**oneAPI** programming model 🡪provides **comprehensive & unified portfolio** of **developer tools** used that can be used across hardware, including a ***range*** of **performance libraries** spanning several workload domains.

**Libraries** include ***functions custom-coded*** for ***each target architecture***, so the ***same function call*** delivers ***optimized performance*** **across supported architectures**. **DPC++ is based on industry standards** and **open specifications** to encourage ecosystem *collaboration* and *innovation*.

data centric space = Space focusing on Data Improvement?

Data-centric computing = describes **information system** where data is **stored independently of the applications**, which can be upgraded **without** costly and complicated data migration

Data-centric hardware = Hardware focused on getting data?

Graphical user interface

Description automatically generatedoneAPI = **solution** to ***deliver unified programming*** ***model* to simplify development across diverse architectures**. It includes a **unified** and **simplified language** and **libraries** for **expressing parallelism** and **delivers uncompromised native high-level language performance** across a range of hardware including ***CPUs, GPUs, FPGAs***. oneAPI initiative is based on industry standards and open specifications and is *interoperable* with existing **HPC programming models**.

OneAPI Piece in the Puzzle

command **%%writefile 'simple.cpp'** = Tells **input cell** to **save** ***the contents of the cell*** *into* a **file named** 'simple.cpp' in your current directory (usually your home directory). As you edit the cell and run it in the Jupyter notebook, it will save your changes into that file.

Code used in the example is SYCL code

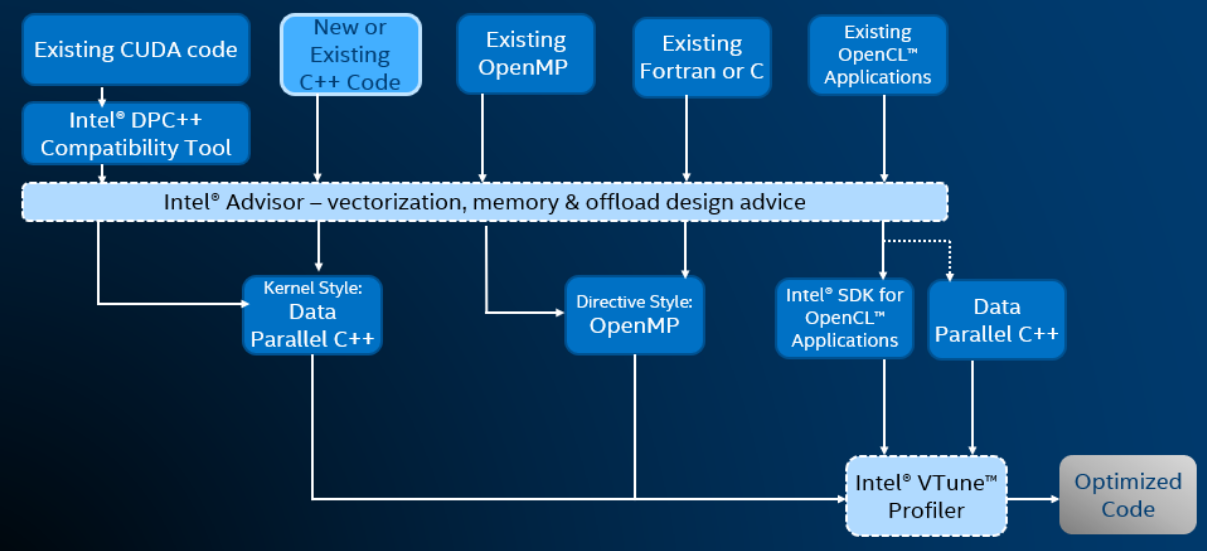
**SYCL** (pronounced ‘sickle’)= **industry standardization effort** that includes **support for data-parallel programming for C++.** It is summarized as “**C++ Single-source Heterogeneous Programming for OpenCL**.” The SYCL standard, like OpenCL\*, is managed by the Khronos Group\*.

**SYCL** is a **cross-platform abstraction** **layer** that **builds on OpenCL**. It **enables code** for **heterogeneous processors** to be **written** in a “**single source**” **style** using **C++.** This is not only useful to the programmers, but it also **gives a compiler the ability to analyze and optimize across the entire program regardless of the device** on which the code is to be run. Unlike OpenCL, **SYCL includes templates and lambda functions** to ***enable higher-level application*** ***software*** to be **cleanly coded with optimized acceleration** **of kernel code**. **Developers program at a higher level than OpenCL** but ***always have access to lower-level code*** through ***seamless integration with OpenCL***, as well as C/C++ libraries.

**Data Parallel C++** (DPC++) is **oneAPI's implementation of SYCL compiler**. It **takes advantage of modern C++ productivity benefits** and *familiar constructs* and ***incorporates the SYCL\* standard for data parallelism and heterogeneous programming***. DPC++ is a **single source language** where ***host code*** and ***heterogeneous accelerator kernels can be mixed in same source files***. A **DPC++ program** is **invoked** **on** the **host computer** and ***offloads the computation to an accelerator***. **Programmers use familiar C++** and library constructs with added functionalities like a **queue for work targeting**, **buffer for data management**, and **parallel\_for for parallelism** **to direct which parts of the computation** and **data should be offloaded**.

DPC++ programs **enhance productivity**. Simple things should be simple to express and lower verbosity and programmer burden. They also **enhance performance** by giving programmers control over program execution and by enabling hardware-specific features. It is a fast-moving open collaboration feeding into the **SYCL\* standard**, and is an **open source** implementation with the goal of upstreaming LLVM and DPC++ extensions to become core **SYCL\***, or **Khronos\*** extensions.

**HPC Single Node Workflow with oneAPI**  
**Accelerated code can be written in either a kernel** (SYCL) or **directive-based style**. Developers can use the Intel® DPC++ Compatibility tool to perform a one-time migration from CUDA to SYCL. **Existing Fortran applications** **can use a directive-based style in OpenMP**. **Existing C++ applications** can **choose either the Kernel style** or **the directive-based style option and** existing **OpenCL applications** can **remain in the OpenCL language or migrate to Data Parallel C++**. **Intel® Advisor** is recommended to **Optimize the design for vectorization and memory** (CPU and GPU) and **Identify loops that are candidates for offload** and **project the performance on target accelerators**. The figure below shows the recommended approach of different starting points for HPC developers:

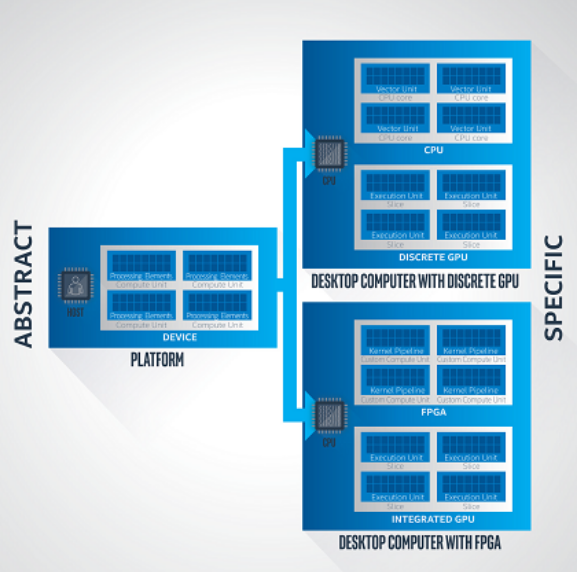


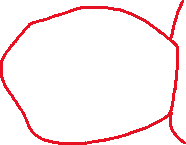
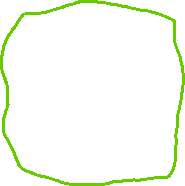
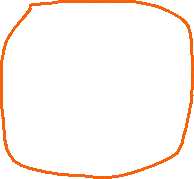
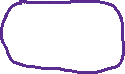
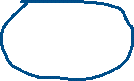
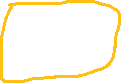
**oneAPI Programming models**

**Platform Model**

Specifies a host (CPU-based system executing the primary portion of a program) controlling one or more devices. The host is specifically the application scope and the command group scope.

**Host** *coordinates and controls* the **compute work** that is **performed on the devices**. **A device is an accelerato**r, a **specialized component containing compute resources** that **can quickly execute a subset of operations** typically **more efficiently than the CPUs in the system**. Each **device contains one or more compute units** that **can execute several operations in parallel**. Each **compute unit contains one** **or more processing elements** that **serve as the individual engine for computation**.

 **One host communicates** with one or more devices. Each device can contain one or more compute units. Each compute unit can contain one or more **processing elements**. In this example, the CPU in a desktop computer is the host and it can also be made available as a device in a platform configuration.



**Execution Model**

Defines and specifies how **code**, **termed kernels**, **execute on the devices** and **interact with the controlling host**. **Host execution model coordinates** **execution and data** **management between** **the host and devices** *via command groups. Command groups (groupings of commands like kernel invocation and accessors) are submitted to queues for execution*.

**Accessors communicate ordering requirements of execution**. The execution model program **declares and instantiates queues**. **Queues** **can execute with an in-order** or **out-of-order** **POLICY** controllable by the program.

**In-order** **execution** is an Intel extension. The **device execution model** *SPECIFIES* ***how computation is accomplished on the accelerator***. **Compute ranging from small one-dimensional data** to **large multidimensional data sets** are **allocated across a hierarchy** of ND-ranges, work-groups, sub-groups (Intel extension), and work-items, which are **all specified when the work is submitted to the command queue**. It is important to note that the actual kernel code represents the work that is executed for one work-item. The ***code outside of the kernel controls*** just ***how much parallelism is executed***; the amount and distribution of the work is controlled by **specification of the sizes of the ND-range and work-group**.

Shape

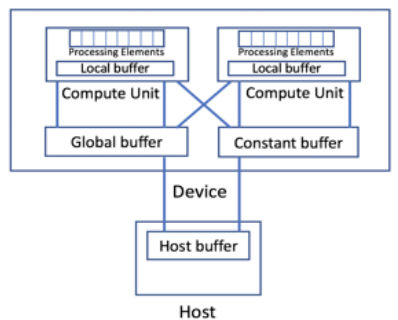
Description automatically generated with low confidenceThe total amount of work is specified by the **ND-range size**. The grouping of the work is specified by the **work-group size**. The example shows the **ND-range size** of X \* Y \* Z, work-group size of X’ \* Y’ \* Z’, and subgroup size of X’. Therefore, there are X \* Y \* Z work-items. There are (X \* Y \* Z) / (X’ \* Y’ \* Z’) work-groups and (X \* Y \* Z) / X’ subgroups.



**Memory Model**

Define how *host and devices* **interact with memory**. It **coordinates the allocation** and **management of memory between** the **host and devices**. This model is an ***abstraction that aims to generalize across*** ***and be adaptable to the different possible host*** and **device configurations**.

**Memory resides upon** and **is owned** by **either** the host or the device and is **specified by declaring a memory object**. There are two different types of memory objects: **buffers and images**. **Interaction** of these memory objects **between the host and device** is **accomplished via an accessor**, which **communicates the desired location of access**, such as host or device, **and the particular mode of access**, such as read or write.

 Consider a case where **memory is allocated** **on** the **host through a traditional malloc call**. Once **the memory is allocated on the host**, a **buffer object is created**, which enables the host allocated memory to be communicated to the device. The **buffer class communicates the type** & **number of items** of **that type to be communicated to the device for computation**. **Once a buffer is created on the host**, the **type of access allowed on the device is communicated** **via an accessor object**, **which specifies the type of access to the buffer**.

**Kernel Programming Model**

**Enables explicit parallelism** between the host and device. The parallelism is explicit in the sense that **the programmer determines what code executes on the host and device**; it is **NOT AUTOMATIC**. **The kernel code executes on the accelerator**. Programs employing the oneAPI programming model **support single source**, meaning **the host code and device code can be in the same source file**. However, **there are differences between the source code accepted** **in** the **host code** and the **device code** **with respect to language** **conformance and language features**. The **SYCL Specification defines** in **detail the required language features for host code and device code**. The following is a summary that is specific to the oneAPI product.



**How to Compile & Run SYCL program**

The three main steps of compiling and running a SYCL program are:

1. Initialize environment variables
2. Compile the SYCL source code
3. Run the application

**Compiling and Running on a Local System:**

If you have installed the Intel® oneAPI Base Toolkit on your local system, you can use the commands below to compile and run a SYCL program:

source /opt/intel/inteloneapi/setvars.sh

dpcpp simple.cpp -o simple

./simple

*Note: run.sh script is a combination of the three steps listec above.*

What is Lambda Function?