneAPI\setvars.bat" && powershell'

NOTE: **The setvars.bat script can be managed using a configuration file, which is especially helpful if you need to initialize specific versions of libraries or the compiler, rather than defaulting to the "latest" version. For more details, see** [**Using a Configuration File to Manage Setvars.bat**](https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/oneapi-development-environment-setup/use-the-setvars-script-with-windows/use-a-config-file-for-setvars-bat-on-windows.html)**. See** [**oneAPI Development Environment Setup**](https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/oneapi-development-environment-setup.html) **for more configuration options.**

3. Run the following command: oneapi-cli.exe

4. Use the up and down arrow keys to select. Create a project, then press Enter

5. Move the arrow key down to select Create a project, then press Enter. The language selection will appear. **If you want to run samples from a toolkit other than the Intel® oneAPI Base Toolkit, install the domain-specific toolkit before proceeding.**

6. Select the language for your sample. For your first project, select cpp, then press Enter. (there is no option for python in my options)

7. Select the Vector Add sample. Vector Add is a simple test application that will help verify that the tools are setup correctly and can access your system's GPU.

8. After you select a sample, press Enter.

Graphical user interface, text, application

Description automatically generated9. Specify the location for the project. The default location includes the path from where the utility was run and the name of the project (this time it was exactly like my display)

10. Navigate to the project we created by typing the following: cd C:\samples\vector-add

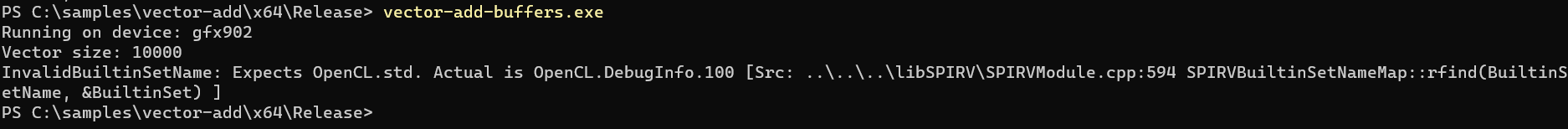
11. Build the program with the following command:

MSBuild vector-add.sln /t:Rebuild /p:Configuration="Release"

In my surprise, it worked. I’m going to attach the command output in a word document

12. Navigate to the Release folder using: cd x64/Release

13. Run the program with the following command: vector-add-buffers.exe

After running the program, I got the following error:

InvalidBuiltinSetName: Expects OpenCL.std. Actual is OpenCL.DebugInfo.100 [Src: ..\..\..\libSPIRV\SPIRVModule.cpp:594 SPIRVBuiltinSetNameMap::rfind(BuiltinSetName, &BuiltinSet) ]

14. The how to fix the error will be explored in the future

After finishing the guide with the command prompt, I decided to move on to the Visual Studio Implementation.

Visual Studio Implementations

You can use the Intel® oneAPI DPC++/C++ Compiler within the Microsoft Visual Studio\* integrated development environment (IDE) to develop C++ applications, including static library (.LIB), dynamic link library (.DLL), and main executable (.EXE) applications. This environment makes it easy to create, debug, and execute programs. You can build your source code into several types of programs and libraries using the IDE or from the command line.

The IDE offers these major advantages:

* Makes application development quicker and easier by providing a visual development environment.
* Provides integration with the native Microsoft Visual Studio\* debugger.
* Makes other IDE tools available.

Create a Project Using Microsoft Visual Studio

I’ll follow the website instead of the YouTube video since the last we look at the YouTube video, it was outdated.

1. Open Microsoft Visual Studio

2. Open Visual Studio 2022 and click Continue without code.

Graphical user interface, text, application, email

Description automatically generated3. From the top menu, select:

Visual Studio 2019 and 2022: Extensions > Intel > Browse an Intel oneAPI Sample

Visual Studio 2017: Intel > Browse an Intel oneAPI Sample

The samples screen will open:

4. In the center area, select Vector Add. Vector Add is a simple test application that will help verify that the tools are setup correctly and can access your system's GPU.

Graphical user interface, application

Description automatically generatedMine is a little different than their example:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generated5. Click Ok and then find the Solution Explorer to right-click on vector-add and select Rebuild.

6. After the program is built, click DebugStart >> Without Debugging

Their results:

Text

Description automatically generated

Text

Description automatically generatedMy Results:

This is probably because I use a different program since there was no “Vector-add” example. We can see it recognizes the integrated graphics that my CPU has. After reading the README, I confirmed what the code does. Here’s the explanation contained in the README:

The `vector-add` is a simple program that adds two large vectors of integers

and verifies the results. This program is implemented using C++ and Data

Parallel C++ (DPC++) for Intel&reg; CPU and accelerators.

In this sample, you can learn how to use the most basic code in C++ language that offloads computations to a GPU using the DPC++ language. This includes using Unified Shared Memory (USM) and buffers. USM requires an explicit wait for the asynchronous kernel's computation to complete. Buffers, at the time they go out of scope, bring main memory in sync with device memory implicitly; the explicit wait on the event is not required as a result. This sample provides examples of both implementations for simple side-by-side reviews (the Windows sample only supports USM).

The code will attempt to execute on an available GPU and fallback to the system's CPU if a compatible GPU is not detected. If successful, the name of the offload device and a success message is displayed. And your development environment is set up correctly!

Create a Project Using Microsoft Visual Studio Code

[Interesting Link 1](https://www.intel.com/content/www/us/en/develop/documentation/using-vs-code-with-intel-oneapi/top/local-host-top/local-host.html) 🡪 Using this link I was able to attempt and run the simple-add code following the instructions in the website and the instructions in the README:

**### On a Windows\* System Using a Command Line Interface**

1. Select **\*\*Programs\*\*** > **\*\*Intel oneAPI 2021\*\*** > **\*\*Intel oneAPI Command Prompt\*\*** to launch a command window.

2. Build the program using the following `nmake` commands (Windows supports USM only):

    ```

    nmake -f Makefile.win

    ```

3. Run the program using:

    ```

    nmake -f Makefile.win run

    ```

4. Clean the program using:

    ```

    nmake -f Makefile.win clean

However, it failed:

Running on device: gfx902

Array size: 10000

Shared memory allocation failure.

NMAKE : fatal error U1077: '.\simple-add-usm.exe' : return code '0xffffffff' Stop.Text

Description automatically generated

After finishing the previous attempts at creating a project, I’ll move on with the tutorial on the **Try a Sample Project**. I’m cloning the repository from the tutorial ([link](https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/Migration))

After reading at the documentation, I decided to ignore it since my previous attempt at it always gave the same result. Instead, I’ll use the built-in tool in Visual Studio 2022.

Instructions on How to Use the Visual Studio 2022 built-in Compatibility Tool

*Coming in the near future*

There is an issue, however. The new attempts are better than the previous thanks that the tool runs “better” commands that I can think of. Moreover, it adds some dpct options I was unaware of. Long story short, I added this command in the additional options block.

Using : --cuda-include-path="C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.7\include"

This gave out a better error than the previous attempts I had. The error apparently now is that the CUDA Libraries I included are not supported by the DPC++ Compatibility Tool. However, I went to the [Intel® DPC++ Compatibility Tool Release Notes](https://www.intel.com/content/www/us/en/developer/articles/release-notes/release-notes-for-intel-dpcpp-compatibility-tool.html) and the 2022.2.0 now supports CUDA version 11.7 (the version I have installed,) so I’m updating my OneAPI ToolKit and hope that now it will be compatible.

After updating the OneAPI ToolKit Tools, I was successfully able to use the DPC++ Tool to translate the Linear Regression code.

However, after inspecting the new translated file, I notice that Allocation of the memory for device size pointers was not correctly translated. After analyzing both codes, I was able to correctly translate the code to their respective CUDA counterparts (details in “Notes on how to fix the first translated code issue.txt”.) However, after running the code, it gave out a wrong result.

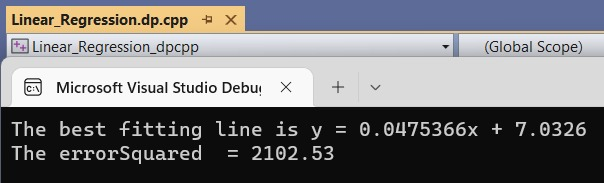
After looking at the 01\_SYCL\_Migration\_Simple\_VectorAdd.ipynb, I decided to modify the original CUDA code to fit the example and make the life easier for the Compatibility tool.

Changes:

* Remove the concatenation to **double pointer floats** of the device pointers. [Remove (float\*\*) from cudaMalloc
* Change the block and grid configuration to a **one-dimensional grid** with **S elements (indexes).** This applies for both kernel activation (in Main) and kernel configurations (in the kernel function declaration).

When I translated it again, it gave out the same untranslated part, but after using the same fix as before. The code worked successfully (giving out the same output as the CUDA code.)

Graphical user interface, text, application

Description automatically generatedFINAL CUDA CODE OUTPUT FINAL DPCPP CODE OUTPUT