Chapter 1 : OpenCL Overview - The Khronos Group Inc

OpenCL Programming Guide. T he OpenGL graphics system is a software interface to graphics tutorial and reference books that help programmers gain a practical.

These kernels are the functions which are to run on the different compute devices. In this post I explain how to get started with OpenCL and how to make a small OpenCL program that will compute the sum of two lists in parallel. Installing and setting up OpenCL on your computer First of all you need to download the newest drivers to your graphics card. Other processor manufacturers such as Intel, also have their own OpenCL implementations. Follow the installation manual of the SDK carefully. You can have multiple OpenCL implementations installed on your system. Suppose we have two lists of numbers, A and B, of equal size. The task of vector addition is to add the elements of A with the elements of B and put the result in the element of a new list called C of the same size. The figure below explains the operation. But since each iteration of this loop is independent on the other iterations this operation is data parallel, meaning that each iteration can be computed simultaneously. So if we have n cores on a processor this operation can be performed in constant time O 1. To make OpenCL perform this operation in parallel we need to make the kernel. The kernel is the function which will run on the compute device. The kernel The kernel is written in the OpenCL language which is a subset of C and has a lot of math and vector functions included. The kernel to perform the vector addition operation is defined below. Below is the code for the host program that executes the kernel above on compute device. I will not go into details on each step as this is supposed to be an introductory article. The main steps of a host program is as follows: Get information about the platform and the devices available on the computer line 42 Select devices to use in execution line 43 Create an OpenCL context line 47 Create a command queue line 50 Create memory buffer objects line Transfer data list A and B to memory buffers on the device line Create program object line 67 Load the kernel source code line and compile it line 71 online exeuction or load the precompiled binary OpenCL program offline execution Create kernel object line 74 Set kernel arguments line Execute the kernel line 84 Read memory objects. This shows how easy OpenCL makes it to run different programs on different compute devices. The source code for this example can be downloaded here.

Chapter 2 : Getting started with OpenCL and GPU Computing â€“ Erik Smistad

Industry Standards for Programming Heterogeneous Platforms OpenCL - Open Computing Language Open, royalty-free standard for portable, parallel programming of.

It was released on November 16, Vulkan and OpenCL 2. Pipe storage is a new device-side type in OpenCL 2. Runs on any OpenCL 2. When releasing OpenCL version 2. An installable client driver ICD must be installed on the platform for every class of vendor for which the runtime would need to support. Each vendor must implement each OpenCL call in their driver. This driver replaces Beignet implementation for supported platforms. This programming framework is built on top of LLVM v5. To unlock the hardware potential, the device runtime uses a push-based task dispatching strategy and the performance of the kernel atomics is improved significantly. This framework has been deployed on the TH-2A system and is readily available to the public [73]. Some of the software will next ported to improve POCL. Nvidia released drivers for OpenCL 1. S3 released their first product supporting native OpenCL 1. Khronos released a WebCL working draft. Khronos announced that the specification for OpenCL 2. Khronos releases the WebCL 1. AMD releases Catalyst Nvidia releases WHQL driver v Nvidia begins evaluation support of OpenCL 2.

Chapter 3 : Introduction to OpenCL

OpenCL (Open Computing Language) is a new framework for writing programs that execute in parallel on different compute devices (such as CPUs and GPUs) from different vendors (AMD, Intel, ATI, Nvidia etc.).

I assume that you are a developer, you know what OpenCL is and you want to get up to speed quickly. You can find the complete source code at Bitbucket. You should fetch the corresponding code to follow along easily. The second part covers how to run a simple kernel, and the third part does a slightly more complicated example where an image is processed. First of all, a quick overview of how OpenCL actually works. OpenCL comes as a runtime environment and has to be installed on your target machine, no matter if you are using Windows or Linux. The runtime installs two things: First, a dispatch library and then the actual runtime containing the implementation. The dispatch library is necessary as there is typically more than one runtime present on a machine. To get started, you need to be able to link against this dispatch library, called OpenCL. It comes with OpenCL 1. It supports OpenCL 1. No matter which one you choose, you have to make sure that the OpenCL. Feel free to grab the one from the repository. If you are not familiar with CMake, then take a look at the CMake tutorial first. The exact path depends on whether you are using Mac OS X or not: Before we get started with actual code, grab a copy of the specification. It contains a reference of all functions, error codes, and structures; making it very handy. In particular, the error codes for each function are well explained and make it easy to understand why a particular call fails. Remember the dispatch library mentioned above which allows choosing between different implementations? An implementation is called a platform in OpenCL; each platform can contain multiple devices. A device is where the code actually gets executed in the end; notice that each device in the same platform can support different features. We begin by querying what platforms are available: After that, we can fetch the platforms in a correctly sized buffer. Not so fast, to actually use it, two more things are needed. A context, which manages resources on a set of devices, and a command queue which executes the commands. This is separated so you can create all resources up-front using only the context, and then create multiple queues on the same device to submit work from multiple threads. Creating a context is straightforward: Every function that can fail in OpenCL returns an error code, either via an output parameter if the function creates an object or directly as the return value. Remember that resources like the context have to be cleaned up later on, by using the appropriate release method. Every resource is reference counted, after the creation, it starts with one reference. At this point, we have a context ready, now we need a queue as well, which is equally easy to create.

Chapter 4 : c# - Complete .NET OpenCL Implementations - Stack Overflow

NVIDIA OpenCL Programming Guide Version programmers to write code that scales with the number of cores, as illustrated in. Figure

Intel , published on June 8, Translating This is a computer translation of the original content. It is provided for general information only and should not be relied upon as complete or accurate. These tutorials work with the supplied sample code to demonstrate important features in this release and can be found on Intel Software Documentation Library repository.. In this tutorial we will show you how to use OpenCL Wizard in Microsoft Visual Studio to create an image processing application for Sobel edge detection of a given image, by creating an OpenCL project based on a project template. Highlights techniques to optimize FFT. We start with a simple Modulate kernel. We proceed with optimizing a Sobel kernel. With OpenCL, a software developer can write a single program running on everything: To reach its full potential, however, OpenCL needs to deliver more than portability. It needs to deliver "performance portability". In this presentation, we discuss the "performance portability" of OpenCL programs. The motion estimation extension includes a set of host-callable functions for frame-based Video Motion Estimation. Advanced Video Motion Estimation Tutorial: The advanced motion estimation extension includes a set of host-callable functions for frame-based Video Motion Estimation. The new feature handles conversion from sRGB into RGB values and speeds up both the development time and the kernel performance. Among new OpenCL 2. These built-ins provide popular parallel primitives that operate at the workgroup level. This article is a short introduction on work-group functions and their usage. The sample code implements an algorithm to demonstrate pointer sharing between host and device with OpenCL SVM features. Advanced topics like use of atomics within SVM allocations and associated performance considerations are out of the scope of this tutorial. Device Self-enqueue in OpenCL 2. Device kernels can enqueue kernels to the same device with no host interaction, enabling flexible work scheduling paradigms and avoiding the need to transfer execution control and data between the device and host, often significantly offloading host processor bottlenecks Device Self-enqueue and Work-Group Scan Functions in OpenCL 2. This tutorial demonstrates these features on our very own GPU-Quicksort implementation in OpenCL, which, as far as we know, is the first known implementation of that algorithm in OpenCL. The tutorial shows an important design pattern of enqueueing kernels of NDRange of size 1 to perform housekeeping and scheduling operations previously reserved for the CPU. Once an image is created, you can use such image features as interpolation and border checking in one kernel, while continuing to access the same physical memory as a regular OpenCL buffer in another kernel. One of the new features of OpenCL 2. Prior to OpenCL 2. This features requires you to set a flag to turn it on, so OpenCL C 1. The tutorial is an interactive image processing Android application. It also demonstrates several simple optimizations, some of optimizations are rather CPU-specific like mapping buffers , while others are more general like using relaxed-math. The sample also shows how to employ the OpenCL profiling events. You can measure performance of OpenCL kernels in many ways. For example, you can perform such measurements using host-side timing mechanisms like QueryPerformanceCounter or rdtsc. We show you some of the methods of measuring OpenCL kernel performance. This tutorial provides an overview of basic methods for resource-sharing and synchronization between these two APIs, supported by performance numbers and recommendations. A few advanced interoperability topics are also introduced, along with references. This sample gives an overview of basic methods for texture sharing and synchronization between the two APIs backed with performance numbers and recommendations. Finally, few advanced interoperability topics are also covered in this document along with some further references. One example use of this is for a real-time computer vision applications where we want to run a feature detector over an image in OpenCL but render the final output to the screen in real time with the detectors clearly marked. In this case you wants access to the expressiveness of the OpenCL C kernel language for compute but the rendering capabilities of the OpenGL API for compatibility with your existing pipeline. Another example might be a dynamically generated procedural texture created in OpenCL used as a texture when rendering a 3D object in the scene. Finally, imagine post processing an image with OpenCL after rendering the scene using the 3D pipeline. This could be useful for color conversions, resampling, or performing compression in some scenarios. The optimization tips, described in this tutorial, are also applicable to any other image processing algorithms targeting the Intel Graphics OpenCL device. Specifically the sample demonstrates how to: The Multi-Device Basic tutorial is an example of utilizing capabilities of a multi-device system. The Multi-Device Basic sample provides an example of three basic scenarios with simultaneous utilization of multiple devices under the same system: In this tutorial we show one important use of API Debugger, and that is to debug and find the root cause of an application failure in the presence of only the kernel source code and the application binary no host side source code. The latest SDK is available at https: This article provides a basic workflow for profiling OpenCL applications on Intel Xeon Phi coprocessors and some examples of performance analysis. For more complete information about compiler optimizations, see our Optimization Notice.

Chapter 5 : OpenCLâ„¢ Tutorials | IntelÂ® Software

Haar Cascade Object Detection Face & Eye - OpenCV with Python for Image and Video Analysis 16 - Duration: sentdex , views.

Chapter 6 : OpenCL - Wikipedia

OpenCL addresses these issues by changing how OpenCL programs are compiled and by introducing SPIR-V. SPIR-V is a low-level representation of an OpenCL program that is designed to be easily loaded by an.

Chapter 7 : OpenCL Programming by Example - O'Reilly Media

data parallel programming model drives the design of OpenCL. Current GPUs can only execute one kernel at a time, so they do not support task parallelism as defined in Section of the OpenCL specification.

Chapter 8 : Getting started with OpenCL, Part #1 | Anteru's blog

An Introduction to the OpenCL Programming Model Jonathan Tompson NYU: Media Research Lab Kristofer Schlachtery NYU: Media Research Lab Abstract This paper presents an overview of the OpenCL standard.