INTEL ONEAPI – SYCL & TOOL OVERVIEW

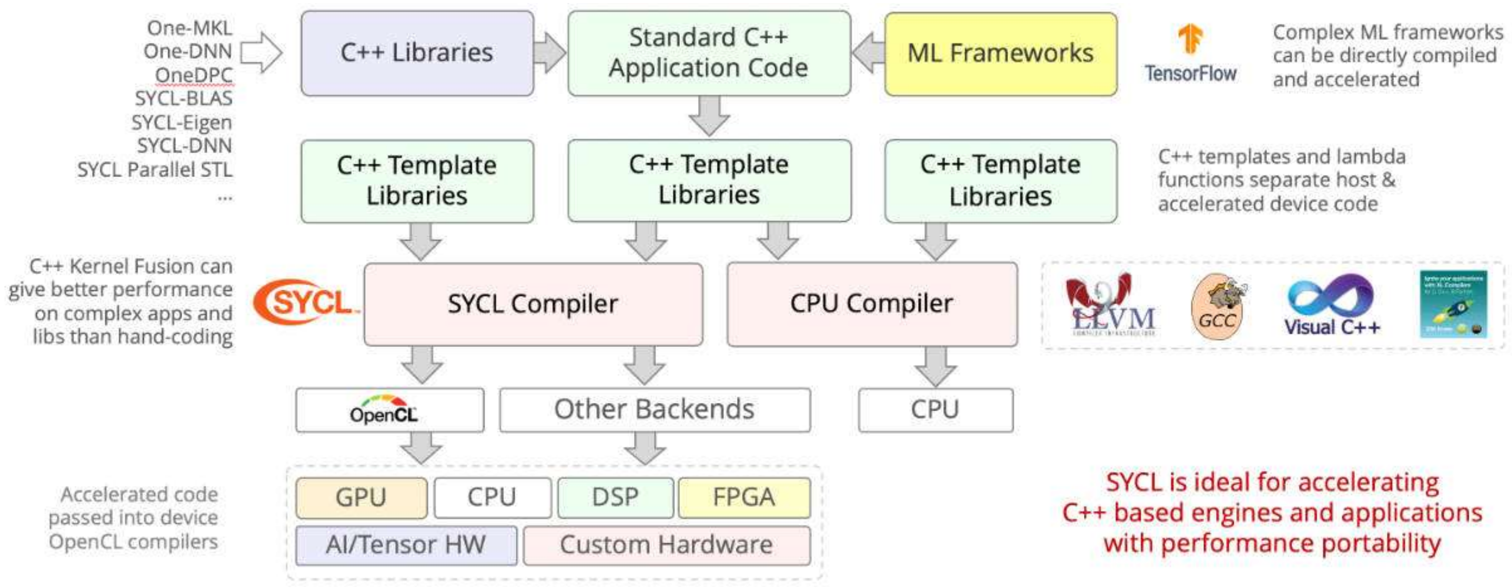
Version 2023.11

Agenda

- SYCL™ Overview
- CUDA® Overview
- SYCLomatics Overview
- oneAPI Direct vs API-based Programming
- SYCLomatics Environment Setup
- SYCLomatics Workshop: SortingNetwork

SYCL OVERVIEW

Overview of SYCL™


One-MKL
One-DNN
OneDPC
SYCL-BLAS
SYCL-Eigen
SYCL-DNN
SYCL Parallel STL
...
C++ Libraries
Standard C++ Application Code
ML Frameworks
TensorFlow
Complex ML frameworks can be directly compiled and accelerated
C++ Template Libraries
C++ Template Libraries
C++ Template Libraries
C++ templates and lambda functions separate host & accelerated device code
C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding
SYCL
SYCL Compiler
CPU Compiler
LLVM
GCC
Visual C++
OpenCL
Other Backends
CPU
Accelerated code passed into device OpenCL compilers
GPU
CPU
DSP
FPGA
AI/Tensor HW
Custom Hardware
SYCL is ideal for accelerating C++ based engines and applications with performance portability

- Initiative by Khronos Group.
- Data/compute parallelism in C++
- Influence direction of C++ standard for supporting heterogenous compute.
- Modern heterogenous system (CPU, GPU, FPGA, accelerator)
- https://www.khronos.org/files/sycl/sycl-2020-reference-guide.pdf

Overview of SYCL™

```
1   #include <sycl/sycl.hpp>
    #include <iostream>

2   using namespace sycl;
    using namespace std;

    int main() {
        // Allocate data buffer on host memory
        int hostData[512];

        // Create default workQueue to SYCL device
3       queue workQueue;

        // Using {} block to ensure all SYCL tasks complete
        // before we move to next host code execution
        {
            // Create buffer that contains host data
4           buffer computeBuffer { hostData };

5           workQueue.submit([&](handler & cmd_group_hdl)
            {
6               accessor kernelResult{computeBuffer, cmd_group_hdl};

                cmd_group_hdl.parallel_for(512, [=](auto  idx){   7
                    // Code logics for device kernel
                    kernelResult[idx] = idx;                   8
                }); // End of kernel function
            }); // End of command group
        }

         for (int=0; i<512; i++) {
             cout << "result["<<i<<"]="<<hostData[i]<<endl;
         }

         return 0;
    }
```

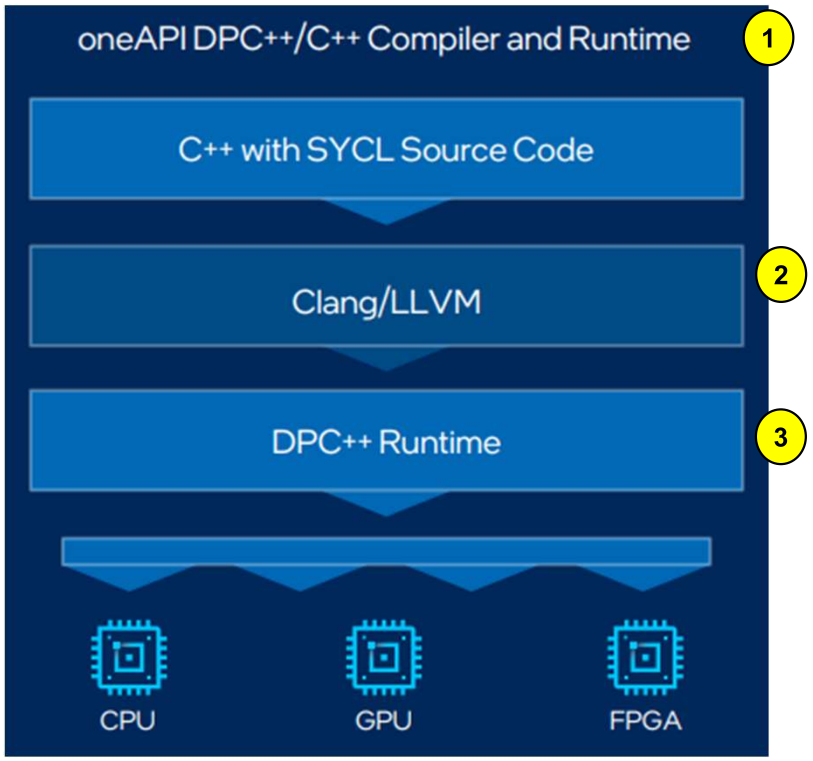
Host code

1. SYCL header file
2. Avoids need for "sycl::"
3. **Queue** - Interface where host code submits parallelized task to **SYCL device** (CPU, GPU, FPGA, etc)
4. **Buffer** – shared array used in kernel function. Can be 3-dimension
5. **Command group** indicates to compiler the scope of the execution and ownership of data/storage in SYCL device. Command group is submitted to SYCL device for parallel execution.
6. **Accessor** – buffer that allows kernel function to access host memory.
7. **Handler** – dispatch command group/kernel to SYCL device

Device code

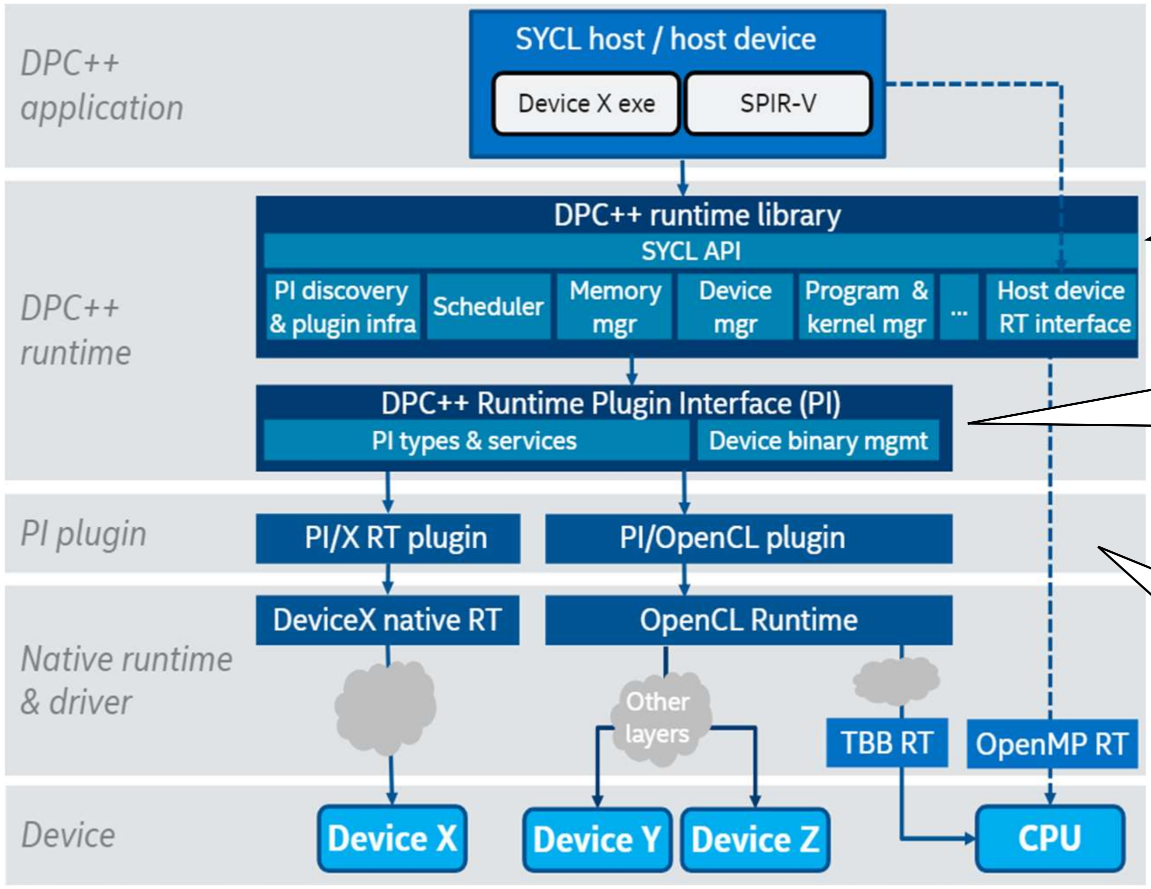
8. **Kernel function** – scope compiled by device compiler and executed on SYCL device

Overview of SYCL™


oneAPI DPC++/C++ Compiler and Runtime
1
C++ with SYCL Source Code
Clang/LLVM
2
DPC++ Runtime
3
CPU
GPU
FPGA

1. Intel implements SYCL specification through Intel oneAPI DPC++.
2. Intel oneAPI Data Parallel C++/C++ compiler (icx) is SYCL compatible compiler built using LLVM compiler infrastructure and Clang front end.
3. DPC++ Runtime provides functionalities e.g. SYCL device discovery & scheduling, host and device memory management, kernel function runtime.

Overview of SYCL™ Architecture



DPC++ Runtime is device-agnostic.

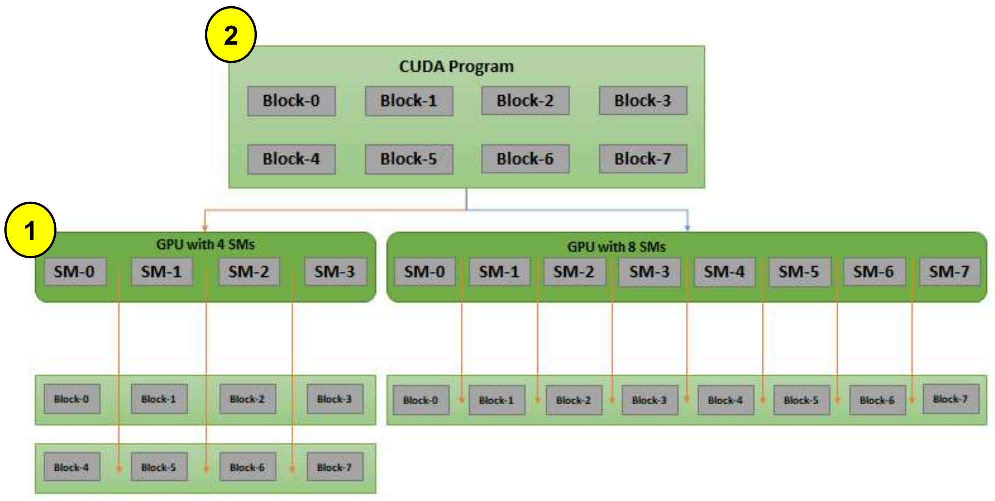
DPC++ Runtime Plugin Interface is based on subset of OpenCL 1.2 runtime and other extension APIs.

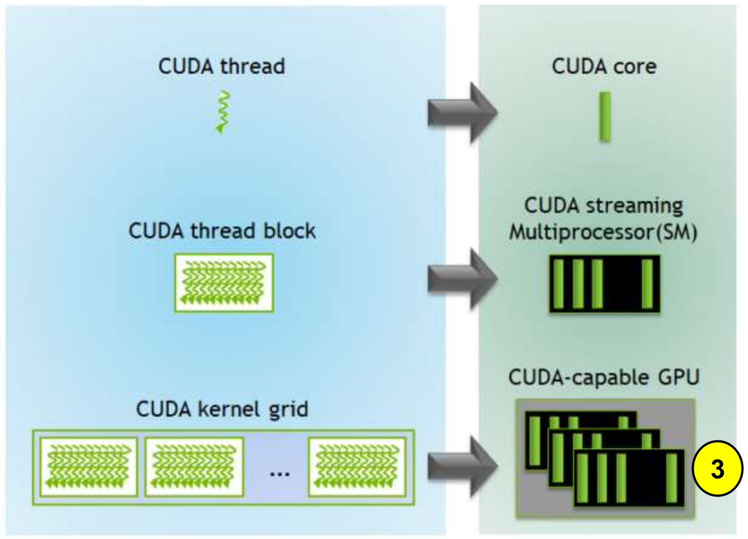
SYCL Device has its own device plugin (e.g. OpenCL for FPGA or native for GPU)

https://intel.github.io/llvm-docs/design/PluginInterface.html#the-dpc-runtime-plugin-interface

CUDA OVERVIEW

Overview of CUDA®





1. CUDA architecture is built around scalable array of SM (streaming multiprocessors). A SM is multi-threaded.

2. A CUDA program is subdivided into **thread block.**

3. Depending on NVIDIA GPU Architecture, #CUDA core per SM and #SM per GPU card is different. **Compute/Data parallelism** is done by **CUDA runtime** that schedules CUDA program blocks onto SM.

Overview of CUDA ®

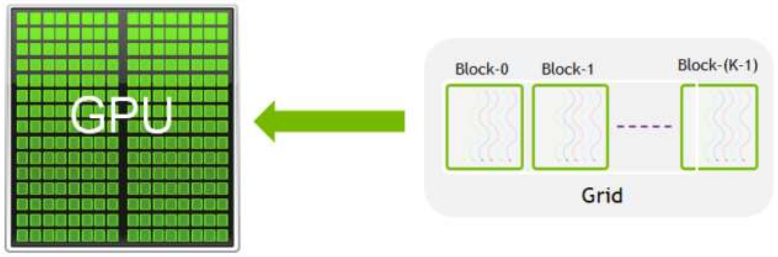


Figure 2. CUDA kernels are subdivided into blocks.

```
// Kernel - Adding two matrices MatA and MatB
__global__ void MatAdd(float MatA[N][N], float MatB[N][N],
float MatC[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        MatC[i][j] = MatA[i][j] + MatB[i][j];
}

int main()
{
    ...
    // Matrix addition kernel launch from host code
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks((N + threadsPerBlock.x -1) / threadsPerBlock.x, (N+threadsPerBlock.y -1) / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(MatA, MatB, MatC);
    ...
}
```

1 (lines 5–8) · 2 (line 17)

CUDA kernel

1 CUDA block and threads have built-in 3D variables:-

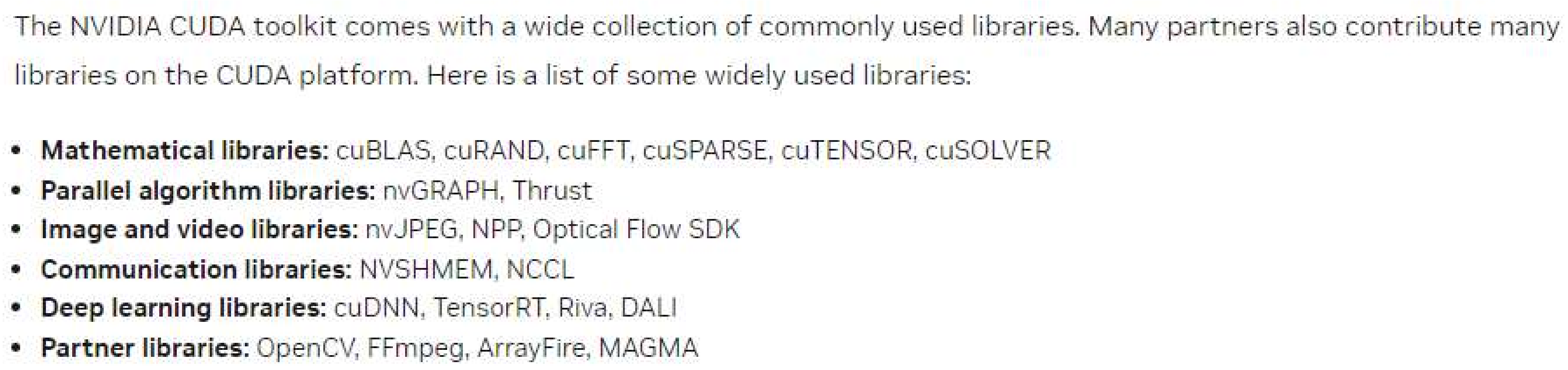
- Thread: `threadIndex.x|y|z`
- Block: `blockIndex.x|y|z`

CUDA thread block dimension is accessible within kernel by built-in `blockDim`.

2 `Func<<<threadBlock, threadsInBlock>>>(args…)`
Marks CUDA kernel.

Overview of CUDA ® Libraries

- NVIDIA provides a layer on top of the CUDA platform called **CUDA-X** , which is a collection of libraries, tools, and technologies.
- GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning, and graph analytics.



The NVIDIA CUDA toolkit comes with a wide collection of commonly used libraries. Many partners also contribute many libraries on the CUDA platform. Here is a list of some widely used libraries:

- **Mathematical libraries:** cuBLAS, cuRAND, cuFFT, cuSPARSE, cuTENSOR, cuSOLVER
- **Parallel algorithm libraries:** nvGRAPH, Thrust
- **Image and video libraries:** nvJPEG, NPP, Optical Flow SDK
- **Communication libraries:** NVSHMEM, NCCL
- **Deep learning libraries:** cuDNN, TensorRT, Riva, DALI
- **Partner libraries:** OpenCV, FFmpeg, ArrayFire, MAGMA

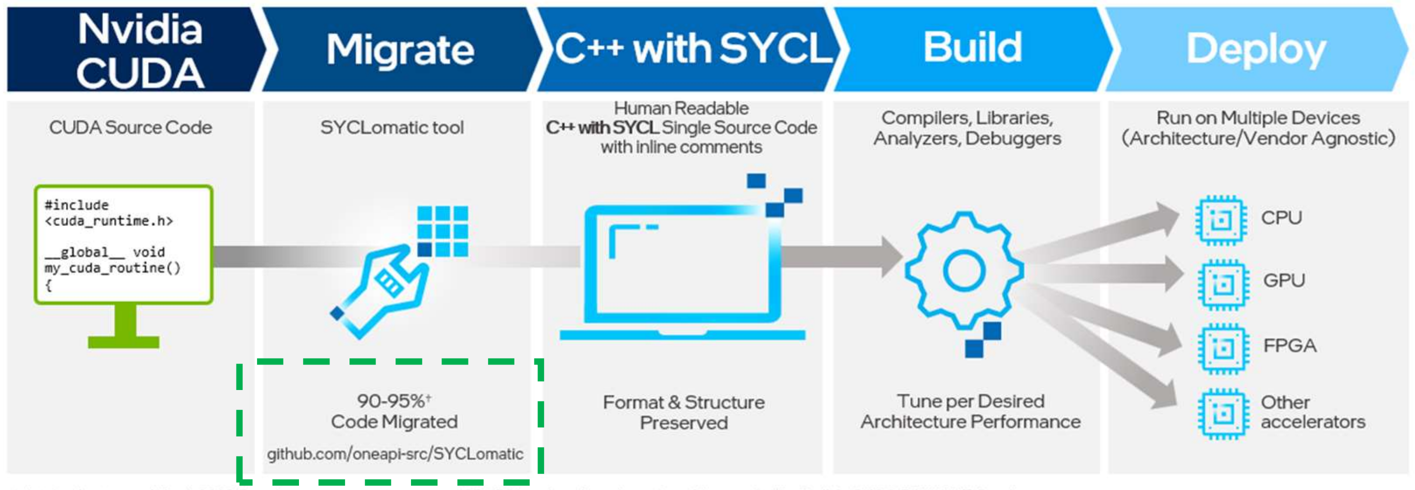
https://developer.nvidia.com/blog/cuda-refresher-the-gpu-computing-ecosystem/

SYCLOMATICS OVERVIEW

Overview of SYCLomatics

Also known as **DPC++ Compatibility Tool**

CUDA[‡] to SYCL[‡] Code Migration & Development Workflow



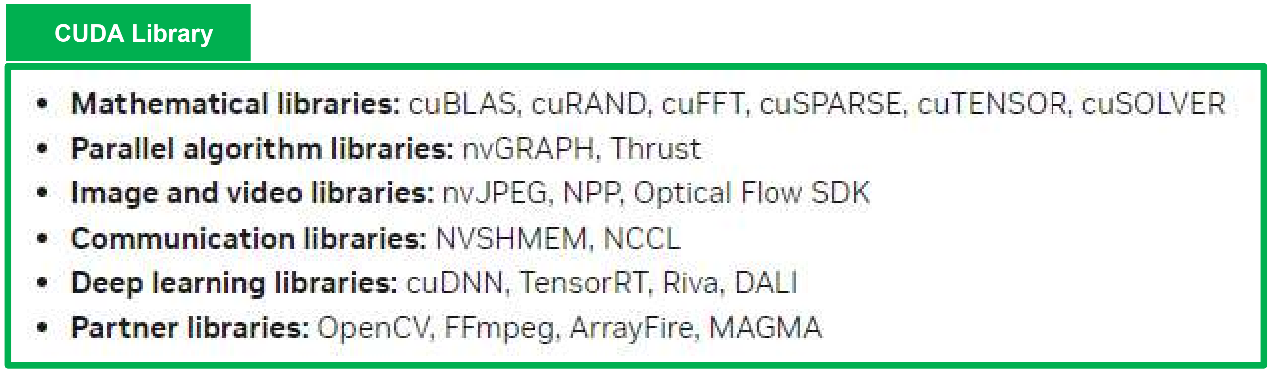
Nvidia CUDA
Migrate
C++ with SYCL
Build
Deploy
CUDA Source Code
SYCLomatic tool
Human Readable C++ with SYCL Single Source Code with inline comments
Compilers, Libraries, Analyzers, Debuggers
Run on Multiple Devices (Architecture/Vendor Agnostic)
#include <cuda_runtime.h>
__global__ void my_cuda_routine() {
CPU
GPU
FPGA
Other accelerators
90-95%†
Code Migrated
github.com/oneapi-src/SYCLomatic
Format & Structure Preserved
Tune per Desired Architecture Performance

† Intel estimates as of March 2023. Based on measurements on a set of 85 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.
‡ Other names and brands may be claimed as the property of others. SYCL is a trademark of the Khronos Group Inc.

https://www.intel.com/content/www/us/en/docs/dpcpp-compatibility-tool/developer-guide-reference/2023-2/overview.html

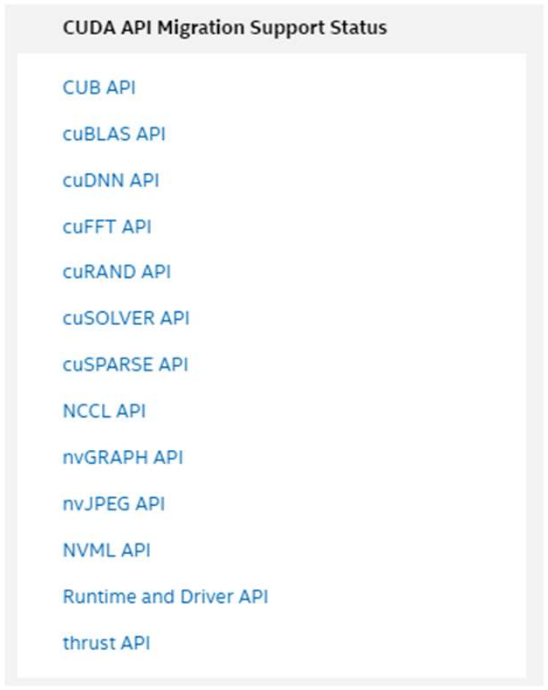
Overview of SYCLomatics

- SYCLomatic supports specific version of CUDA Toolkit.
  Example, 2023.3 SYCLomatics tools support CUDA ver12.1
- SYCLomatics supports most popular CUDA libraries API but it is not a complete list. Please check https://www.intel.com/content/www/us/en/docs/dpcpp-compatibility-tool/developer-guide-reference/2023-2/cuda-api-migration-support-status.html



CUDA Library

- **Mathematical libraries:** cuBLAS, cuRAND, cuFFT, cuSPARSE, cuTENSOR, cuSOLVER
- **Parallel algorithm libraries:** nvGRAPH, Thrust
- **Image and video libraries:** nvJPEG, NPP, Optical Flow SDK
- **Communication libraries:** NVSHMEM, NCCL
- **Deep learning libraries:** cuDNN, TensorRT, Riva, DALI
- **Partner libraries:** OpenCV, FFmpeg, ArrayFire, MAGMA



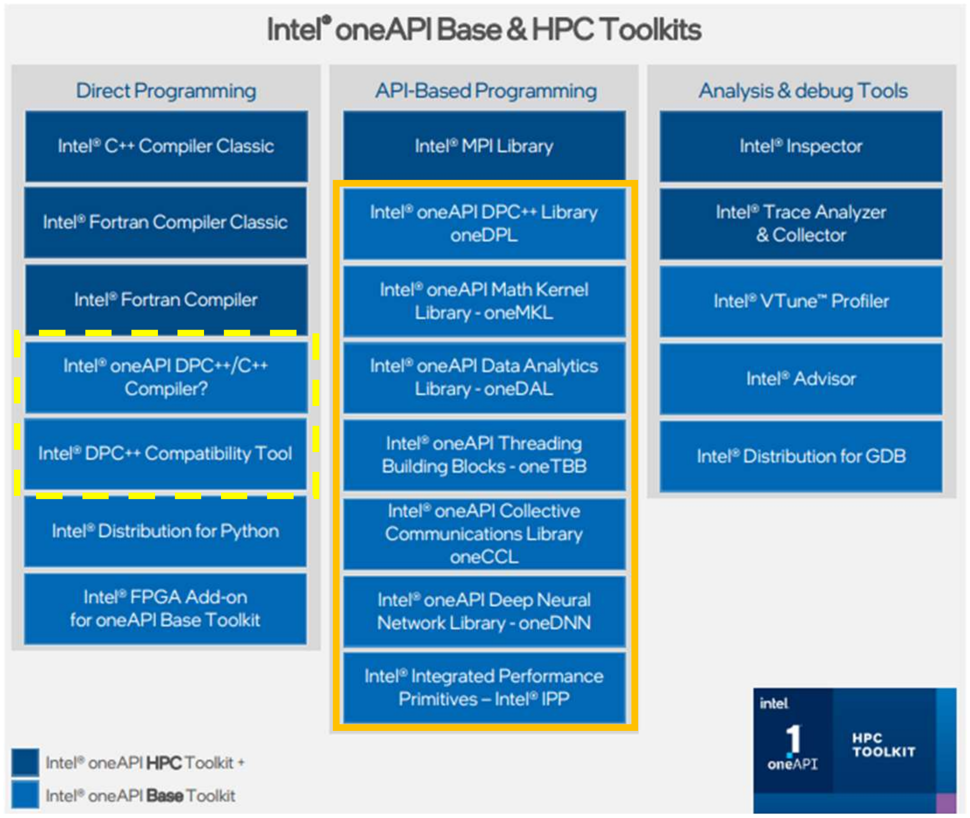
CUDA API Migration Support Status

- CUB API
- cuBLAS API
- cuDNN API
- cuFFT API
- cuRAND API
- cuSOLVER API
- cuSPARSE API
- NCCL API
- nvGRAPH API
- nvJPEG API
- NVML API
- Runtime and Driver API
- thrust API

ONEAPI API-BASED PROGRAMMING OVERVIEW

Overview of oneAPI: Direct vs API-based Programming

oneAPI Libraries = foundational SYCL-based API libraries



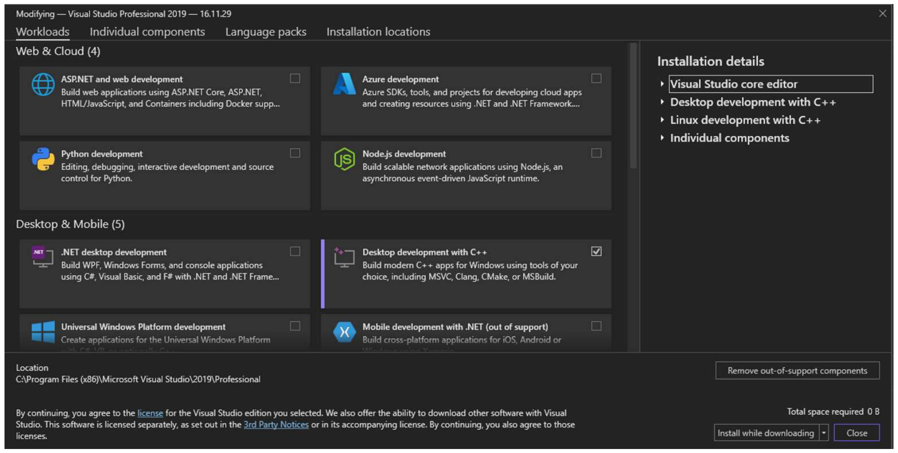
Intel® oneAPI Base & HPC Toolkits
Direct Programming
Intel® C++ Compiler Classic
Intel® Fortran Compiler Classic
Intel® Fortran Compiler
Intel® oneAPI DPC++/C++ Compiler?
Intel® DPC++ Compatibility Tool
Intel® Distribution for Python
Intel® FPGA Add-on for oneAPI Base Toolkit
API-Based Programming
Intel® MPI Library
Intel® oneAPI DPC++ Library oneDPL
Intel® oneAPI Math Kernel Library - oneMKL
Intel® oneAPI Data Analytics Library - oneDAL
Intel® oneAPI Threading Building Blocks - oneTBB
Intel® oneAPI Collective Communications Library oneCCL
Intel® oneAPI Deep Neural Network Library - oneDNN
Intel® Integrated Performance Primitives – Intel® IPP
Analysis & debug Tools
Intel® Inspector
Intel® Trace Analyzer & Collector
Intel® VTune™ Profiler
Intel® Advisor
Intel® Distribution for GDB
Intel® oneAPI HPC Toolkit +
Intel® oneAPI Base Toolkit
intel 1 oneAPI HPC TOOLKIT

```
#include <oneapi/dpl/algorithm>
#include <oneapi/dpl/execution>

#include <sycl/sycl.hpp>

using namespace sycl;
using namespace std;

int main() {
    int hostData[512][1024];

    auto lambda_func = [](int &data) {
        // Computation on the HostData

    };

    // Other compute logics

    // Maximize parallel execution on SYCL devices
    auto policy = oneapi::dpl::execution::dpcpp_default;

    // Use block to make sure data destruction is done after all
    // parallellized tasks complete
    {
       buffer<int> b(hostData, 512 * 1024);

       auto b_begin = oneapi::dpl::begin(b);
       auto b_end = oneapi::dpl::end(b);

       // Parallelize the computation
       std::for_each(policy, b_begin, b_end, lambda_func);
    }

    // Print the result

    return 0;
}
```

SYCLOMATICS ENVIRONMENT SETUP

Setup: Visual Studio

1) Install Visual Studio 2019.
   - Select “Desktop Development with C++” and “Linux development with C++” in the workload section.
2) Install Visual Studio 2019 license key.



Modifying — Visual Studio Professional 2019 — 16.11.29

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- **ASP.NET and web development** ☐ Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- **Azure development** ☐ Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- **Python development** ☐ Editing, debugging, interactive development and source control for Python.
- **Node.js development** ☐ Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- **.NET desktop development** ☐ Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- **Desktop development with C++** ☑ Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- **Universal Windows Platform development** ☐ Create applications for the Universal Windows Platform
- **Mobile development with .NET (out of support)** ☐ Build cross-platform applications for iOS, Android or

Installation details
- Visual Studio core editor
- Desktop development with C++
- Linux development with C++
- Individual components

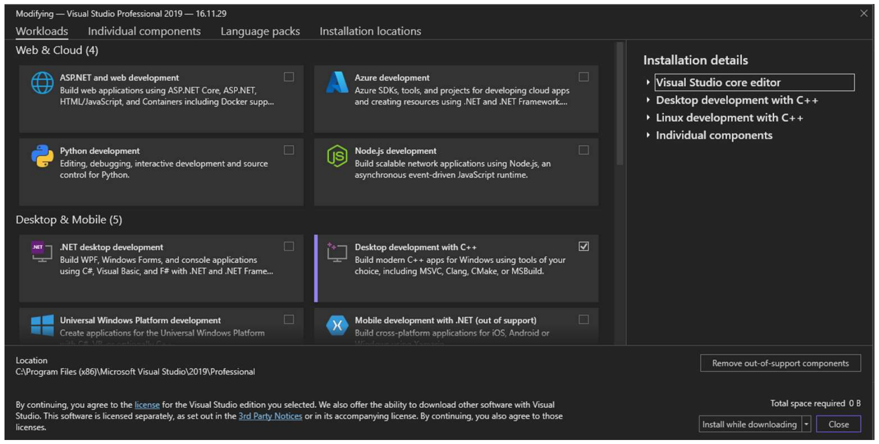
Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional

Remove out-of-support components

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

Install while downloading | Close



Modifying — Visual Studio Professional 2019 — 16.11.29

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- **ASP.NET and web development** ☐ Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- **Azure development** ☐ Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- **Python development** ☐ Editing, debugging, interactive development and source control for Python.
- **Node.js development** ☐ Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- **.NET desktop development** ☐ Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- **Desktop development with C++** ☑ Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- **Universal Windows Platform development** ☐ Create applications for the Universal Windows Platform
- **Mobile development with .NET (out of support)** ☐ Build cross-platform applications for iOS, Android or

Installation details
- Visual Studio core editor
- Desktop development with C++
- Linux development with C++
- Individual components

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional

Remove out-of-support components

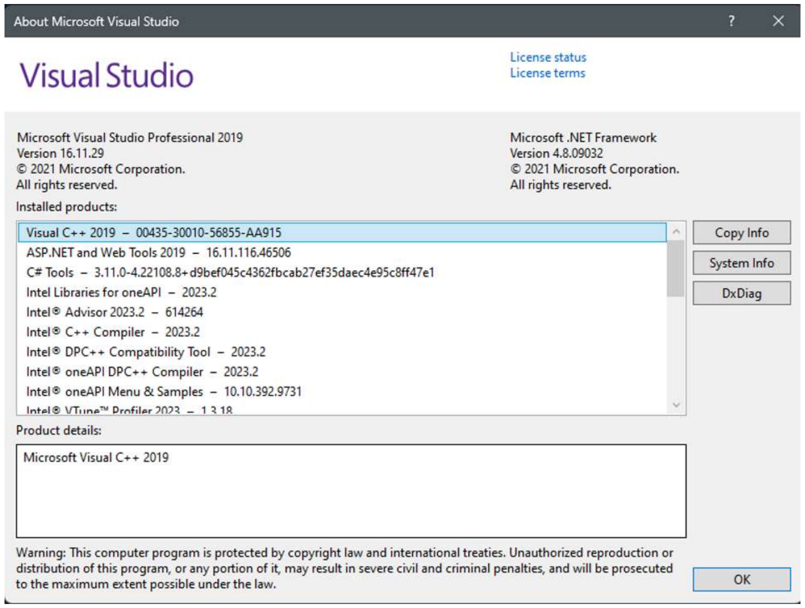
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

Install while downloading | Close

Setup: Intel oneAPI Base Toolkit

1) From https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit-download.html, select version 2023.02.
2) After oneAPI Base Toolkit installation completes, check that the integration of toolkit with Visual Studio.



Microsoft Visual Studio Professional 2019
Version 16.11.29
© 2021 Microsoft Corporation.
All rights reserved.

Microsoft .NET Framework
Version 4.8.09032
© 2021 Microsoft Corporation.
All rights reserved.

Installed products:

- Visual C++ 2019 – 00435-30010-56855-AA915
- ASP.NET and Web Tools 2019 – 16.11.116.46506
- C# Tools – 3.11.0-4.22108.8+d9bef045c4362fbcab27ef35daec4e95c8ff47e1
- Intel Libraries for oneAPI – 2023.2
- Intel® Advisor 2023.2 – 614264
- Intel® C++ Compiler – 2023.2
- Intel® DPC++ Compatibility Tool – 2023.2
- Intel® oneAPI DPC++ Compiler – 2023.2
- Intel® oneAPI Menu & Samples – 10.10.392.9731
- Intel® VTune™ Profiler 2023 – 1.3.18

Product details:

Microsoft Visual C++ 2019

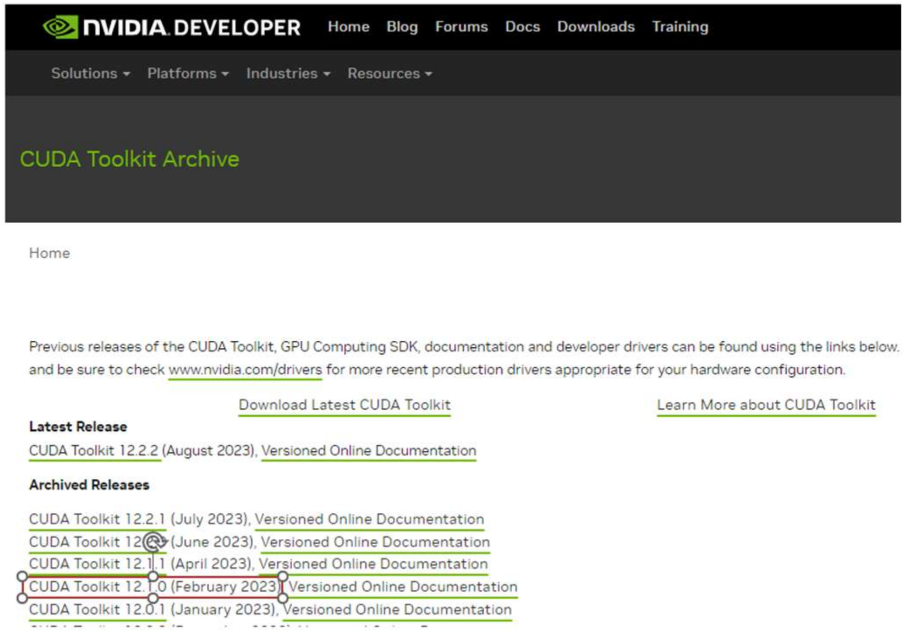
Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Setup: Intel Graphic Driver

1) From https://www.intel.com/content/www/us/en/support/intel-driver-support-assistant.html, install Intel® Driver & Support Assistant (DSA).
2) Follow the step and upgrade the latest Graphic Driver for Windows system.

Setup: CUDA Toolkit

1) For 2023.03 release of oneAPI Base Toolkit, the supported CUDA® version is v12.1.
2) Download and install CUDA v12.1 from https://developer.nvidia.com/cuda-toolkit-archive



CUDA Toolkit Archive

Home

Previous releases of the CUDA Toolkit, GPU Computing SDK, documentation and developer drivers can be found using the links below. and be sure to check www.nvidia.com/drivers for more recent production drivers appropriate for your hardware configuration.

Download Latest CUDA Toolkit   Learn More about CUDA Toolkit

Latest Release
CUDA Toolkit 12.2.2 (August 2023), Versioned Online Documentation

Archived Releases

CUDA Toolkit 12.2.1 (July 2023), Versioned Online Documentation
CUDA Toolkit 12.2.0 (June 2023), Versioned Online Documentation
CUDA Toolkit 12.1.1 (April 2023), Versioned Online Documentation
CUDA Toolkit 12.1.0 (February 2023), Versioned Online Documentation
CUDA Toolkit 12.0.1 (January 2023), Versioned Online Documentation

Setup: Git

1) From https://git-scm.com/download/win, install “64-bit Git for Windows”.
2) After git installation is done, start command prompt and configure its network proxy setting

```
$ git config --global http.proxy http://proxy.png.intel.com:911

$ git config --global https.proxy http://proxy.png.intel.com:911
```

SYCLOMATICS DEMO

Workshop: sortingNetwork – 1/5

1) Start Windows PowerShell and initialize it with intel oneAPI environment

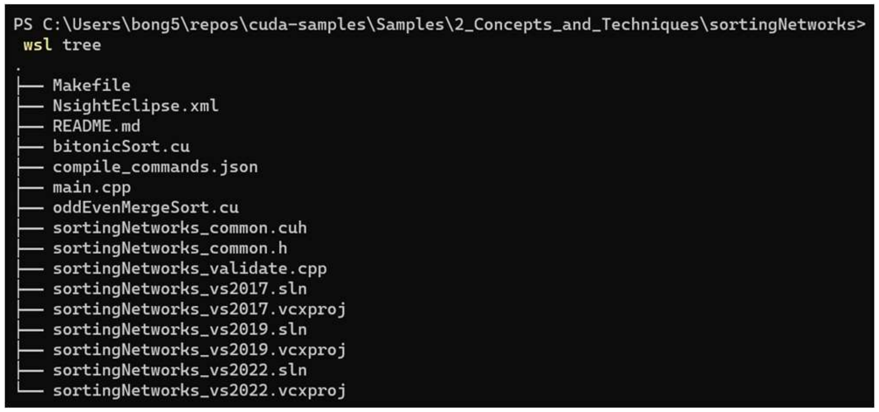
```
PS > cmd.exe "/K"'"C:\Program Files (x86)\Intel\oneAPI\setvars.bat" && powershell’
```

2) Git clone CUDA samples

```
PS > cd C:\Users\<user>\repos
PS > git clone https://github.com/NVIDIA/cuda-samples.git
```

3) Go to sortingNetworks samples

```
PS > cd cuda-samples\Samples\2_Concepts_and_Techniques\sortingNetworks
```



```
PS C:\Users\bong5\repos\cuda-samples\Samples\2_Concepts_and_Techniques\sortingNetworks>
 wsl tree
.
├── Makefile
├── NsightEclipse.xml
├── README.md
├── bitonicSort.cu
├── compile_commands.json
├── main.cpp
├── oddEvenMergeSort.cu
├── sortingNetworks_common.cuh
├── sortingNetworks_common.h
├── sortingNetworks_validate.cpp
├── sortingNetworks_vs2017.sln
├── sortingNetworks_vs2017.vcxproj
├── sortingNetworks_vs2019.sln
├── sortingNetworks_vs2019.vcxproj
├── sortingNetworks_vs2022.sln
└── sortingNetworks_vs2022.vcxproj
```

Workshop: sortingNetwork – 2/5

4) Convert CUDA to SYCL

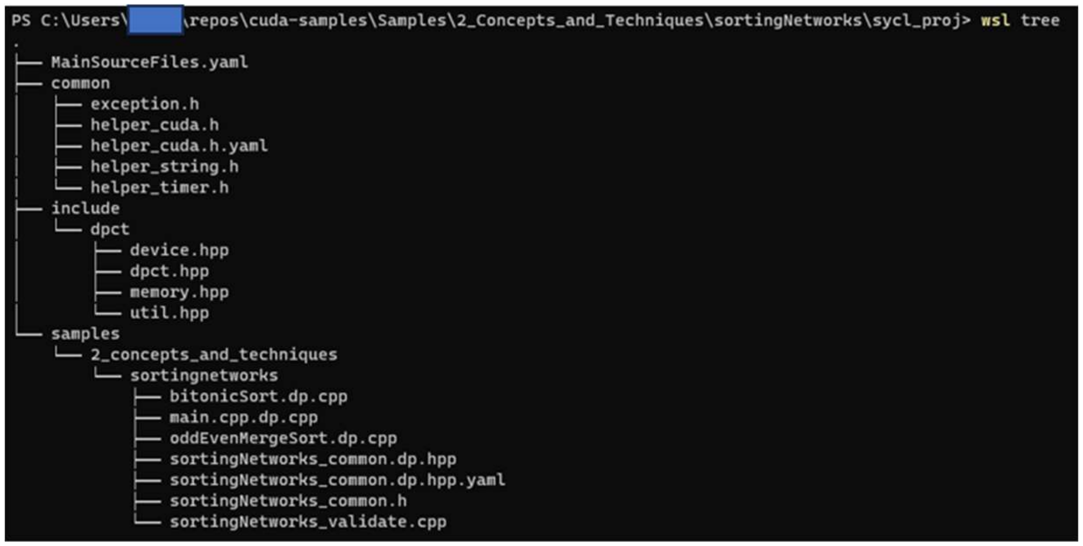
```
PS > dpct.exe --vcxprojfile=sortingNetworks_vs2019.vcxproj --in-root=..\..\.. --out-
        root=sycl_proj --gen-helper-function
```

Note:

- dpct.exe and c2s.exe are the same program (C2S renaming to DPCT).
- --in-root=..\..\.. :- points to the top directory of CUDA sample project.
- --gen-helper-function : Generate SYCLomatic helper header files to output
- --out-root=sycl_proj :- specify the output directory of the conversion

Workshop: sortingNetwork – 3/5

5) The converted SYCL source code:



```
PS C:\Users\[redacted]\repos\cuda-samples\Samples\2_Concepts_and_Techniques\sortingNetworks\sycl_proj> wsl tree
.
├── MainSourceFiles.yaml
├── common
│   ├── exception.h
│   ├── helper_cuda.h
│   ├── helper_cuda.h.yaml
│   ├── helper_string.h
│   └── helper_timer.h
├── include
│   └── dpct
│       ├── device.hpp
│       ├── dpct.hpp
│       ├── memory.hpp
│       └── util.hpp
└── samples
    └── 2_concepts_and_techniques
        └── sortingnetworks
            ├── bitonicSort.dp.cpp
            ├── main.cpp.dp.cpp
            ├── oddEvenMergeSort.dp.cpp
            ├── sortingNetworks_common.dp.hpp
            ├── sortingNetworks_common.dp.hpp.yaml
            ├── sortingNetworks_common.h
            └── sortingNetworks_validate.cpp
```

Note:
- MainSourceFiles.yaml :– CUDA to SYCL conversion log
- common/ :- required CUDA Header files
- include/ :- SYCL helper function
- samples/ :- The SYCL converted source and header files

Workshop: sortingNetwork – 4/5

6) Let's increase the computation iteration in
`sycl_proj\samples\2_concepts_and_techniques\sortingnetworks\main.cpp.dp.cpp`

```
int main(int argc, char **argv) try {
  <removed code chunk>

  const uint N = 1048576;
  const uint DIR = 0;
  const uint numValues = 65536;
  const uint numIterations = 1000;
```

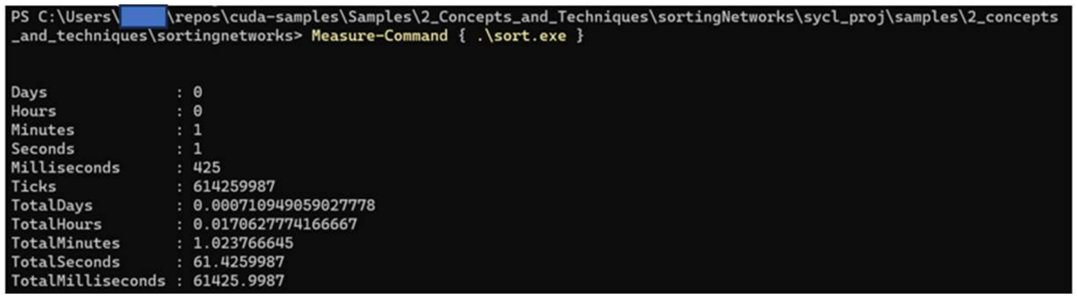
7) Let's compile the SYCL code:

```
PS > cd sycl_proj\samples\2_concepts_and_techniques\sortingnetworks
PS > icpx -fsycl -I ../../../Common -I ../../../include *.cpp -o sort.exe
```

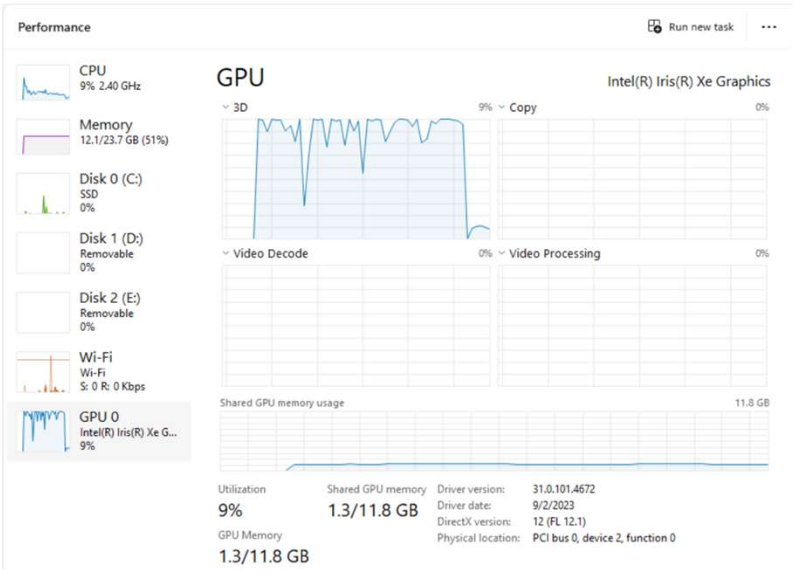
Workshop: sortingNetwork – 5/5

8) Start **Task Manager** and switch to **Performance** view.
9) Execute the SYCL program:

```
PS > Measure-Command { .\sort.exe }
```



```
PS C:\Users\[redacted]\repos\cuda-samples\Samples\2_Concepts_and_Techniques\sortingNetworks\sycl_proj\samples\2_concepts
_and_techniques\sortingnetworks> Measure-Command { .\sort.exe }


Days              : 0
Hours             : 0
Minutes           : 1
Seconds           : 1
Milliseconds      : 425
Ticks             : 614259987
TotalDays         : 0.000710949059027778
TotalHours        : 0.0170627774166667
TotalMinutes      : 1.023766645
TotalSeconds      : 61.4259987
TotalMilliseconds : 61425.9987
```



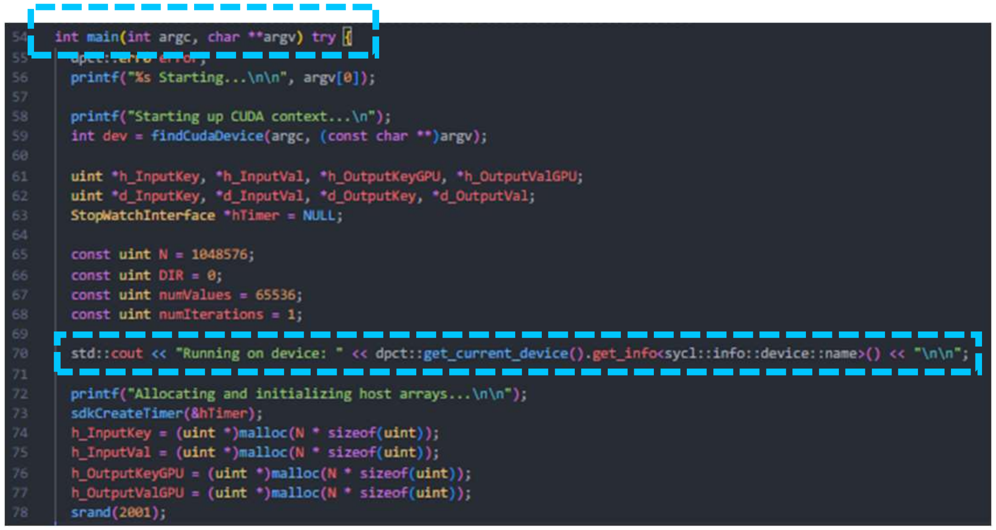
Utilization
9%
GPU Memory
1.3/11.8 GB
Shared GPU memory
1.3/11.8 GB

Driver version: 31.0.101.4672
Driver date: 9/2/2023
DirectX version: 12 (FL 12.1)
Physical location: PCI bus 0, device 2, function 0

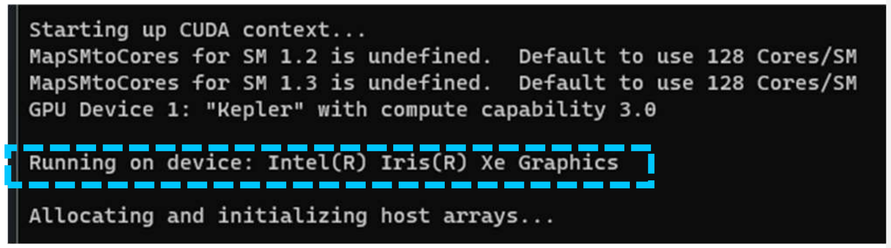
Workshop: sortingNetwork – Tips

- You may also add following code into main.cpp.dp.cpp:

```
std::cout << "Running on device: " <<
             dpct::get_current_device().get_info<sycl::
             info::device::name>() << "\n\n";
```



```
int main(int argc, char **argv) try {
  dpct::err0 error;
  printf("%s Starting...\n\n", argv[0]);

  printf("Starting up CUDA context...\n");
  int dev = findCudaDevice(argc, (const char **)argv);

  uint *h_InputKey, *h_InputVal, *h_OutputKeyGPU, *h_OutputValGPU;
  uint *d_InputKey, *d_InputVal, *d_OutputKey, *d_OutputVal;
  StopWatchInterface *hTimer = NULL;

  const uint N = 1048576;
  const uint DIR = 0;
  const uint numValues = 65536;
  const uint numIterations = 1;

  std::cout << "Running on device: " << dpct::get_current_device().get_info<sycl::info::device::name>() << "\n\n";

  printf("Allocating and initializing host arrays...\n\n");
  sdkCreateTimer(&hTimer);
  h_InputKey = (uint *)malloc(N * sizeof(uint));
  h_InputVal = (uint *)malloc(N * sizeof(uint));
  h_OutputKeyGPU = (uint *)malloc(N * sizeof(uint));
  h_OutputValGPU = (uint *)malloc(N * sizeof(uint));
  srand(2001);
```



```
Starting up CUDA context...
MapSMtoCores for SM 1.2 is undefined.  Default to use 128 Cores/SM
MapSMtoCores for SM 1.3 is undefined.  Default to use 128 Cores/SM
GPU Device 1: "Kepler" with compute capability 3.0

Running on device: Intel(R) Iris(R) Xe Graphics

Allocating and initializing host arrays...
```

Useful link

- https://www.intel.com/content/www/us/en/docs/dpcpp-compatibility-tool/developer-guide-reference/2023-2/dpct-namespace-usage-guide.html
- https://www.intel.com/content/www/us/en/docs/dpcpp-compatibility-tool/developer-guide-reference/2023-2/cuda-to-sycl-term-mapping-quick-reference.html
- https://www.oneapi.io/blog/sycl-performance-for-nvidia-and-amd-gpus-matches-native-system-language/