Comparative Performance Analysis of Vulkan Implementations of Computational Applications

Maria Rafaela Gkeka Nikolaos Bellas Christos D. Antonopoulos

Department of Electrical and Computer Engineering
University of Thessaly, Volos, Greece
Email: {margkeka, nbellas, cda}@inf.uth.gr

ABSTRACT

The recent introduction of the Vulkan API and the SPIR-V intermediate-level language by the Khronos Group provides a new GPU programming model in an effort to combine the advantages of its predecessors, OpenGL for 3D graphics and OpenCL for computing. Vulkan's low-level and more direct control over the underlying GPU hardware as well as its support for explicit multi-threaded execution offers opportunities for better performance at the cost of higher programming effort.

Most of the previous work associated with Vulkan has targeted the graphics pipeline. The fact that Vulkan also supports the compute pipeline has motivated us to examine it from the GPGPU perspective, by porting a number of realistic applications to a desktop GPU and evaluating their Vulkan implementations in terms of performance and programmability. Specifically, we consider the *Laplacian filter* which is used in image processing to detect areas of rapid change (edges) in images. Also, we consider a *Visual Odometry* (VO) application used to track the position and pose of a robot by analyzing a sequence of camera frames. VO is part of a *Simultaneous Localization and Mapping (SLAM)* application used in autonomous navigation systems to build a map of surrounding environments and to determine the location of a moving robot inside this map. These applications require advanced pixel-level processing at different levels of pyramid-based granularity, and may even require real-time performance (when, for example, SLAM is used in a robot navigation system). We ported the original implementations (written in C for Laplacian filter and in CUDA for SLAM) to OpenCL, OpenGL and Vulkan and evaluated their performance on a desktop NVIDIA GPGPU.

We show that Vulkan performance is comparable (within 10%) with the performance attained by OpenCL and higher than the performance attained by OpenGL compute shader implementations. By exploiting Vulkan synchronization primitives using the command buffer, we can eliminate the overhead of launching multiple kernel invocations in iterative applications and improve performance of Vulkan implementations by up to 30%. However, the OpenCL compiler seems to be more mature than the SPIR-V compiler used in Vulkan implementations resulting in slightly faster OpenCL kernel execution.

On the other hand, the low-level semantics of Vulkan demand higher programming effort compared with OpenCL/OpenGL which can be a burden if Vulkan is to be used as a GPGPU programming model. Most of the additional effort, however, is boilerplate code that can be reused in more than one Vulkan applications.

Our work is one of the first to consider Vulkan compute as an implementation language for larger scale applications (and not just for small kernels as in previous work).