Getting Started

Getting Started with ComputeCpp

ComputeCpp™

ComputeCpp Community Edition (CE) is a heterogeneous parallel programming platform that provides a conformant implementation of SYCL™ 1.2.1 Khronos **specification**.

The supported OpenCL 1.2 platforms for ComputeCpp are AMD® and Intel®. Any platform with OpenCL SPIR 1.2, SPIR-V, or PTX support should work. For example any ARM™ device that features an OpenCL platform with the ability to consume SPIR-V.

If you want to try out SYCL and ComputeCpp without installing anything on your computer, you can follow the **interactive SYCL tutorial on the tech.io website**. This allows you to compile and execute SYCL code from your web browser.

Getting started with SYCL™ using ComputeCpp™

If you want to know more about what ComputeCpp, SYCL and OpenCL are, take a look at the ComputeCpp Overview **ComputeCpp Overview section.**

This guide will show you how to set up your environment for ComputeCpp then compile and execute a simple code sample, this should take around 15 minutes. After that you can examine the code and discover how ComputeCpp follows the OpenCL execution model.

For full information on ComputeCpp platform support you can read the **Platform Support Notes**.

Step 1: Download ComputeCpp Community Edition

If you have not already, download the archive containing ComputeCpp and extract it somewhere on your computer. Now clone the ComputeCpp package from GitHub that **contains the sample code here**.

Step 2: Install any pre-requisite tools you do not already have

The pre-requisites for this guide are as follows, if you do not already have these environments installed you'll need to get them now. We don't provide a guide to setting up an OpenCL environment here, but there are plenty on the Internet.

- CMake (version 3.2.2 and above)
- OpenCL 1.2-capable hardware and drivers with SPIR 1.2 or SPIR-V support
- OpenCL ICD Loader
- OpenCL headers
- gcc (version 4.8 and above)

Step 3: Check your hardware support

ComputeCpp includes a tool called computecpp_info. This allows you to check whether the hardware on your target machine is supported by ComputeCpp. It will also tell us if the pre-requisites have been met.

In a Terminal command line change to the "bin" folder in the location you extracted ComputeCpp to and type the command:

```
>./computecpp_info
```

The output of this command will look something like this.

```
*******************************************************************************

ComputeCpp Info (CE 1.0.0)

*******************************************************************************

Toolchain information:

GLIBCXX: 20150426
This version of libstdc++ is supported.

*******************************************************************************


Device Info:

Discovered 1 devices matching:
platform    : <any>
    device type : <any>

        --------------------------------------------------------------------
--------------
        Device 0:

        Device is supported                     : YES - Tested internally
by Codeplay Software Ltd.
        CL_DEVICE_NAME                          : Intel(R) Core(TM)
i7-6600U CPU @ 2.60GHz
        CL_DEVICE_VENDOR                        : Intel(R) Corporation
        CL_DRIVER_VERSION                       : 1.2.0.25
        CL_DEVICE_TYPE                          : CL_DEVICE_TYPE_CPU
*******************************************************************************

```

```

```

This output will show what CPUs or GPUs on your hardware that are supported by ComputeCpp. There is also a guide for the **computecpp_info tool**, and **a Supported Platforms guide**.

Step 4: Set up the ComputeCpp build environment

The sample code for ComputeCpp is built using CMake (minimum v 3.2.2), you can find out more about CMake here **you can find out more about CMake here**.

Most Linux platforms also offer CMake through a packaging system such as wget or apt-get making it easy to install.

The CMake variable

```
ComputeCpp_DIR
```

must be set in order to build with ComputeCpp.

This should be set to the root directory of the ComputeCpp install (i.e the directory with the folders bin, include, lib)

Step 5: Build a ComputeCpp application

We'll now build the simple_vector_add sample code, this application adds two vectors of scalar type on a SYCL device.

First in the terminal change to the root directory of the package you cloned from GitHub. This is the folder containing "samples" and "documents"

```
>mkdir build
```

Now change into that directory

```
>cd build
```

Then call cmake to build all the sample code in the package

```
>cmake ../ -DComputeCpp_DIR=/path/to/computecpp
>make
```

This will build the executable for the sample code.

Step 6: Execute a ComputeCpp application

In the "simple_vector_add" folder simply call the generated file to execute

```
>cd samples/simple_vector_add
```

```
>./simple_vector_add
```

When successful you will see an ouput in the terminal window stating whether the programming executed successfully or not.

Step 7: Modify a ComputeCpp application

This sample application adds two simple vectors together. These vectors are held in arrays within the code.

The application adds A and B together and puts the sum into C, and also adds D and E together and puts them in F.

Change some of the numbers in the arrays, build again using the

```
>make
```

command and then execute again in the same way as in “Step 5.”

If you would like to learn more about how to develop with SYCL try following our **SYCL guide** or you can find out more about how ComputeCpp applications are structured in the the **Anatomy Of A ComputeCpp Application** guide.

The ComputeCpp SDK complements the ComputeCpp Package with build system integration, sample code and documentation and is available on **GitHub**. The **ComputeCpp Integration Guide** offers advice on integrating ComputeCpp with existing applications