An Investigation into improving multi-GPU applications with Data compression

Week 9 Report

Sam Serrels -- 40082367

Project Aims

This project aims to research the viability of compressing data on a Graphics Processing Unit(GPU) before sending it to another GPU to reduce the transfer time and bandwidth utilisation.

Project progress

Opencl was chosen as the initial platform for targeting GPU hardware. This was due to Opencl being compatible on AMD and Nvidia hardware, this was important in the initial stages as the capabilities and limitations of each vendor's hardware and driver software were not yet fully investigated.
One of the objectives of the project is to complete research that will be of use for the upcoming new rendering standards (Vulkan, DirectX12). Vulkan is based on a AMD's Mantle, a technology which was loosely based around the OpenGl standard, an additional factor in the choice to use OpenCL .
Vulkan allows for fine-grained control of task execution across multiple GPUs, which has not been possible with existing Graphics APIs. With new capabilities comes new issues to solve, the primary issue being data transfer, which this project aims to optimise.
As these upcoming technologies are not available at the current time, this project is making use of the GPGPU frameworks to emulate potential compute tasks that should reflect real-world rendering applications in the future.
This will also provide useful data for general parallel computing as any optimisation that increase OpenCL performance in an rendering application, could be implemented in other GPGPU projects.

Framework

The first implementation task was to design and build a test framework that could be deployed across many systems with differing hardware. This framework would run tests, gather results and other system data and report back for analysis.
At this current time, the deployment inventory consists of various Windows and Linux machines. The framework code was written to be cross platform, and a development environment was setup to automated deployments and testing whenever code was committed to the version control system.

Current Testing Hardware					
Name	Tesla "GreenBeast" Server	Sam's Lab Pc	Sam's Pc 1	Sam's Pc 2	Cloud Build Server
OS	Linux (CentOS)	Windows 7	Windows 10	Linux (Debian)	Linux (Ubuntu Server)
GPU	2x Tesla K40	2x GTX980	AMD R9 290 2x AMD 5770	2x GTX970	[Intel CPU based opencl only]
Notes	Networked Server, No Graphical output, CMD line only				Used for deployment to other machines, compilation, and static code analysis

OpenCL

It was during this multiplatform development, that issues with OpenCL started to arise. While OpenCL is a pure standard, how the hardware, driver software, and operating system carry out the OpenCL calls is left to the respected manufacturers. Unfortunately "How calls are executed" is a critical part of the investigation of this project.
The OpenCL standard does not define how and when data is transferred from the Host (CPU+Ram) to the Devices(GPUS).
All that is required is that the data is on the correct hardware by the time it is needed (e.g data must be copied to the gpu before the gpu OpenCL kernels can process it) there is no method to explicitly start a data transfer between the host and devices.
Using black box analysis, it can be determined and/or predicted when data is transferred, usually this occurs in an expected manner, e.g immediately after the application defines the data, or just before Kernel execution begins.
This varies with hardware and software, and can be disrupted by other system activities.

Although this project is only targeting data content and it's processing, not how or when it transfers between devices; scheduling transfers in between compression and decompression could be critical to providing a usable case study for this project. This was not a fatal setback for the project, but it has added significant investigation work to the project scope.

Transferring data directly between devices, without passing through Main Memory, was found to be a more complex issue than planned for. OpenCL has no methods or constructs for allowing such transfers. Although the hardware is more than capable of such transfers, as it is used when enabling SLI/Crossfire Dual GPU modes for rendering, the capabilities are not made available for GPGPU tasks.
Extensions for OpenCL to use direct device transfers are provided by AMD, but only for specific hardware (AMD FirePro Compute cards). This means that all transfers between cards had to pass through main memory to some degree, significantly increasing the transfer time. This was not necessarily an issue, as it means that there is a greater demand for potential throughput increase via compression, but it does put further limitations on the application of OpenCL in a multi-gpu realtime rendering scenario.

Cuda

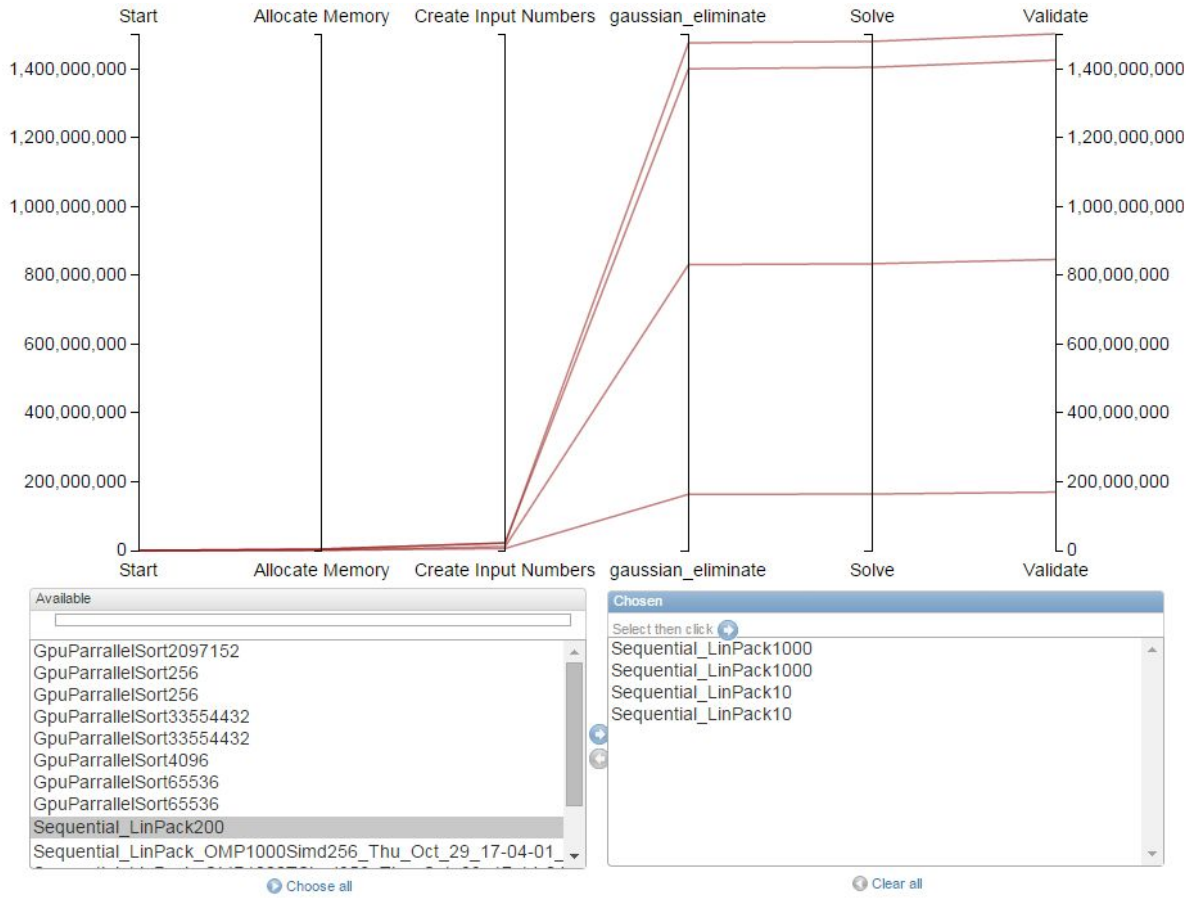
Nvidia's GPGPU framework Cuda, was investigated to see if it could supply any additional features that could be utilised by the project. It was found that it was similar, yet marginally better than Opencl with regards to control over data flow. The Cuda runtime has Intrinsics available to specify the type of data transfers that are to be carried out. An available function is the "cudaMemcpyPeer" which allows for copying directly between devices.
Additionally, on certain hardware (Tesla compute cards) a shared addressing mode can be activated which reduces the overhead of peer to peer transfers.
Cuda is now being used as the primary platform in this project, while it does mean that the software is limited to Nvidia products, the benefit of low level control over data allows for much more detailed evaluations of possible performance increases.

Current Implementation status

- C++ Testing Framework Completed
    - Cross platform, C++11
    - CMake build system, requires/checks for/finds Cuda+Opencl SDK
    - Command line user interface, can be automated with cmdline arguments.
    - timing data recorded and outputted to a standard csv format
- Deployment framework
    - Version control with Git scm
    - Jenkins CI build system deploys to all available test hardware on git commit
    - Jenkins also manages code metrics, project static analysis, and can initiate valgrind testing tools on the application
- Results visualiser
    - Web based visualiser of result csvs.
    - Framework automatically uploads results to the website.
    - Allows Quick and easy viewing and comparison of new and historic result data.
- OpenCL Multi Card sort bitonic sort
    - Each card sort locally, then swaps half of data with other card, sorts again. Swap size halves each iteration, when it reaches 1, data is sorted across both cards.
- Cuda Sort
    - Cuda version of Opencl sort, working, still being tuned.
    - Different data swapping methods currently being investigated

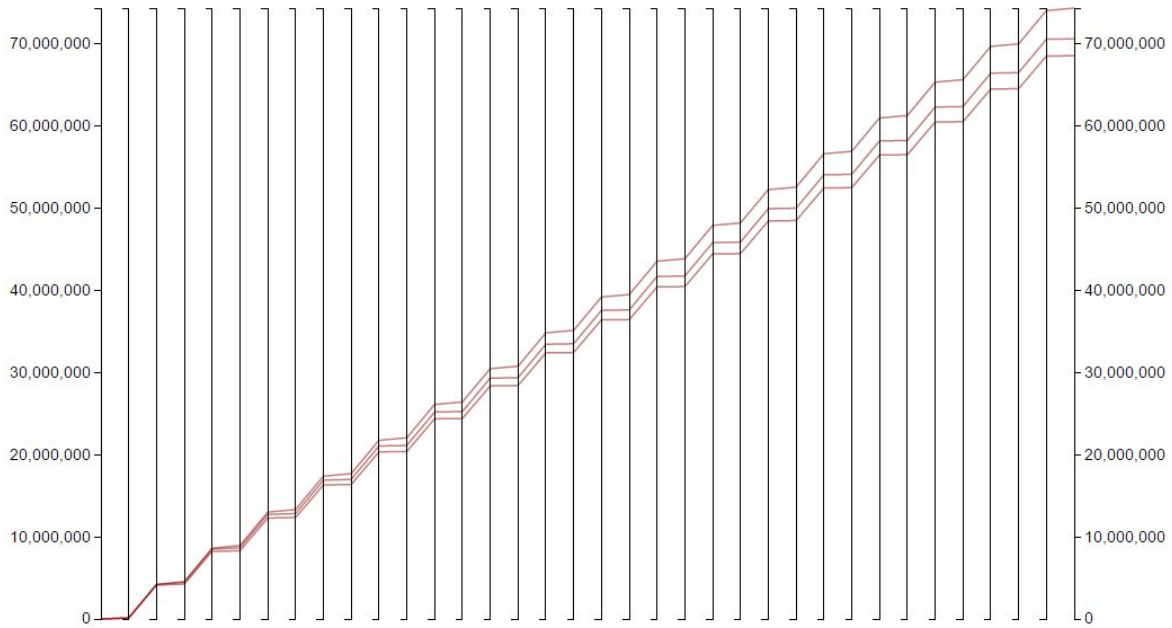
Next Steps

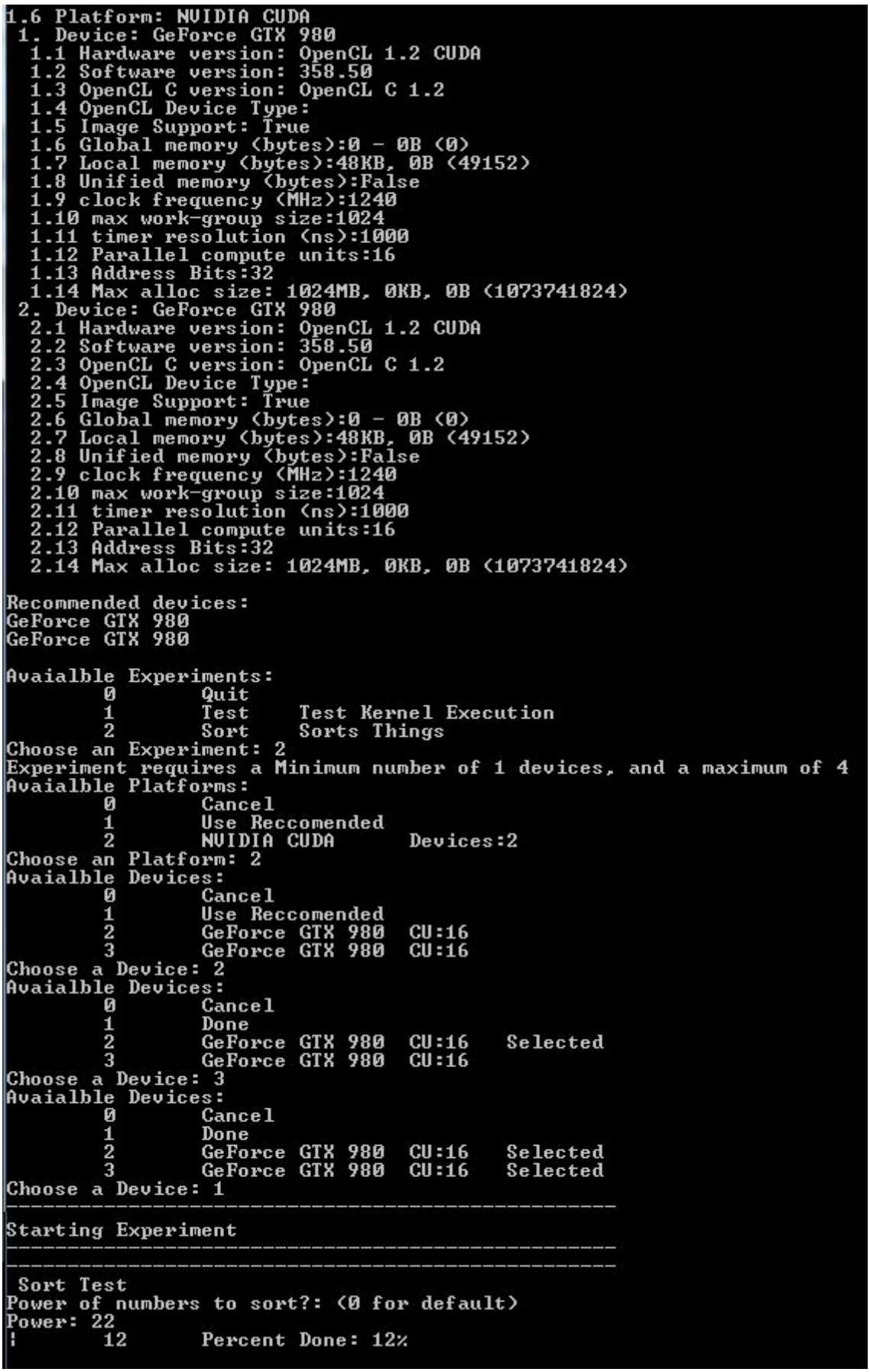
- Investigate and measure data transfer throughput to and from all devices with different methods.
- Implement a Gpu based compression//decompression function
- Investigate and measure viability of compression.
    - Processing time / Compression Amount / Quality (Lossy/Lossless)
- Implement a case study, using real time rendering, multiple cards, and compression.



Results visualizer Tool

Comparison Sort Algorithm with different parameters





Current Framework Implementation.