OpenKL Specification

Version: 1.0

Document Revision: A

Stillwater/Simunova OpenKL Working Group

Editors: Peter Gottschling, Theodore Omtzigt

Table of Contents

[Acknowledgements 4](#_Toc35973475)

[Introduction 5](#_Toc35973476)

[Glossary 7](#_Toc35973477)

[The OpenKL Architecture 8](#_Toc35973478)

[Device Model 8](#_Toc35973479)

[Memory Model 8](#_Toc35973480)

[Execution Model 8](#_Toc35973481)

[Programming Model 8](#_Toc35973482)

[The OpenKL Device Layer 9](#_Toc35973483)

[The OpenKL runtime 10](#_Toc35973484)

[The OpenKL reproducible numerical compliance 11](#_Toc35973485)

[OpenKL Tensor Algebra 12](#_Toc35973486)

Copyright © 2020 Cloud-Native Supercomputing. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to Stillwater Supercomputing, Inc. and Simunova, AG. This specification, nor any of its component, may not be reproduced, republished, distributed, transmitted, displayed, broadcast, or otherwise exploited in any manner without the express prior written permission of Cloud-Native Supercomputing. You may use this specification to create a software implementation of the functionality defined therein. You may not alter or omit any trademark, copyright, or other notice from the specification. The receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

KPU, Knowledge Processing Unit, Domain Flow Architecture are trademarks of Stillwater Supercomputing, Inc. MTL4 and MTL5 are trademarks of Simunova, AG. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

# Acknowledgements

The OpenKL specification is the result of contributes of many people in the cloud-native and high-performance computing software and hardware communities. Following is a partial list of contributors, including the company or university they represented at the time of their contribution.

Peter Gottschling, Simunova, AG

Theodore Omtzigt, Stillwater Supercomputing, Inc.

# Introduction

Demand for high-performance computing solutions is rapidly expanding with key new applications in cloud, edge, and embedded and autonomous computing. The key attributes of these new architectures are that they are highly distributed, and frequently represent custom application-specific compute engines tightly integrated with sensors or actuators to autonomously act on continuous stimulus. The compute engines may not be continuously connected to remote compute infrastructures, thus requiring a dynamic check-in/report upload/download capability. Furthermore, it is possible that a device might only have the resources to orchestrate a large-scale computation and only consume the results, all the while communicating asynchronously with remote compute or storage. Fundamentally, these autonomous intelligent applications require seamless access to high-performance, specialized application hardware or infrastructure.

The traditional execution models of hardware accelerated applications have been those of tightly coupled collaborations between a small number of processors and accelerators, typically modelled as two address spaces, one for the processors, and one for the hardware accelerators. Collaboration consists of moving data structures between these two address spaces to enable either the processors or the accelerators to modify state. Quintessential examples of this architecture are networking and graphics applications.

Modern distributed systems break this simple model of a collaboration between a processor and an accelerator. Modern systems consist of many additional layers of infrastructure. For example, a typical robotics application will have a locally distributed network of sensors and processors, aggregating into a centralized communication processor that connects to a gateway or base station representing the first layer in a hierarchy to create collective intelligence and value-added services. This layer is typically referred to as the edge computing layer, which reports back up to a geographically consolidated cloud service, which may receive commands from a global control plane that ties these geographic cloud services together into a cohesive global service.

An individual application that runs at each level in the hierarchy might request services from layers below or above it. These services tend to be complex integrations of data and context specific services. For example, connected car services need to run in the 5G base station (edge computing) context to yield sufficiently low latency to create the collective intelligence required for vehicle-to-vehicle information. That context is not required for vehicles that drive around in different cities. As these data sets tend to be large, they cannot move to the device, and thus the device must integrate with the computational infrastructure that contains the data set used to create the value-added service.

OpenKL, the Open Knowledge Language, is an open royalty-free standard for general purpose distributed computing across hardware accelerated application specific devices using cloud-native architecture supporting scalable, remote execution and support for check-pointing and self-healing computation.

OpenKL consists of an API for orchestrating real-time, low latency computation with cloud-native distributed computation. It is defined as a C++ strongly typed interface to facilitate dynamic computation in a standard language environment. No special domain specific languages or specialized compilers are required to write OpenKL high-performance applications. The OpenKL standard:

* Defines a fine-grain parallel execution model in a coarse-grain data flow model
* Uses standard C++
* Defines a high-performance reproducible numerical environment based on posits and quires for user-defined rounding

This document provides a basic overview of the distributed execution environments OpenKL is targeting and defines a list of key requirements for a cloud-native supercomputing execution model. Following the requirements, the architecture of OpenKL, its execution model, its memory model, and its synchronization model are described. The description of the run-time API is followed by a detailed description of the OpenKL C++ programming model. Many examples are given that describe reproducible computation and how they are naturally supported by OpenKL.

The specification is divided into a core specification that any hardware accelerated OpenKL compliant implementation must support; a remote execution profile that supports scalable connected computes, and an asynchronous execution profile that supports program and data migration and digital twins.

# Glossary

**Application:** the combination of the program running on the host, hardware accelerated devices, connected remote compute infrastructures, and digital twins.

**Barrier:** a barrier ensures that all previous commands have finished execution before any following commands can begin execution.

**Command:** an operation representing an atomic amount of work. Commands can represent state manipulation operators such as creating memory objects, or represent large scale fine-grain parallel computation such as solving a system of equations.

**Device:** a device is a stateful execution unit consisting of local memory and a processor fabric capable of executing programs to transform the content of the local memory.

**Handle:** an opaque type that refers to an object, such as a dense vector or a sparse matrix. Any operations on an object occur by reference to its handle.

**Kernel:** a kernel is a fine-grain parallel program that can execute atomically on a device.

**Local Memory:** a memory region associated with a device.

**Object:** an object is an abstract representation of a complex data structure that can be manipulated by the OpenKL API and kernel programs. Examples are a sparse column-oriented matrix of 20-bit posit numbers.

# The OpenKL Architecture

**OpenKL** is an open industry standard for programming an hierarchical distributed machine consisting of application-specific hardware devices, edge, cloud servers, and their proxies.

The target of OpenKL is expert programmers creating next generation applications for high-performance intelligent autonomous embedded devices connected to cloud-native, and IoT infrastructures.

To describe the core architecture of OpenKL, we use the following hierarchy

* Device model
* Memory model
* Execution model
* Programming model

## Device Model

The Device model for OpenKL is defined in figure bla.

## Memory Model

The memory model for OpenKL is shown in figure foo.

## Execution Model

The execution model of OpenKL programs is described in figure gla.

## Programming Model

The programming model of OpenKL is presented in figure goo.

# The OpenKL Device Layer

This section describes the OpenKL layer which implements device-specific features enabling applications to query OpenKL devices, device configuration information, and to create OpenKL contexts to dispatch kernel programs.

# The OpenKL runtime

This section descxribes the OpenKL runtime.

# The OpenKL reproducible numerical compliance

Posits

# OpenKL Tensor Algebra

Scalar, vector, matrix, and tensor abstraction and operators.