COMP 426
Assignment 3
Written Report

Shawn Ostad Abbasi 40191688

Gina Cody School of Engineering and Computer Science

Concordia University

Introduction

OpenCL (Open Computing Language) is a framework and standard for the parallel programming of heterogeneous systems through utilizing the CPU, GPU, and other processors that are available. OpenCL is managed by the Khronos Group and was created from the increasing need of a standard amongst many big tech companies like AMD, NVIDIA, Intel, Apple, and others. As stated earlier, OpenCL allows programmers to utilize multiple processors for the execution of their program. The benefit of using different processors is not only being able to improve parallelism but to also benefit from the design of a processing device. In this assignment, we will leverage the GPU's substantially greater number of processing elements and lightweight threads relative to the CPU.

Problem Statement

In this assignment, the computation is not only performed on a different piece of hardware but also implemented differently in the source code compared to previous assignments. Prior iterations of this project performed computations on the CPU through multithreading with C++'s native thread library, as well as multicore programming through the use of the TBB library. Both of these previous iterations performed all the computation on the CPU with its implementation code in .cpp files. On the other hand, the implementation code is in .cl files representing kernel functions with the actual computation being performed on the GPU rather than the CPU through the use of OpenCL. OpenCL accomplishes computation on the GPU and other devices through the use of device queues, work-groups, kernels, and memory buffers.

Considering how the program is heavily data parallel and the significant amount of processing elements on the GPU I believe that the performance will be enhanced with the use of OpenCL compared to the two previous assignments. More specifically, the data parallel parts that will benefit are the rule checking on the grid of species due to the rule checking having to be performed on every cell, as well as the conversion of numerical values to RGB values due to its SIMD (single instruction multiple data) structure.

Procedure

The structure of the program is separated by the host code (.cpp) that is implemented on the CPU, and the kernel code (.cl) implemented by OpenCL on the device. Species exists on a flattened two dimensional array consisting of int8_t/signed char in order to reduce the size of the array and improve locality. The kernel code consists of the rule checking functionality as well as the conversion of numbers to their respective RGB values. In the host code, the arrays of species are initialized, OpenCL and OpenGL are initialized then a series of kernel computations are queued to the device and rendered by OpenGL. In the program the necessary OpenCL components such as the device queue, context, and memory buffers are initialized to enable computation on the device. Initially, I attempted to share the display buffer between OpenCL and OpenGL to avoid copying data between the GPU and CPU, reducing computational overhead. However, I ultimately decided to perform the data copy instead. During my attempts to create a shared buffer between OpenCL and OpenGL, I found that many solutions did not work as intended, and others were platform-dependent, which I chose to avoid. As a result, in my render loop, the program enqueues a kernel to the device queue, performs the computations, reads the data back to the host, and then sends it to OpenGL for rendering.

The program consists of a control thread and computation threads. The control thread in this case is the main thread which initializes OpenCL, OpenGL, queues the kernels, reads the buffer data, and makes the necessary calls to OpenGL to render the data. On the other hand, the computation threads are the hardware threads in the GPU represented as work-items which perform the kernel functions on each of the cells.

Analysis

Performance like prior assignments was measured with frames per second (FPS), where each frame consists of the rule checking through a kernel function and rendering the data through OpenGL. The execution of the program using OpenCL demonstrated FPS ranging from 110-120. With that being said there is a significant improvement in FPS when using OpenCL with the GPU as the computing device in comparison to using multithreading on the CPU or the TBB multicore programming library which demonstrated average FPS of 60 and 80 respectively. The significant improvement is likely due to the GPU’s ability to support a much larger number of threads, made possible by its greater number of processing elements. Although GPUs are designed to handle thousands of threads concurrently and achieve high efficiency through massive parallelism, CPUs, by contrast, experience performance degradation when managing excessive threads due to their limited processing units and scheduling overhead.