

Get Started with the Intel® oneAPI HPC Toolkit for Windows*

Contents

Chapter 1: Get Started with the Intel® oneAPI HPC Toolkit for Windows*

Configure Visual Studio*	4
Build and Run a Sample Project Using the Visual Studio* Command Line	5
Next Steps	9
Build and Run a Sample Project Using Visual Studio Code	9
Next Steps	22
Run a Sample Project with Visual Studio*	23
Next Steps	24
Using Containers	25
Using Cloud CI Systems	25
Next Steps	25
Using Intel Compilers and Libraries for the Best Experience	26
Troubleshooting	27
Notices and Disclaimers	29

Get Started with the Intel® oneAPI HPC Toolkit for Windows*

1

The Intel® oneAPI HPC Toolkit is a comprehensive suite of development tools that make it fast and easy to build modern code that gets maximum performance out of the newest Intel® processors. This toolkit enables high performance computing on clusters or individual nodes with flexible options including optimal performance on a CPU or GPU.

The Intel® oneAPI HPC toolkit helps streamline coding with advanced techniques in vectorization, multi-threading, multi-node, and memory optimization. Get powerful, consistent programming with support for 512-bit Intel® Advanced Vector Extensions (Intel® AVX-512) for Intel® Core™ and Intel® Xeon® processors, and modern development standards. The next-generation Intel® MPI Library offers improved scalability and reduced latency.

Before you begin, make sure that the Intel® oneAPI HPC Toolkit is installed. The Intel® oneAPI Base Toolkit is required to run the samples in the Intel® HPC Toolkit. Visit the [Intel oneAPI Toolkits page](#) for installation instructions.

Follow these steps to run a sample project:

1. [Configure Your System](#). Depending on your development environment and hardware, some additional configurations may be required.
2. Build and run a sample project using one of these methods:
 - [Command Line](#)
 - [Visual Studio*](#)
 - [Visual Studio Code*](#)

After running a sample, learn more about the Intel® oneAPI HPC toolkit in the [Next Steps](#) section and explore the tools included in the toolkit:

- [Intel® oneAPI DPC++/C++ Compiler](#)
- [Intel® C++ Compiler](#)
- [Intel® Fortran Compiler](#)
- [Intel® MPI Library](#)
- [Intel® Inspector](#)
- [Intel® Trace Analyzer and Collector](#)

You can also download an offline version of this guide from the [Downloadable Documentation](#) page.

Subset Bundles

If you need a smaller, targeted set of tools, consider using these subset bundles designed for specific development use cases:

Intel® C++ Essentials

Compile, debug, and use the most popular performance libraries from the Intel oneAPI Base Toolkit for C++ and SYCL on Intel CPUs and GPUs.

- Intel® oneAPI DPC++/C++ Compiler
- Intel® Distribution for GDB*
- Intel® DPC++ Compatibility Tool
- Intel® oneAPI DPC++ Library (oneDPL)

- Intel® oneAPI Threading Building Blocks (oneTBB)
- Intel® oneAPI Math Kernel Library (oneMKL)

Download [Intel® C++ Essentials](#).

Intel® Fortran Essentials

Compile, debug, and use the most popular performance libraries from the Intel oneAPI HPC Toolkit for Fortran numeric workloads on Intel CPUs and GPUs.

- Intel® Fortran Compiler
- Intel® Distribution for GDB*
- Intel® oneAPI Math Kernel Library (oneMKL)
- Intel® MPI Library

Download [Intel® Fortran Essentials](#).

Additional Resources

- [Intel® oneAPI Programming Guide](#)
- [Explore SYCL* with Samples from Intel](#)
- [Intel® oneAPI HPC Toolkit Overview](#)
- [Intel® oneAPI Toolkits Installation Guide](#)

Configure Visual Studio*

The Intel® oneAPI Base Toolkit (Base Kit) is required if you want to run the samples in the Intel® HPC Toolkit. If you have not installed the Base Kit please return to the [Installation Guide](#) and install the Toolkit.

To set up your system, you need to:

- [Install CMake*](#) to build many of the samples
- [For GPU users, install GPU drivers](#)

Install CMake* to Build Samples (optional)

This optional section only applies if you plan to use Microsoft Visual Studio for cross-platform development. Many of the samples require CMake in such a development environment.

CMake is not required to use the oneAPI tools and toolkits. Most of the oneAPI samples that can be built and run on Microsoft Windows* include a Visual Studio project file to manage the build process.

To build those samples on Windows that require CMake and do not include a Visual Studio project file, you may need to add some additional Visual Studio *workloads* as part of your Visual Studio installation.

The Visual Studio CMake tools for Windows are part of the Desktop development with C++ workload and the Linux Development with C++ workload. Both of these Visual Studio workloads are required for cross-platform CMake development on Windows. To install these workloads, see the following links:

- [Visual Studio 2019](#)
- [Visual Studio 2022](#)

If you installed an Intel® oneAPI toolkit before installing Visual Studio, the oneAPI plug-ins for Visual Studio may be absent. In this case, install Visual Studio and then refer to the [Troubleshooting](#) section of this documentation for instructions on how to fix or add the missing oneAPI plug-ins for Visual Studio.

For more information about CMake, refer to [CMake.org](#).

For Microsoft Windows*, the Visual Studio* C++ Development tools must be installed, including the CMake tools in the Desktop development with C++ workload. Installing the Linux Development with C++ workload will not provide the necessary tools to run all samples. Use the links below to install Visual Studio and CMake.

If you already installed an Intel® oneAPI toolkit before installing Visual Studio, the oneAPI plug-ins for Visual Studio will be absent. If this happens, install Visual Studio and CMake using the links below. Then refer to the [Troubleshooting](#) section for instructions on how to fix the plug-ins.

- [Visual Studio 2019](#)
- [Visual Studio 2022](#)
- [CMake.org](#)

Visual C++ Tools for CMake is installed by default as part of the *Desktop Development with C++ workload*.

For more information about CMake, refer to [CMake.org](#).

GPU Drivers or Plug-ins (Optional)

You can develop oneAPI applications using C++ and SYCL* that run on Intel, AMD*, or NVIDIA* GPUs.

To develop and run applications for specific GPUs, you must first install the corresponding drivers or plug-ins:

- To use an Intel GPU, [install the latest Intel GPU drivers](#).
- To use an AMD GPU (Linux only):
 - [Read the oneAPI for AMD GPUs Guide](#) from Codeplay.
 - [Download oneAPI for AMD GPUs](#).
- To use an NVIDIA GPU (Linux and Windows):
 - [Read the oneAPI for NVIDIA® GPUs Guide](#) from Codeplay.
 - [Download oneAPI for NVIDIA® GPUs](#).

Run a Sample Project

Run a sample project using one of these methods:

- [Command Line](#)
- [Visual Studio*](#)

Build and Run a Sample Project Using the Visual Studio* Command Line

NOTE If you have not already configured your development environment, go to [Configure Your System](#) then return to this page. If you have already completed the steps to configure your system, continue with the steps below.

Command line development can be done with a terminal window or done through Visual Studio Code*. Some tasks can be automated using extensions. To learn more, see [Using Visual Studio Code with Intel® oneAPI Toolkits](#).

To compile and run a sample:

1. Download the sample using the oneAPI CLI Samples Browser.
2. Compile and run the sample with Microsoft Build*

Download Samples using the oneAPI CLI Samples Browser

Use the oneAPI CLI Samples Browser to browse the collection of online oneAPI samples. As you browse the oneAPI samples, you can copy them to your 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.

An internet connection is required to download the samples for oneAPI toolkits. For information on how to use this toolkit offline, see [Developing with Offline Systems](#) in the [Troubleshooting](#) section.

If you develop using Visual Studio Code*, some tasks can be automated with extensions. To learn more, see [Using Visual Studio Code with Intel® oneAPI Toolkits](#).

To watch a video presentation of how to create a project with the command line, see [Exploring Intel® oneAPI Samples from the Command Line](#).

NOTE The oneAPI CLI Samples Browser does not work with system proxy settings and does not support WPAD proxy. If you have trouble connecting from behind a proxy, please see [Troubleshooting](#).

1. Open a command window.
2. Set system variables by running setvars:

Component Directory Layout:

```
"C:\Program Files (x86)\Intel\oneAPI\setvars.bat"
```

NOTE For Windows PowerShell* users, execute this command:

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

Unified Directory Layout:

```
"C:\Program Files (x86)\Intel\oneAPI<toolkit-version>\oneapi-vars.bat"
```

NOTE For Windows PowerShell* users, execute this command:

```
cmd.exe "/K" '"C:\Program Files (x86)\Intel\oneAPI<toolkit-version>\oneapi-vars.bat" && powershell'
```

The command above assumes you installed to the default folder. If you customized the installation folder, `setvars` | `oneapi-vars` script is in your custom folder.

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](#). See [oneAPI Development Environment Setup](#) for more configuration options.

3. In the same command window, run the application :

```
oneapi-cli
```

The oneAPI CLI menu appears:

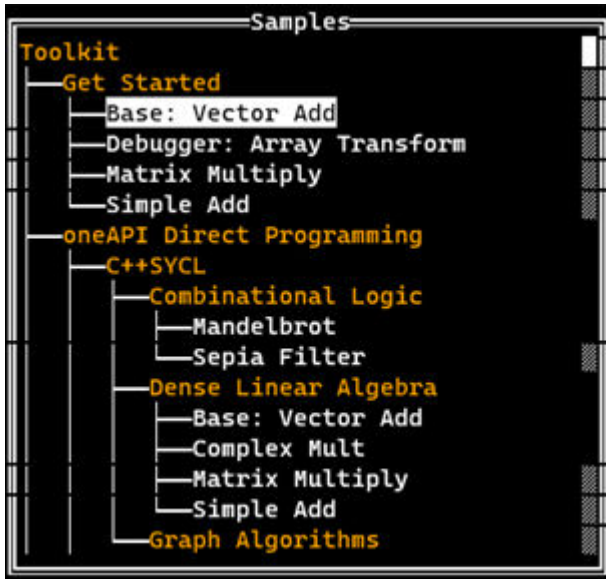
```
(1) Create a project
(2) View oneAPI docs in browser
(q) Quit
```

4. Move the arrow key down to select **Create a project**, then press Enter. The language selection will appear. If you installed Intel® oneAPI Base Toolkit, but you want to work with the Intel® oneAPI HPC Toolkit and samples, ensure the HPC toolkit is installed. If it is not installed, [install the HPC toolkit](#), then return to step 1 of this procedure.

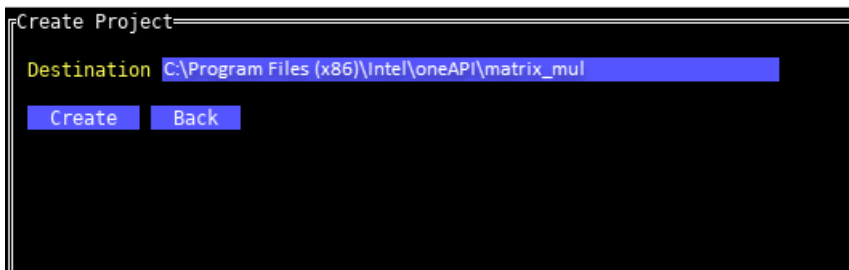
```
-----Select sample language-----
(1) cpp
(2) python
(b) Back
(q) Quit
```

Install a domain-specific toolkit
for more code samples support

5. Select the language for your sample. For your first project, select **cpp**, then press Enter. The toolkit samples list appears.
6. Select the **Matrix Mul** sample which appears at the top of the list:



7. After you select a sample, press Enter.
8. Specify the location for the project. The default location includes the path from where the utility was run and the name of the project.
9. Press Tab to select Create, then press Enter:



Now that you have the samples downloaded, compile and run the sample with [Microsoft Build*](#)

Build and Run a Sample Using Microsoft Build*

1. Using the same command prompt window where you ran `setvars.bat`, navigate to the folder where you downloaded the samples.
2. Build the program:

```
MSBuild matrix_mul.sln /t:Rebuild /p:Configuration="Release"
```

3. Navigate to the Release folder (example: x64/Release)
4. Run the program:

```
matrix_mul_dpccpp.exe
```

NOTE Some samples require additional steps or arguments for building and/or running the sample. Review the sample's `README.md` file for specific details regarding how to build and run the sample.

A success message will appear.

See [Explore SYCL* Through Samples](#) to learn more.

Next Steps

Intel® oneAPI HPC Toolkit

After successfully building a sample application, [Explore SYCL* Through Samples](#).

Maximize resources with [OpenMP Offloading](#).

Diagnose issues in the [Troubleshooting](#) section.

Learn more about the components in this toolkit:

Tool	Description
Intel® oneAPI DPC++/C++ Compiler	Use this standards-based C++ compiler with support for OpenMP* to take advantage of more cores and built-in technologies in platforms based on Intel® Xeon® Scalable processors and Intel® Core™ processors. Learn more .
Intel® C++ Compiler	Create code that takes advantage of more cores and built-in technologies in platforms based on Intel® processors. Learn more .
Intel® Fortran Compiler Classic	Generate optimized, scalable code for Intel® Xeon® Scalable processors and Intel® Core™ processors with this standards-based Fortran compiler with support for OpenMP*. Get Started .
Intel® MPI Library	Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. Use the library to develop applications that can run on multiple cluster interconnects. Get Started .
Intel® Inspector	Locate and debug threading, memory, and persistent memory errors early in the design cycle to avoid costly errors later. Learn more .
Intel® Trace Analyzer and Collector	Understand MPI application behavior across its full runtime. Learn more .
oneAPI GPU Optimization Guide	The oneAPI GPU Optimization Guide demonstrates how to improve the behavior of your software by partitioning it across the host and accelerator to specialize portions of the computation that run best on the accelerator. Specialization includes restructuring and tuning the code to create the best mapping of the application to the hardware. The value of oneAPI is that it allows each of these variations to be expressed in a common language with device-specific variants launched on the appropriate accelerator.

For more information about this toolkit, see the [Intel® oneAPI Toolkits](#) page.

Build and Run a Sample Project Using Visual Studio Code

Intel® oneAPI toolkits integrate with third-party IDEs to provide a seamless GUI experience for software development.

NOTE If you are using Visual Studio Code (VS Code) with FPGA, see the [FPGA Workflows on Third-Party IDEs for Intel® oneAPI Toolkits](#).

NOTE

An internet connection is required to download the samples for oneAPI toolkits. If you are using an offline system, download the samples from a system that is internet connected and transfer the sample files to your offline system. If you are using an IDE for development, you will not be able to use the oneAPI CLI Samples Browser while you are offline. Instead, download the samples and extract them to a directory. Then open the sample with your IDE. The samples can be downloaded from here: [Intel® oneAPI Toolkit Code Samples](#)

This procedure requires the Sample Browser extension to be installed. The next section will describe how to install it. If you have already installed it, skip to [Create a Project Using Visual Studio Code](#).

Extensions for Visual Studio Code Users

To watch a video presentation of how to install extensions and use them to set up your environment, explore sample code, and connect to the Intel® Developer Cloud using Visual Studio Code, see [oneAPI Visual Studio Code Extensions](#).

You can use VS Code extensions to set your environment, create launch configurations, and browse and download samples:

1. From Visual Studio Code, click on the Extensions logo in the left navigation.



2. Locate the extension titled **Sample Browser for Intel oneAPI Toolkits**, or visit <https://marketplace.visualstudio.com/publishers/intel-corporation> to browse available extensions.
3. Click **Install**.
4. Next, locate the extension titled **Environment Configurator for Intel oneAPI Toolkits**.
5. Click **Install**.



For more information about VS Code extensions for Intel oneAPI Toolkits, see [Using Visual Studio Code* to Develop Intel® oneAPI Applications](#).

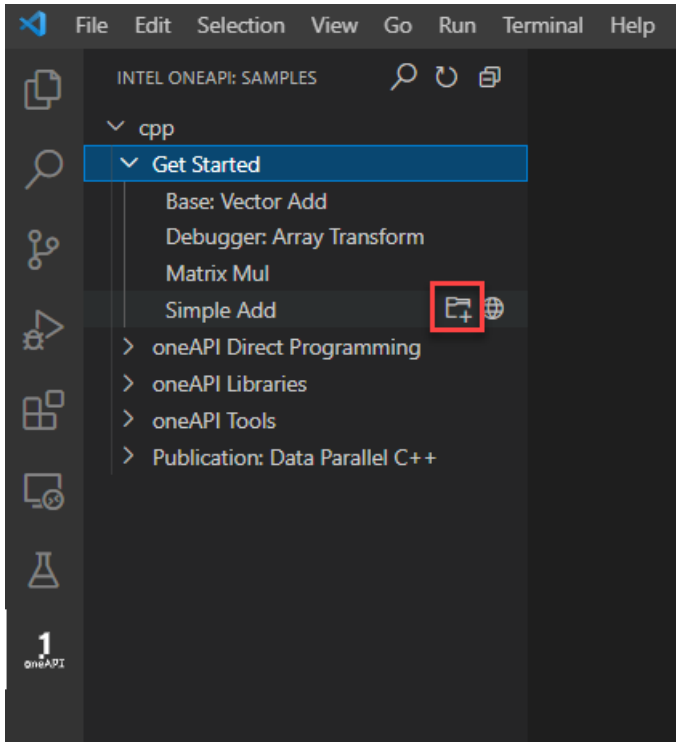
Create a Project Using Visual Studio Code

1. Click on the oneAPI button on the left navigation to view samples.

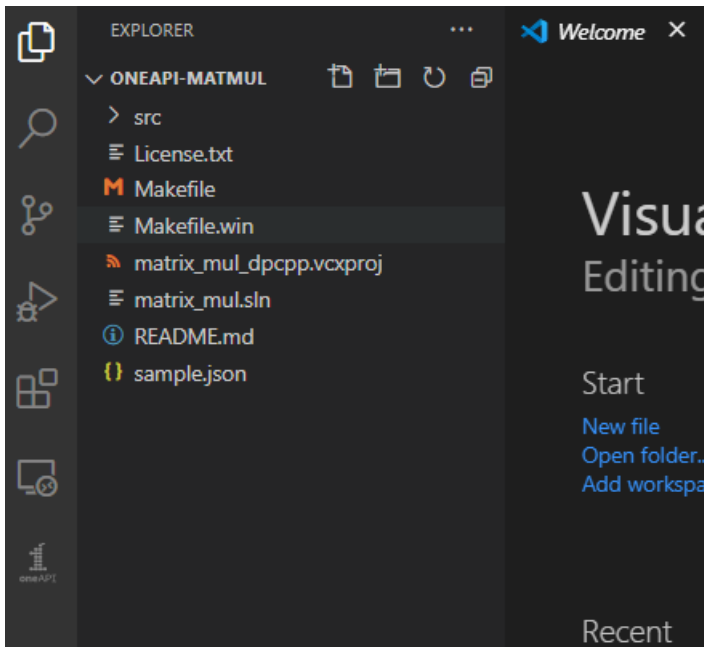
To watch a video presentation of how to install extensions and use them to set up your environment, explore sample code, and connect to the Intel® Developer Cloud using Visual Studio Code, see [oneAPI Visual Studio Code Extensions](#).



2. A list of available samples will open in the left navigation.
3. To view the readme for the sample, click the  next to the sample. If you choose to build and run the sample, the readme will also be downloaded with the sample.
4. Find the sample you want to build and run. Click the  to the right of the sample name.

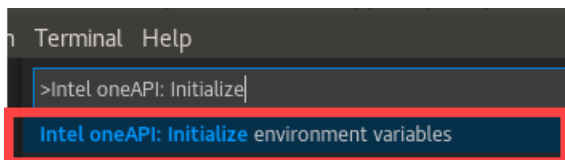


5. Create a new folder for the sample. The sample will load in a new window:



Set the oneAPI Environment

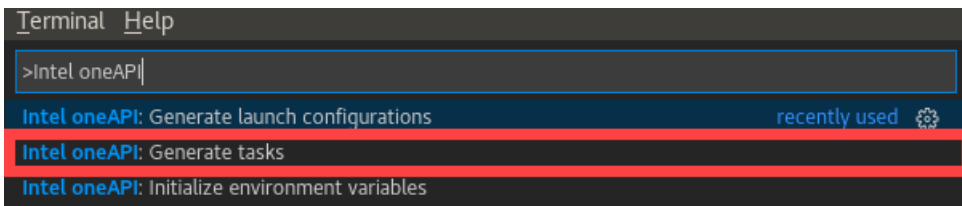
1. Press **Ctrl+Shift+P** (or **View -> Command Palette...**) to open the Command Palette.
2. Type **Intel oneAPI: Initialize environment variables**. Click on **Intel oneAPI: Initialize environment variables**.



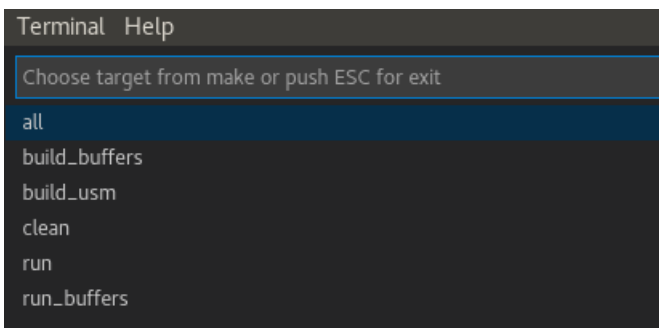
3. From the left navigation, click README.md to view instructions for the sample.

Prepare Build Tasks from Make / CMake Files

1. Press **Ctrl+Shift+P** or **View -> Command Palette...** to open the Command Palette.
2. Type **Intel oneAPI** and select **Intel oneAPI: Generate tasks**.



3. Select the build tasks (target) from your Make/CMake oneAPI project that you want to use.



4. Run the task/target by selecting **Terminal -> Run task...**
5. Select the task to run.

NOTE Not all oneAPI sample projects use CMake. The `README.md` file for each sample specifies how to build the sample. We recommend that you check out the [CMake extension for VS Code](#) that is maintained by Microsoft.

Build the Project

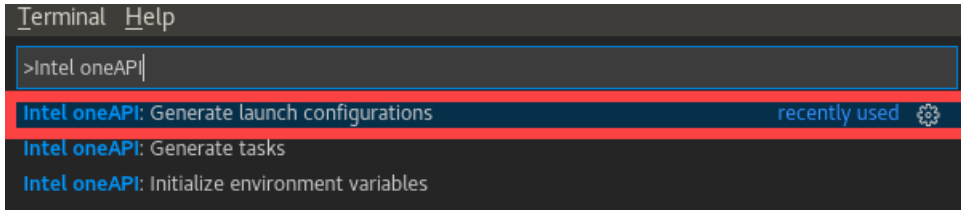
The oneAPI extensions enable the ability to prepare launch configurations for running and debugging projects created using Intel oneAPI toolkits:

1. Press **Ctrl+Shift+B** or **Terminal -> Run Build Task...** to set the default build task.
2. Select the task from the command prompt list to build your project.
3. Press **Ctrl+Shift+B** or **Terminal -> Run Build Task...** again to build your project.

Prepare Launch Configuration for Debugging

The following section describes how to use the Intel oneAPI FPGA compiler with Visual Studio Code, and use your native debugger to debug device code in emulation.

1. Press **Ctrl+Shift+P** or **View -> Command Palette...** to open the Command Palette.
2. Type **Intel oneAPI** and select **Intel oneAPI: Generate launch configurations**.

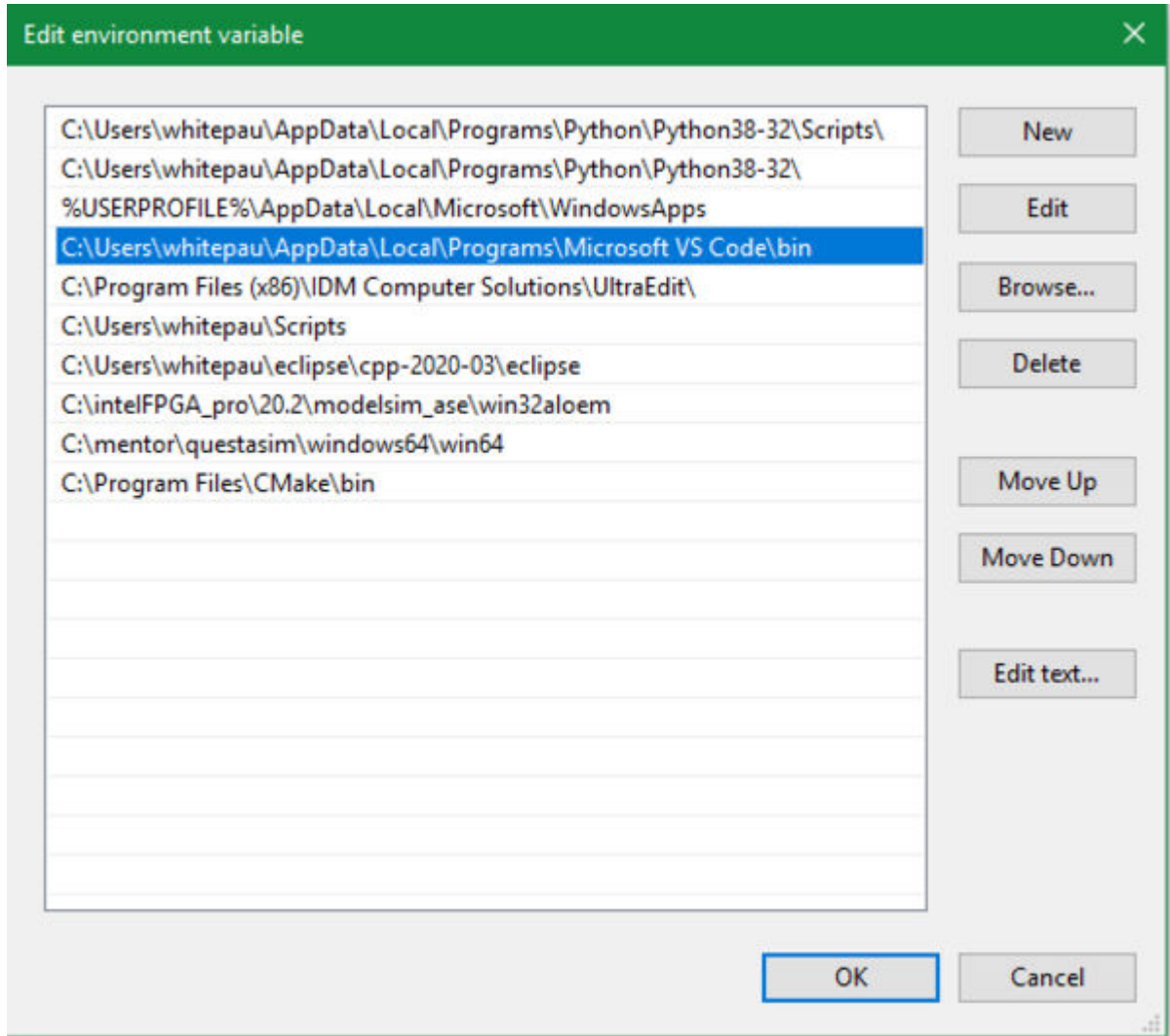


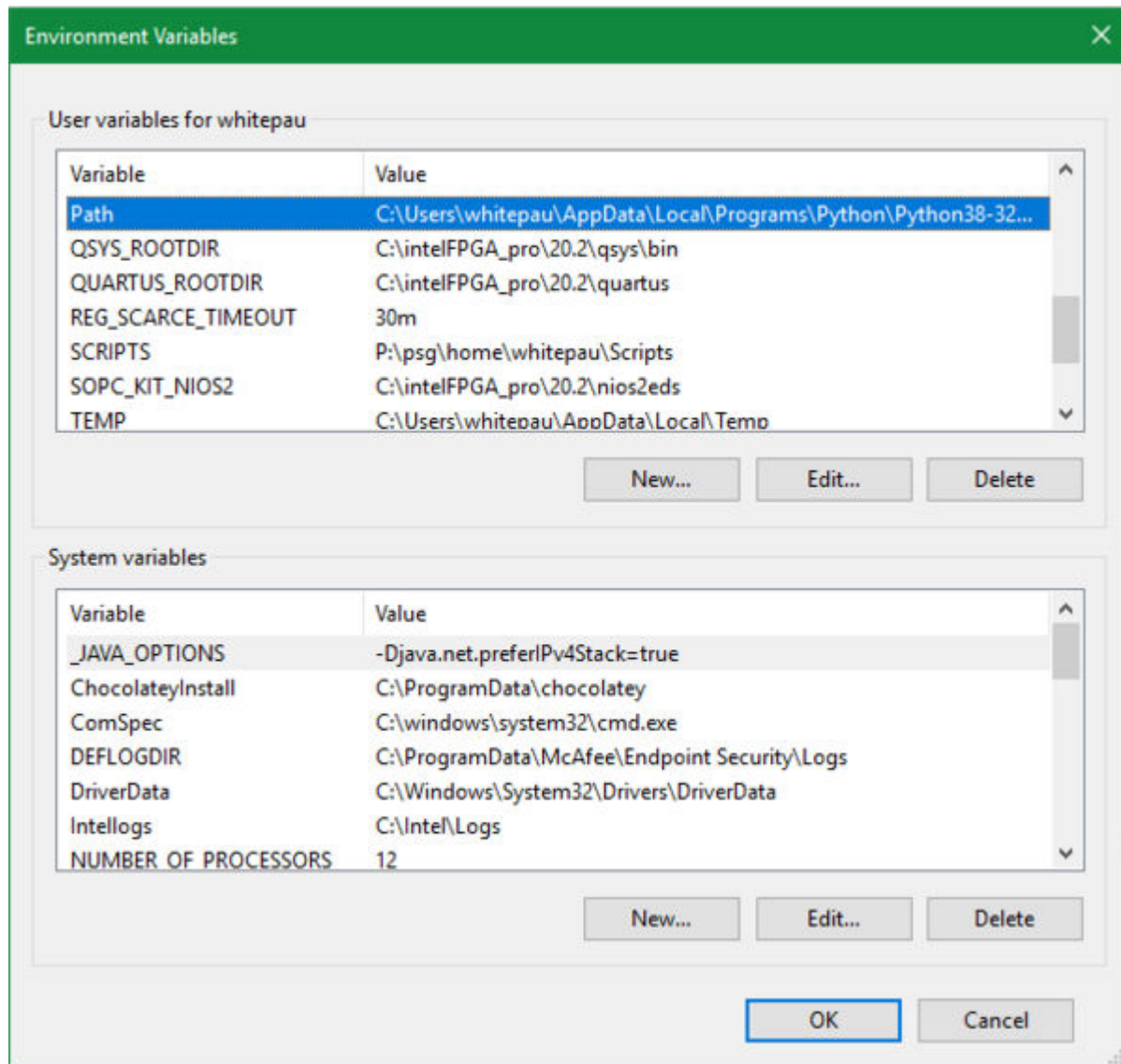
3. Select the executable (target) you want to debug.
Optional: select any task you want to run before and/or after launching the debugger (for example, build the project before debug, clean the project after debug).
4. The configuration is now available to debug and run using the gdb-oneapi debugger. You can find it in `.vscode/launch.json`. To debug and run, click on the Run icon or press `Ctrl+Shift+D`.

Using the Native Debugger to Debug Device Code in Emulation

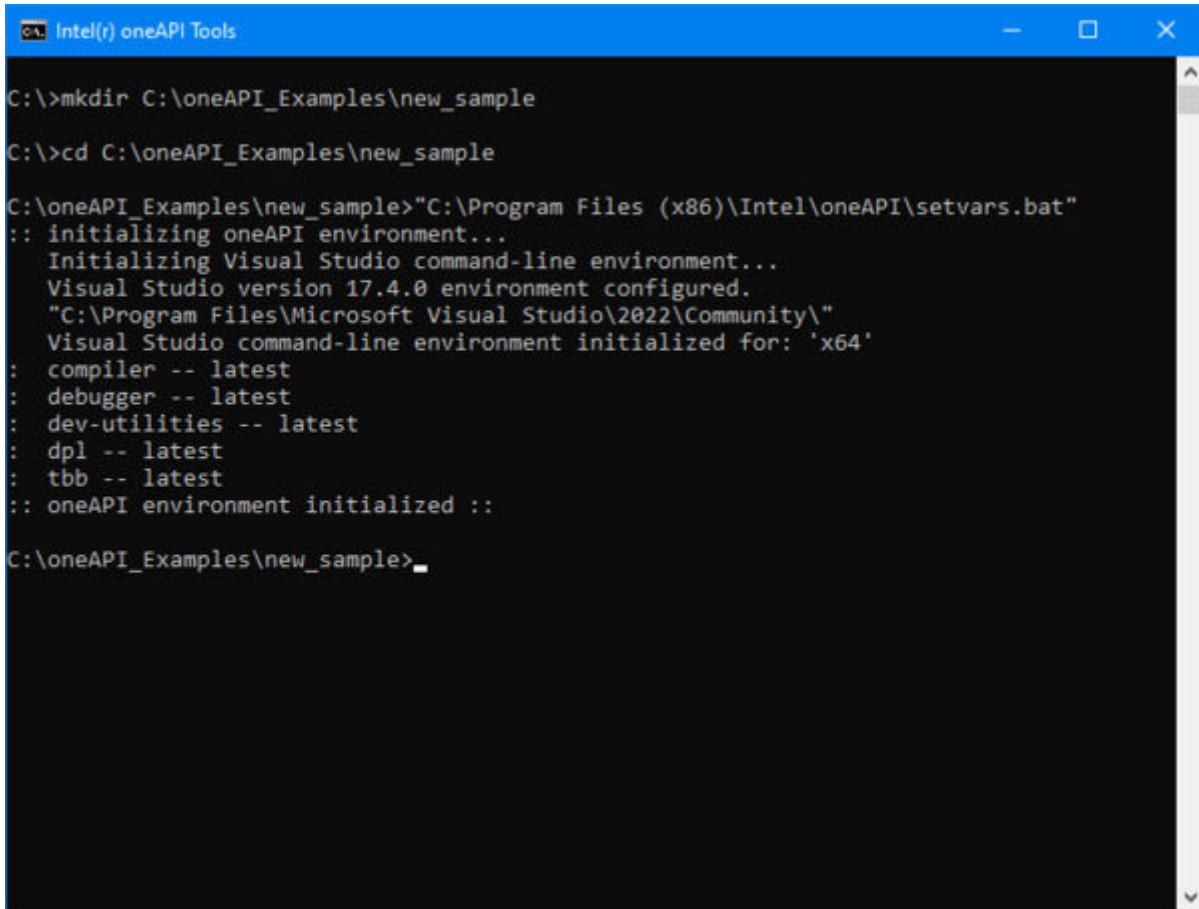
The oneAPI extensions enable the ability to prepare launch configurations for running and debugging projects created using Intel oneAPI toolkits:

1. Download and install the C/C++ extension for Visual Studio Code. You can search for it in the Extensions Marketplace, or download it from the [repository](#)
2. Configure the C/C++ extension to see the oneAPI compiler's header files. To do this, open the Extensions tab, and click the little gear icon next to the C/C++ extension, then choose 'Extension Settings'. Adjust the Cpp Standard to `c++17`.
3. Add Visual Studio Code to your PATH environment variable, so that you can call it from the command line:



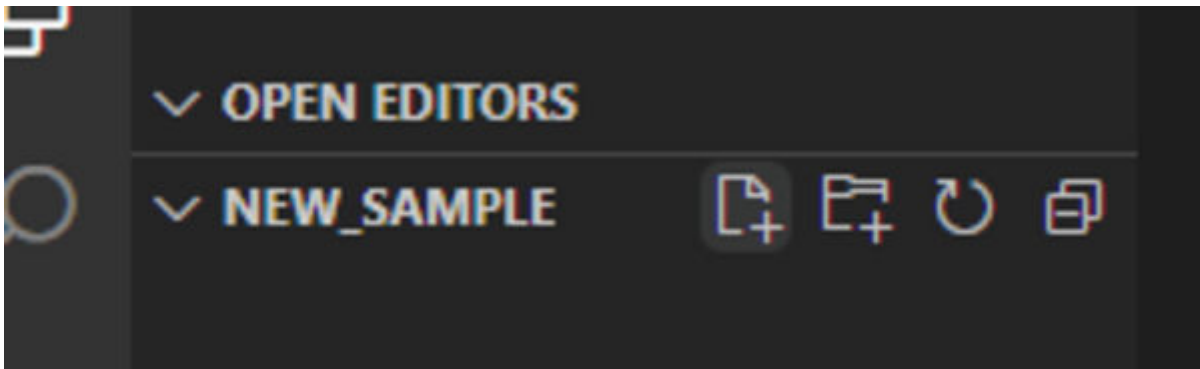


4. Create a oneAPI project by creating a folder in your filesystem, then navigating to the folder using the command line and initializing the oneAPI compiler:



```
C:\>mkdir C:\oneAPI_Examples\new_sample
C:\>cd C:\oneAPI_Examples\new_sample
C:\oneAPI_Examples\new_sample>"C:\Program Files (x86)\Intel\oneAPI\setvars.bat"
:: initializing oneAPI environment...
  Initializing Visual Studio command-line environment...
  Visual Studio version 17.4.0 environment configured.
  "C:\Program Files\Microsoft Visual Studio\2022\Community\"
  Visual Studio command-line environment initialized for: 'x64'
: compiler -- latest
: debugger -- latest
: dev-utilities -- latest
: dpl -- latest
: tbb -- latest
:: oneAPI environment initialized ::
C:\oneAPI_Examples\new_sample>
```

5. Open VS Code in this directory using the `code .` command. The Visual Studio debugger will not work properly if you do not launch VS code from a terminal like this. The visual Studio Code GUI should open. Create a new source file by clicking the 'new file' button by the project name:



Name it `test1.cpp`. Paste in this code (or write your own):

```
#include <iostream>

// oneAPI headers
#include <sycl/ext/intel/fpga_extensions.hpp>
#include <sycl/sycl.hpp>

// Forward declare the kernel name in the global scope. This is an FPGA best
// practice that reduces name mangling in the optimization reports.
class VectorAddID;
```

```

struct VectorAdd {
    int *const a_in;
    int *const b_in;
    int *const c_out;
    int len;

    void operator()() const {
        for (int idx = 0; idx < len; idx++) {
            int a_val = a_in[idx];
            int b_val = b_in[idx];
            int sum = a_val + b_val;
            c_out[idx] = sum;
        }
    }
};

constexpr int kVectSize = 256;

int main() {
    bool passed = false;

    try {

        // Use compile-time macros to select either:
        // - the FPGA emulator device (CPU emulation of the FPGA)
        // - the FPGA device (a real FPGA)
        // - the simulator device
#ifdef FPGA_SIMULATOR
        auto selector = sycl::ext::intel::fpga_simulator_selector_v;
#elif FPGA_HARDWARE
        auto selector = sycl::ext::intel::fpga_selector_v;
#else // #if FPGA_EMULATOR
        auto selector = sycl::ext::intel::fpga_emulator_selector_v;
#endif

        sycl::queue q(selector);

        auto device = q.get_device();

        std::cout << "Running on device: "
                  << device.get_info<sycl::info::device::name>().c_str()
                  << std::endl;

        // declare arrays and fill them
        // allocate in shared memory so the kernel can see them
        int *a = sycl::malloc_shared<int>(kVectSize, q);
        int *b = sycl::malloc_shared<int>(kVectSize, q);
        int *c = sycl::malloc_shared<int>(kVectSize, q);
        for (int i = 0; i < kVectSize; i++) {
            a[i] = i;
            b[i] = (kVectSize - i);
        }

        std::cout << "add two vectors of size " << kVectSize << std::endl;

        q.single_task<VectorAddID>(VectorAdd{a, b, c, kVectSize}).wait();

        // verify that c is correct

```

```

passed = true;
for (int i = 0; i < kVectSize; i++) {
    int expected = a[i] + b[i];
    if (c[i] != expected) {
        std::cout << "idx=" << i << ": result " << c[i] << ", expected ("
            << expected << ") A=" << a[i] << " + B=" << b[i] << std::endl;
        passed = false;
    }
}

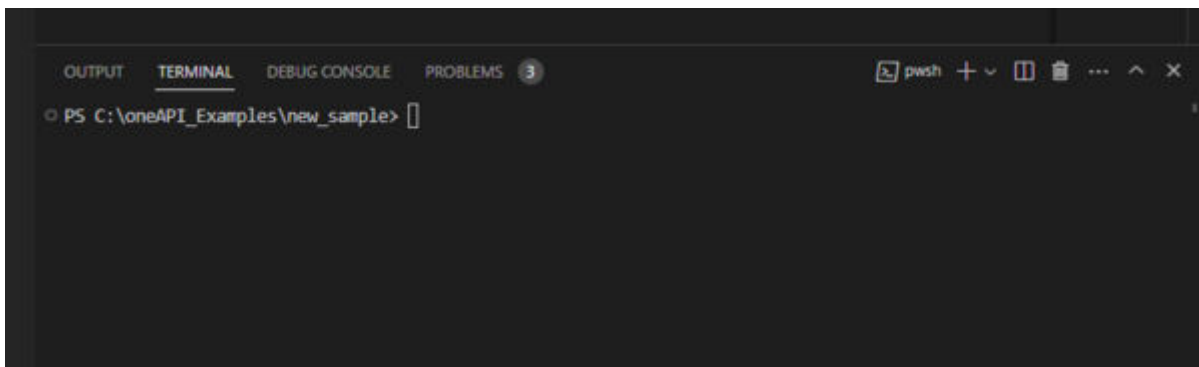
std::cout << (passed ? "PASSED" : "FAILED") << std::endl;

sycl::free(a, q);
sycl::free(b, q);
sycl::free(c, q);
}

return passed ? EXIT_SUCCESS : EXIT_FAILURE;
}

```

6. Compile and debug your oneAPI design by opening a terminal by clicking 'terminal > New Terminal'. This will open a terminal pane at the bottom of the VS Code GUI:



Click in the terminal area and press the 'enter' key to activate the terminal. If you opened VS code from a command line as described in step 3 above, you should see the oneAPI prompt. If you opened the GUI by any other means, you will need to initialize the oneAPI compiler using the set_vars.bat script.

Compile your source code using the `icx-cl` command (Be sure to include the debug flags `/DEBUG /Od!`): `icx-cl -fsycl -fintelfpga /DEBUG /Od /EHsc test1.cpp -o fpga_emu.exe`

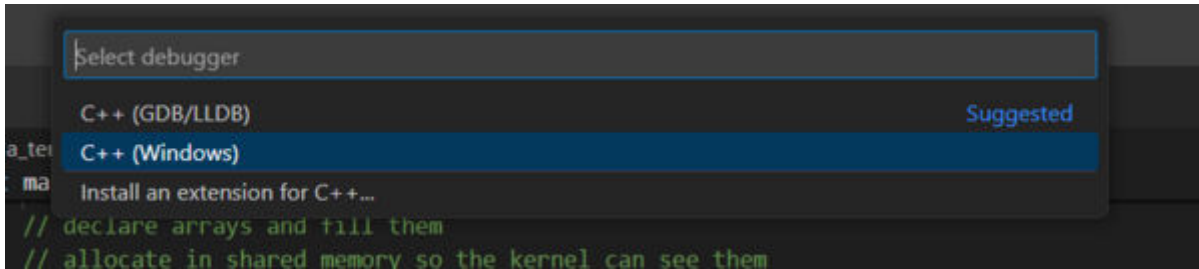
If you are compiling a code sample, the debug flags will be generated for you by CMake.

```

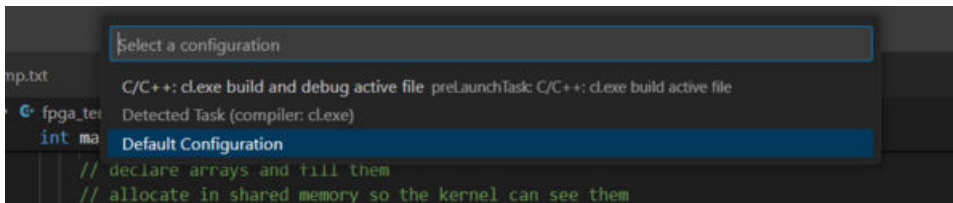
mkdir build
cd build
cmake .. -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Debug
nmake fpga_emu

```

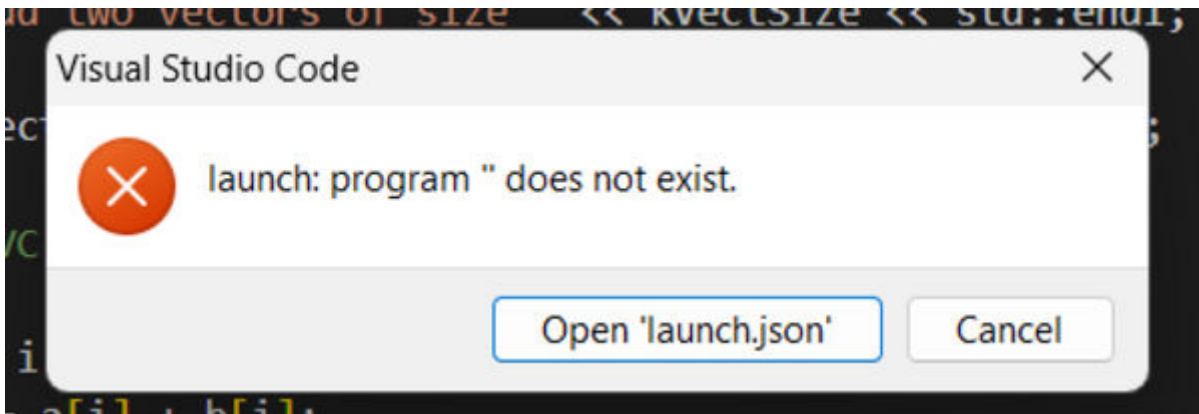
7. You can launch the debugger by pressing 'F5' on your keyboard, or by clicking the 'Run' button in the GUI. The first time you try to debug an application, you will be prompted to create a launch.json file. To do this, select the C++ (Windows) environment in the dropdown:



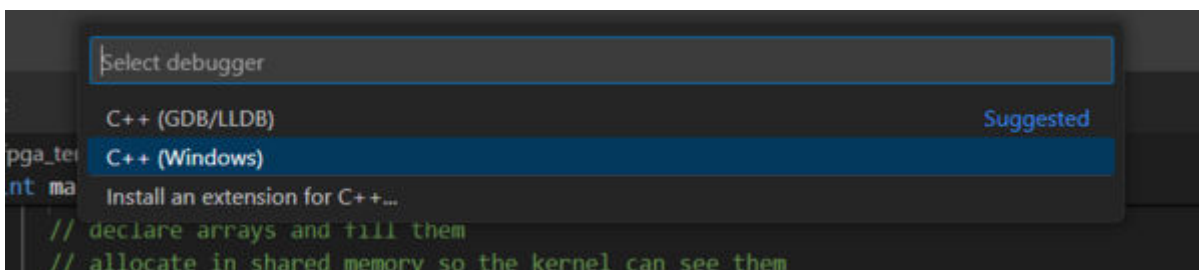
Then choose default configuration:



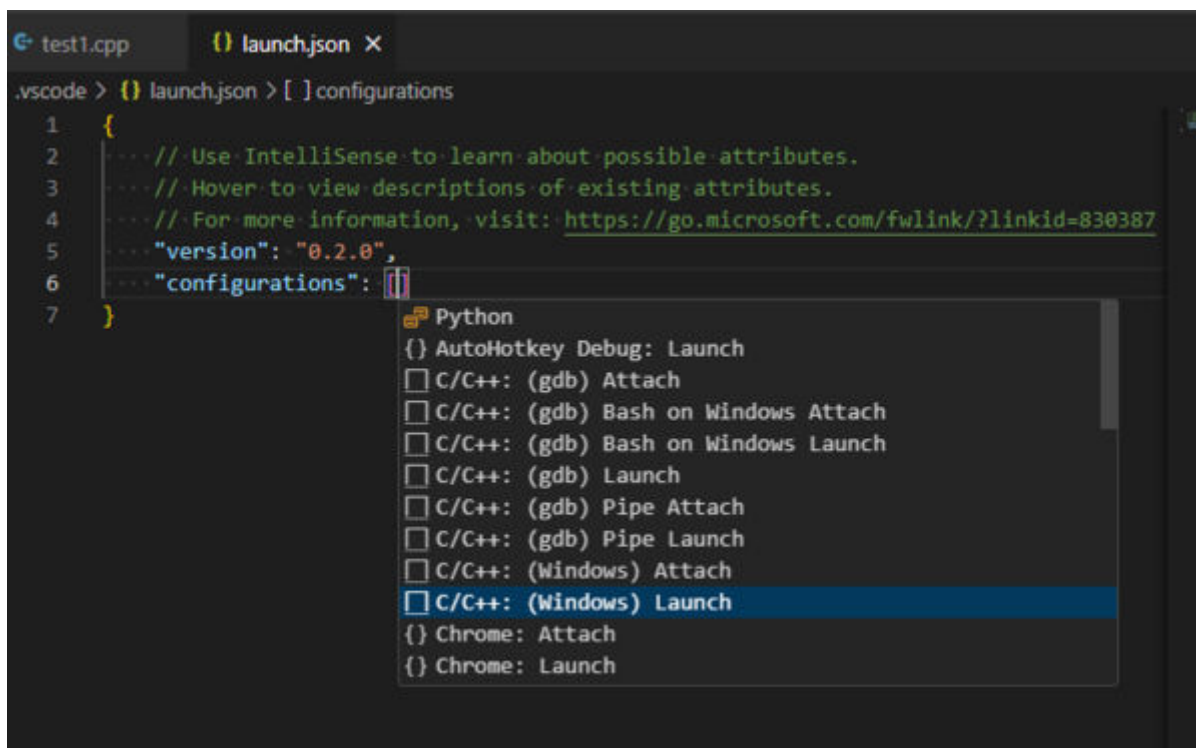
You will be prompted to open the Launch.json file:



Choose the type as C++ (Windows):

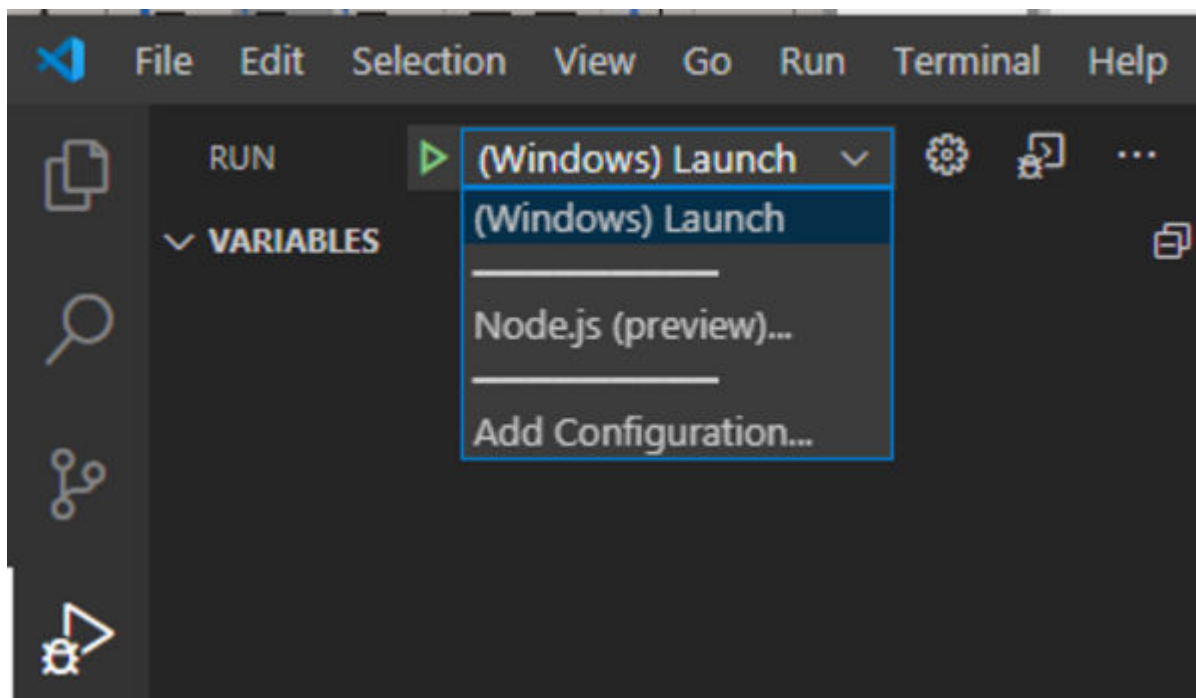


In the new 'launch.json', place your cursor between the [] and press `ctrl + spacebar` to get a prompt, and choose the 'Windows Launch' template:



Specify the program member to point to your compiled executable that you would like to debug. Save the launch.json, then close it.

Make sure that (Windows) Launch is selected in the Run bar, and click the green 'start' arrow:



The debugger will automatically stop at any breakpoints that you have set in your code. you can inspect your variables and step through the code as you would with any GUI-based debugger.

Debug, Analyze, Develop with More Extensions

There are more oneAPI extensions for Visual Studio Code which enable:

- debugging
- remote development
- connection to Intel Developer Cloud
- analysis configuration

To learn more about the extensions, see [Intel® oneAPI Extensions for Visual Studio Code*](#).

To learn more about more capabilities and options, see [Using Visual Studio Code with Intel® oneAPI Toolkits](#).

NOTE Not all oneAPI sample projects use CMake. The `README.md` file for each sample specifies how to build the sample. We recommend that you check out the [CMake extension for VSCode](#) that is maintained by Microsoft.

See [Explore SYCL* Through Samples](#) to learn more.

Next Steps

Intel® oneAPI HPC Toolkit

After successfully building a sample application, [Explore SYCL* Through Samples](#).

Maximize resources with [OpenMP Offloading](#).

Diagnose issues in the [Troubleshooting](#) section.

Learn more about the components in this toolkit:

Tool	Description
Intel® oneAPI DPC++/C++ Compiler	Use this standards-based C++ compiler with support for OpenMP* to take advantage of more cores and built-in technologies in platforms based on Intel® Xeon® Scalable processors and Intel® Core™ processors. Learn more .
Intel® C++ Compiler	Create code that takes advantage of more cores and built-in technologies in platforms based on Intel® processors. Learn more .
Intel® Fortran Compiler Classic	Generate optimized, scalable code for Intel® Xeon® Scalable processors and Intel® Core™ processors with this standards-based Fortran compiler with support for OpenMP*. Get Started .
Intel® MPI Library	Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. Use the library to develop applications that can run on multiple cluster interconnects. Get Started .
Intel® Inspector	Locate and debug threading, memory, and persistent memory errors early in the design cycle to avoid costly errors later. Learn more .
Intel® Trace Analyzer and Collector	Understand MPI application behavior across its full runtime. Learn more .
oneAPI GPU Optimization Guide	The oneAPI GPU Optimization Guide demonstrates how to improve the behavior of your software by partitioning it across the host and accelerator to specialize portions of the computation that run best on the accelerator. Specialization includes restructuring and tuning the code to create the best mapping of the application to the hardware. The value of oneAPI is that it allows each of these variations to be expressed in a common language with device-specific variants launched on the appropriate accelerator.

For more information about this toolkit, see the [Intel® oneAPI Toolkits](#) page.

Run a Sample Project with Visual Studio*

Intel® oneAPI HPC Toolkit

NOTE

An internet connection is required to download the samples for oneAPI toolkits. For information on how to use this toolkit offline, see [Developing with Offline Systems](#) in the Troubleshooting section.

NOTE If you are using Visual Studio 2019, samples will only work with version 16.4.0 and later.

NOTE If you are using Visual Studio with FPGA, see [FPGA Workflows on Third-Party IDEs for Intel® oneAPI Toolkits](#).

To watch a video presentation of how to create a project, see [Intel® oneAPI Visual Studio Samples Browser](#).

Define the `SETVARS_CONFIG` Environment Variable:

The `SETVARS_CONFIG` environment variable is not automatically defined during installation, you must add it to your environment before starting Visual Studio using the following. This only needs to be set once.

1. Open a command window.
2. Set system variables for Visual Studio.

```
setx SETVARS_CONFIG " "
```

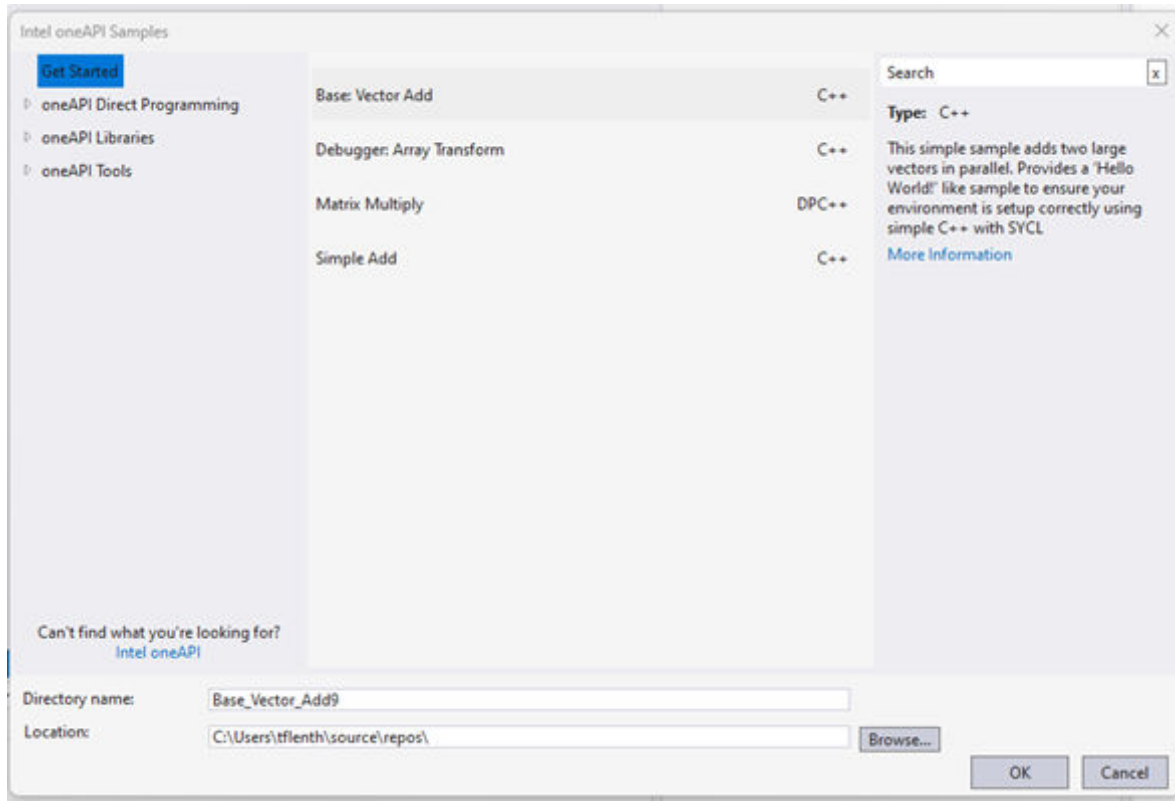
To run the **Matrix Multiplication** sample project:

1. Open Microsoft Visual Studio.
2. For Visual Studio 2019 and 2022, a page may display showing recent projects. Click **Continue without code**.

NOTE In the next step, you will look for a menu named **Extensions > Intel** or **Intel**. If you do not see either of these menu choices, then the plug-ins have not been installed. See [Troubleshooting](#) to fix the plug-ins.

3. From the top menu, select:
 - Visual Studio 2019 and 2022: **Extensions > Intel > Browse Intel oneAPI Samples**

The samples screen will open:



4. On the left side, select **Get Started**.
5. On the right side, select **Matrix Mul**.
6. Click **OK**.
7. From the Solution Explorer, right click on `matrix_mul_mkl` and select **Rebuild**.
8. From the top menu, select **Debug > Start Without Debugging**. The debug results will appear:

```

Microsoft Visual Studio Debug Console
Problem size: c(600,2400) = a(600,1200) * b(1200,2400)
Device: Intel(R) HD Graphics P530
success - The results are correct!

C:\Users\Administrator\source\repos\Matrix_Multiplication\x64\Debug\matrix_mul_mkl.exe (process 8820) exited w
.
Press any key to close this window . . .

```

See [Explore SYCL* Through Samples](#) to learn more.

Next Steps

Intel® oneAPI HPC Toolkit

After successfully building a sample application, [Explore SYCL* Through Samples](#).

Maximize resources with [OpenMP Offloading](#).

Diagnose issues in the [Troubleshooting](#) section.

Learn more about the components in this toolkit:

Tool	Description
Intel® oneAPI DPC++/C++ Compiler	Use this standards-based C++ compiler with support for OpenMP* to take advantage of more cores and built-in technologies in platforms based on Intel® Xeon® Scalable processors and Intel® Core™ processors. Learn more.
Intel® C++ Compiler	Create code that takes advantage of more cores and built-in technologies in platforms based on Intel® processors. Learn more.
Intel® Fortran Compiler Classic	Generate optimized, scalable code for Intel® Xeon® Scalable processors and Intel® Core™ processors with this standards-based Fortran compiler with support for OpenMP*. Get Started.
Intel® MPI Library	Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. Use the library to develop applications that can run on multiple cluster interconnects. Get Started.
Intel® Inspector	Locate and debug threading, memory, and persistent memory errors early in the design cycle to avoid costly errors later. Learn more.
Intel® Trace Analyzer and Collector	Understand MPI application behavior across its full runtime. Learn more.
oneAPI GPU Optimization Guide	The oneAPI GPU Optimization Guide demonstrates how to improve the behavior of your software by partitioning it across the host and accelerator to specialize portions of the computation that run best on the accelerator. Specialization includes restructuring and tuning the code to create the best mapping of the application to the hardware. The value of oneAPI is that it allows each of these variations to be expressed in a common language with device-specific variants launched on the appropriate accelerator.

For more information about this toolkit, see the [Intel® oneAPI Toolkits](#) page.

Using Containers

oneAPI containers are not officially supported or tested on Windows* host systems. However, the instructions in the [Using Containers](#) section of the Get Started with the Intel® oneAPI Base Toolkit for Linux* should apply to a Windows host. There are known issues regarding access to the GPU on a Windows host.

Using Cloud CI Systems

Cloud CI systems allow you to build and test your software automatically. See the [repo in github](#) for examples of configuration files that use oneAPI for the popular cloud CI systems.

Next Steps

Intel® oneAPI HPC Toolkit

After successfully building a sample application, [Explore SYCL* Through Samples.](#)

Maximize resources with [OpenMP Offloading.](#)

Diagnose issues in the [Troubleshooting](#) section.

Learn more about the components in this toolkit:

Tool	Description
------	-------------

Intel® oneAPI DPC++/C++ Compiler	Use this standards-based C++ compiler with support for OpenMP* to take advantage of more cores and built-in technologies in platforms based on Intel® Xeon® Scalable processors and Intel® Core™ processors. Learn more.
Intel® C++ Compiler	Create code that takes advantage of more cores and built-in technologies in platforms based on Intel® processors. Learn more.
Intel® Fortran Compiler Classic	Generate optimized, scalable code for Intel® Xeon® Scalable processors and Intel® Core™ processors with this standards-based Fortran compiler with support for OpenMP*. Get Started.
Intel® MPI Library	Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. Use the library to develop applications that can run on multiple cluster interconnects. Get Started.
Intel® Inspector	Locate and debug threading, memory, and persistent memory errors early in the design cycle to avoid costly errors later. Learn more.
Intel® Trace Analyzer and Collector	Understand MPI application behavior across its full runtime. Learn more.
oneAPI GPU Optimization Guide	The oneAPI GPU Optimization Guide demonstrates how to improve the behavior of your software by partitioning it across the host and accelerator to specialize portions of the computation that run best on the accelerator. Specialization includes restructuring and tuning the code to create the best mapping of the application to the hardware. The value of oneAPI is that it allows each of these variations to be expressed in a common language with device-specific variants launched on the appropriate accelerator.

For more information about this toolkit, see the [Intel® oneAPI Toolkits](#) page.

Using Intel Compilers and Libraries for the Best Experience

Intel® oneAPI HPC Toolkit

The Intel® oneAPI HPC Toolkit has the tools you need to start building and analyzing applications out of the box. Start compiling and optimizing your code today using all the tools and libraries in this suite.

Component	Description
Intel® C++ Compiler and Intel® Fortran Compiler Classic	The Intel® C++ and Intel® Fortran optimizing compilers create fast code for modern processors. They use the latest instruction sets, auto-vectorize code for supporting/ utilizing wider vector registers, and highly-tuned parallel models like OpenMP*, and Intel® TBB. The compilers offer broad support for the latest C, C++, and Fortran standards.
Intel® MPI Library	<p>Intel® MPI Library is a multifabric message-passing library that implements the open-source Message Passing Interface (MPI) specification. Use the library to create, maintain, and test advanced, complex applications that perform better on HPC clusters based on Intel® processors.</p> <p>Develop applications that can run on multiple cluster interconnects chosen by the user at run time.</p> <p>Quickly deliver maximum end-user performance without having to change the software or operating environment.</p> <p>Reduce the time to market by linking to one library and deploying on the latest optimized fabrics.</p>

Component	Description
Intel® Inspector	<p>Memory errors and non-deterministic threading errors are difficult to find without the right tool. Intel® Inspector is designed to find these errors. It is a dynamic memory and threading error debugger for C, C++, and Fortran applications that run on Windows* and Linux* operating systems. It helps you:</p> <p>Save money: Locate the root cause of memory and threading errors before you release.</p> <p>Save time: Quickly debug intermittent races and deadlocks.</p> <p>Save data: Find errors like missing or redundant cache flushes for persistent memory implementations.</p> <p>Save effort: Use the stand-alone interface, Microsoft Visual Studio* plug-in, or command line. No special compilers or builds are required.</p>
Intel® Trace Analyzer and Collector	<p>Use this graphical tool to understand MPI application behavior across its full runtime. It helps you:</p> <p>Find temporal dependencies and bottlenecks in your code</p> <p>Check the correctness of your application</p> <p>Locate potential programming errors, buffer overlaps, and deadlocks</p> <p>Visualize and understand parallel application behavior</p> <p>Evaluate profiling statistics and load balancing</p> <p>Analyze performance of subroutines or code blocks</p> <p>Learn about communication patterns, parameters, and performance data</p> <p>Identify communication hot spots</p> <p>Decrease time to solution and increase application efficiency</p>

Troubleshooting

Potential errors and how to avoid or fix them.

Issue	How to fix
Errors that occur during installation or directly after installation. Undefined error appears when building a sample.	<p>See the Troubleshooting page of the Intel® oneAPI Toolkits Installation Guide for Windows* OS.</p> <p>If the sample was built by using <code>cmake</code> and then <code>make</code>, more details to diagnose the reason for your error are available by rebuilding with the <code>VERBOSE</code> option:</p> <pre>make VERBOSE=1</pre>
Developing with Offline Systems: I need to develop on a system that is not connected to the internet.	<p>If you are using an offline system, download the samples from a system that is internet connected and transfer the sample files to your offline system.</p> <p>After you have downloaded the samples, follow the instructions in the <code>README.md</code> file. The readme file is located in the folder of the sample you are interested in.</p> <p>The samples can be downloaded from the code samples repository.</p>

Issue	How to fix
<p>Unable to install toolkits or access libraries.</p> <p>When building a sample on Windows, error MSB8020: compiler not found appears.</p>	<p>The full set of documentation can be downloaded from Downloadable Documentation.</p> <p>Verify that you meet the System Requirements listed on the Intel® oneAPI Base Toolkit product page</p> <p>When compiling, this error appears:</p> <p>error MSB8020: The build tools for Intel® oneAPI DPC++ Compiler (Platform Toolset = 'Intel® oneAPI DPC++ Compiler') cannot be found. To build using the Intel® oneAPI DPC++ Compiler build tools, please install Intel® oneAPI DPC++ Compiler build tools. Alternatively, you may upgrade to the current Visual Studio tools by selecting the Project menu or right-click the solution, and then select "Retarget solution."</p> <p>This issue can occur if Visual Studio is installed after a oneAPI Toolkit, resulting in the toolkit plug-ins being absent.</p> <p>To fix this problem, uninstall and reinstall your oneAPI toolkits:</p> <ol style="list-style-type: none"> 1. In Windows settings, open Add or Remove Programs. 2. Click on Intel® oneAPI toolkits. 3. Click Uninstall. 4. After the Uninstall process is complete, verify that Visual Studio and/or MSBuild are installed. 5. Install your oneAPI toolkits following the same procedure you used the first time. To download the software, go to the Intel® oneAPI Toolkits page.
<p>When running a sample on Windows, SPIRV internal error appears.</p>	<p>The SPIRV error could be occurring if the default processor is a GPU but you are running a sample for a CPU. To force the sample to compile on the CPU:</p> <ol style="list-style-type: none"> 1. In the src file for the sample, open the .cpp file. For example, C:\samples\vector-add\src\vector-add-buffers.cpp. 2. Find the #else statement that starts with //The default device selector will select the most performant device. 3. Replace the entire #else #endif statement with: <pre data-bbox="824 1583 1442 1759">#else // The default device selector will select the most performant device. // default_selector d_selector; cpu_selector d_selector; #endif</pre> <ol style="list-style-type: none"> 4. Compile the sample again, then run the sample.
<p>In Visual Studio, the Intel menu does not appear and/or the samples from the installed toolkit are missing.</p>	<p>If Visual Studio is installed after a oneAPI toolkit, the plug-ins that come as part of the toolkit are not installed.</p>

Issue	How to fix
Problems connecting from behind a proxy	<p>To fix this problem, uninstall and reinstall your oneAPI toolkits:</p> <ol style="list-style-type: none"> 1. In Windows settings, open Add or Remove Programs. 2. Select on Intel® oneAPI toolkits. 3. Click Uninstall. 4. After the Uninstall process is complete, verify that Visual Studio 2017 and/or Visual Studio 2019 is installed. 5. Install your oneAPI toolkits following the same procedure you used the first time. The software is available for download at the Intel® oneAPI Toolkits page. <p>The oneAPI CLI Samples Browser does not work with system proxy settings and does not support WPAD proxy.</p> <p>If you are unable to access the samples from the oneAPI CLI Samples Browser, set the value of <code>https_proxy</code> and <code>http_proxy</code>, then run <code>oneapi-cli.exe</code> in the same command line window. To set the proxy enter this command, where <code>your.proxy</code> is the name of your proxy server and <code>8080</code> is your default port.</p> <pre>set http_proxy=http://your.proxy:8080 set https_proxy=https://your.proxy:8080</pre> <p>After you set the proxy, return to the instructions for Running a Sample Using the Command Line and try again.</p>
Long-running applications are terminated.	<p>If your application has long-running GPU compute workloads in native environments, those workloads may be terminated by the hardware. This functionality is intended to prevent process hangs to run indefinitely; however, this behavior is not recommended for virtualizations or other standard usages of GPU, such as gaming.</p> <p>A workload that takes more than four seconds for GPU hardware to execute is considered a long-running workload. By default, individual threads that qualify as long-running workloads are considered hung and are terminated.</p> <p>To modify the termination process, see Timeout Detection and Recovery (TDR) Registry Keys from Microsoft.</p>

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.