COMP 426
Assignment 4
Written Report

Shawn Ostad Abbasi 40191688

Gina Cody School of Engineering and Computer Science

Concordia University

Introduction

In this iteration of the project, the goal was to extend the “Game of Life” program with several new features, including using both the CPU and GPU as OpenCL devices, sharing a buffer between OpenCL and OpenGL to avoid device-to-host data copies, and adding OpenCL pipe functions for kernel communication and ordering. Previous iterations of the project were developed on an Apple M2, whereas this iteration was completed on a Windows machine with a NVIDIA GPU and an Intel CPU. This change was necessary because the macOS system could not expose the CPU device or support OpenCL–OpenGL buffer sharing, likely due to Apple’s deprecation of OpenCL. Moving to a Windows machine resolved some of these issues, although using the CPU as an OpenCL device remained impossible due to lacking permissions to install the required OpenCL CPU runtime on a computer made available by the school.

Problem Statement

In this iteration of the “Game of Life” the GPU performs two tasks, each of which was developed in its own kernel. The first task to be completed is the rule checking for cells/species that exists in our flattened array of species, while the second task is to convert the numerical representation of species into RGB values and later update the shared texture between OpenCL and OpenGL. Due to these two tasks being performed on the GPU, the two kernels implement data-parallelism, leveraging from the massively data-parallel design of GPUs.

Procedure

My program is constructed using both OpenGL for rendering the existing species, and OpenCL for performing parallel computation on the arrays of species. Having initialized both OpenGL and OpenCL, the program calculates the next iteration of species through a series of rule checking on the entire array of species. Having calculated the next iteration of species, the

following is the conversion of species to their corresponding RGB values. This conversion of numerical values to RGB values is made necessary due to how computation and representation of species is strictly numerical. After computing the next iteration of species and determining their RGB values, the program proceeds to render the previously calculated and converted iteration. This process then repeats continuously.

To determine whether each species/cell survives or dies in the next iteration, the program iterates over every cell and applies a series of rule checks. Taking advantage of the GPU's significantly larger number of cores compared to the CPU, the program uses OpenCL to run these checks on the GPU, creating a kernel to perform the rule evaluation in a data-parallel manner. Within the kernel, each work-item obtains its unique ID, which corresponds to a specific cell in the array. The kernel then examines the cell's neighbors, counting neighbors of the same species if the current cell is alive, or counting all neighboring cells if it is dead. Based on this information, the cell may die, remain alive, or be reproduced into a new species, all of which depends on the cell's previous state and its neighbors.

After computing the next iteration of species using the previously described kernel, the program converts the numerical species IDs into RGB values and updates the OpenCL–OpenGL shared texture. To accomplish this without copying data from the device back to the host for OpenGL to use, a second GPU kernel is launched specifically for updating the texture. In this kernel, each work-item processes its corresponding cell by converting its value into an RGB color. Cell values range from –1 to (number of species – 1), where –1 represents a dead cell.

Finally, the host program initially initializes OpenGL and OpenGL, ensuring to create an OpenCL context associated with the already established OpenGL context. After initializing our arrays with a random distribution of cells and performing our original rendering of the cells, the

program repeatedly performs the computation of the following iterations of species followed by rendering. In order to utilize the shared texture, the main rendering loop acquires the OpenGL texture, queues the rule checking and RGB conversion kernels to the GPU, then releases the shared texture. Finally, in order to maintain the order of execution of kernels, an event-based pipeline approach was used to ensure that the RGB conversion kernel only executes after the rule checking kernel finishes.

Analysis

In this project, performance is measured in frames per second (FPS), which indicates how many generations of species can be computed and rendered each second. Although this iteration of the project had to be executed on different hardware than previous versions, the program still achieved strong performance, averaging 342.88 FPS whereas In the prior iteration FPS ranged from 110-120. While the difference in hardware in the last and current iteration makes it difficult to directly compare the results, I still believe there's an improvement that can be attributed to the use of OpenGL–OpenCL texture sharing, which removes the need to copy data from the device to the host.

Portability

To evaluate portability, the program was tested on another computer available through the school. While some systems encountered issues running the program, others executed it without problems. The cause of this inconsistency would require further investigation. As noted earlier, the computer used to develop the program achieved an average FPS of 342.88 using a NVIDIA GeForce RTX 3060 Ti. A second computer, with a NVIDIA Quadro RTX 4000, obtained an average FPS of 281.44. The higher performance of the RTX 3060 Ti is likely due to its significantly larger number of CUDA cores, more than double that of the Quadro RTX 4000.