GIS based terrain analysis with GPU and CPU strategies

Alex Fuerst[1], Charles Kazer[2], and William Hoffman[3].
Faculty Advisor: Arthur Lembo[3]

Introduction

Previous research has shown that the benefits of using GPU based calculations in GIScience for embarrassingly parallel tasks related to terrain modeling yields impressive results. Further, the integration of CUDA into more user-friendly programming languages like Python allow novice programmers to leverage the power of parallel processing for GIS. The widespread adoption of free, open source software for GIS provides a great opportunity to integrate CUDA GIS processing into open source projects.

The goal of this research was to create an open source plug-in, written in Python and PyCUDA, that would enable QGIS to run CUDA driven terrain analysis from the desktop software. The algorithms were written to run in parallel (PyCUDA) and tested against the serial QGIS implementation (C++) in order to evaluate the benefits of the parallel implementation.

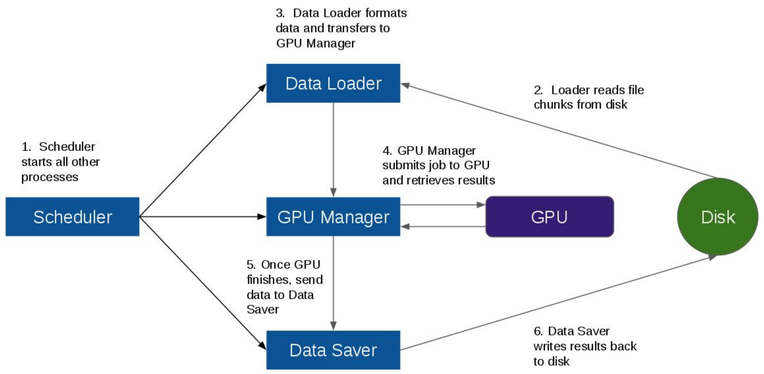

Figure 1: Program model

Methods

Figure 1 shows our design model. There are three main processes, the data loader, the data saver, and GPU manager. Broadly, the data loader is responsible for loading data from disk and formatting from its initial file type, the data saver is responsible for compressing and saving data back to disk, and the GPU manager is responsible for running GPU tasks, including copying data to and back from the GPU. The three processes run independently, so data can be loaded from disk, calculations can be performed on the GPU, and data can be saved to disk all simultaneously. This provides parallelism both on the CPU for I/O, and on the GPU for calculations, improving overall computation time. The encapsulation of the three processes allows for readability and simple modification.

The GPU manager runs the GPU kernel. It can calculate slope, aspect, or hillshade on raster files. The kernel was designed to be easily modifiable so that any calculation on a 3x3 grid around a pixel can be performed -- all of the data management and indexing is handled by the kernel already, and a separate function can be written for any desired algorithm.

[1]Xavier University, [2]Swarthmore College, [3]Salisbury University

Results

Tests were run on PCs running Linux with NVIDIA GTX 670 GPUs with 1344 cores and 2GB of GPU RAM, Intel Xeon E5607 processors with 4 cores running at 2.27 GHz, and 8 GB of main memory reading from and writing to a HDD.

PyCUDA is significantly faster than QGIS when computing both slope and hillshade. The tables below represent the times to compute slope and hillshade on a 1.5 GB raster file. PyCUDA finishes in ⅓ the time of the serial version or better. Looking at just the computation time, the GPU was able to compute hillshade for the 1.5 GB file in under 2 seconds. The rest of the time spent was mostly copying data.

These time gains are less drastic for larger files unfortunately, due to the I/O bottlenecks, though still an improvement over serial. A 12 GB file was completed by QGIS in 45 minutes, whereas PyCUDA took roughly 28 minutes to complete.

Function	PyCUDA	QGIS
Input	1:55	2:00
Computation	1:00	5:00
Output	2:20	2:00
TOTAL Time	**3:35**	**9:00**

Table 1: A comparison of our PyCUDA implementation vs. QGIS calculating slope showed that the CUDA version was slightly faster QGIS.

Function	CUDA	QGIS
Input	1:55	2:00
Computation	1:00	7:00
Output	2:20	2:00
TOTAL Time	**3:35**	**11:00**

Table 2: A comparison of our PyCUDA implementation vs. QGIS calculating hillshade showed that the CUDA version was significantly faster.

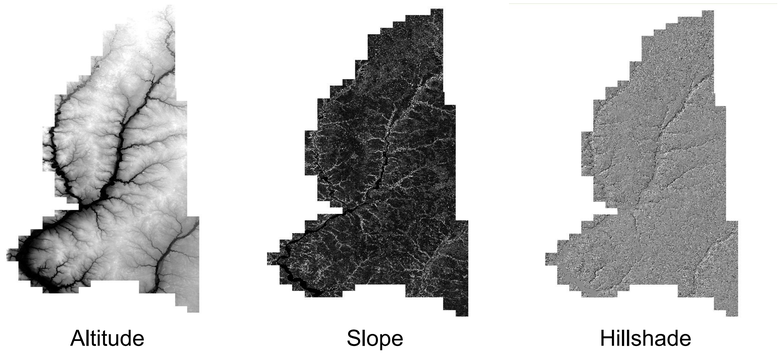

Figure 2: Visualization of Input/Output

Function	PyCUDA	QGIS
Slope	3:01	1:00
Hillshade	2:55	1:00

Table 3: A comparison of our PyCUDA implementation vs. QGIS when running from an SSD. QGIS was 3X faster in this regard.

References

1. S. P. Kirby, W. B. Kostan, and A. J. Lembo, "High performance desktop computing with video gaming cards: Parallelizing raster based functions in gis," 2013.
2. A. Klockner, N. Pinto, B. Catanzaro, Y. Lee, P. Ivanov, and A. Fasih, "Gpu scripting and code generation with pycuda," GPU Computing Gems Jade Edition, pp. 373–385, 2012.

Discussion

Python is a significantly slower language than either C++ or C. A pure Python implementation of slope and hillshade are nearly 10 times slower than C++. Leveraging PyCUDA and Numpy, which are libraries written in C and designed to be heavily optimized, helped reduce the slowdown from Python. However, even with the many optimized libraries we used, and optimizations we built in ourselves, a well tuned python library will always be slower than the equivalent C code.

Once the data starts being processed on the GPU is when significant time differences become apparent. While QGIS slowed down when calculating hillshade, a more computationally expensive algorithm than slope, the PyCUDA implementation ran in the same amount of time, indicating that the GPU parallelization has not taxed the computational power of the card.

The majority of time is spent on disk I/O, reading and writing data off of permanent storage. We were able to reduce this by reading in large amounts of data per call to disk. Using SSDs further mitigates the I/O problem, as they have much better read times than traditional disk drives, though we're unsure why our implementation doesn't benefit from using SSDs.

Conclusions and Considerations

Parallel processing can easily beat serial methods when performing raster analyses. Additional complex functions expand this difference even further. Hillshade requires roughly 45 calculations per element to complete and is no slower than slope which only requires 15. A function that needed hundreds of steps per element would be perfect for adaptation to GPU useage.

Even using just CPU parallelization improves the speed of the calculations. Reducing the I/O bottleneck will yield even greater improvements. This project shows that the introduction of multithreading into GIS applications is very effective.

The code was written under open-source guidelines so that the community can examine and try it out for themselves. We chose Python specifically due to its readability, ease of development, and widespread use in the GIS community. We hope that others in the GIS community can expand on the work we have done to make this code even better and increase the number of raster analyses it can do. It is all available on github at

https://github.com/aFuerst/PyCUDA-Raster.

Acknowledgements

We would like to thank Dr. Lembo for his careful mentoring and useful insight throughout this project. We would also like to thank the NSF for funding our REU research. NSF Award # 1460900