Exploring Methods to Improve the Performance of CUDA Implementation for GIS Terrain Modeling: What Are Reasonable Expectations

David Knipprath, Arcadia University
Dr. Arthur J. Lembo, Jr., Salisbury University

Abstract:

Previous research has shown that with little programming effort, the benefits of using CUDA in GIScience for embarrassingly parallel tasks related to terrain modeling yields impressive results. However, researchers have considered the possibility of further improvements for even faster results.

The goal of this research was to examine how much one can realistically expect to improve the runtimes of GIS raster functions running on CUDA enabled devices compared to both rudimentary CUDA implementations and traditional serial algorithms. Specifically, this research evaluated more complex algorithms utilizing multi-threading, novel tiling schemes, and memory mapped files.

The results of this work showed that while CUDA performs significantly faster than serial algorithms, the more sophisticated algorithm architecture provide only moderate improvements over simpler implementations.

Data and Methods:

Six Idrisi RST raster files (ranging from 512MB to 12GB) were processed and analyzed using a 3x3 kernel function for slope. The function was implemented using both a multi-threaded and serial architecture. Additional methods for improving the algorithm were explored including tiling, and memory mapped files.

The general method for processing a slope function on a raster dataset includes reading in the file, processing the data, and outputting the file. For our multi-threading improvement, we broke the algorithm into three threads, illustrated in Figure 1. Figure 2 illustrates two ways to break a file into tiles, the first methods uses a tile that spans the full length of the file, where as the second uses tiles with a length smaller than that of the file.

Results and Discussion:

The various potential improvements that were implemented showed varying degrees of success.

It was thought that tiling may allow the algorithm to read smaller chunks of data at a time and therefore feed the input buffer faster, and thus get greater parallelism through multithreading. This turned out not to be the case. The file input is such a large part of the total runtime (As documented in figure 3) that the added parallelism hardly makes any difference.

However, we did see marked improvement in runtime of the multi-threaded architecture when compared to a non-threaded version of the same algorithm. A two sample difference (Table 1) test was used to evaluate the introduction of multi-threading in processing Howard County (1.2GB, 336,960,000 pixels). The test showed that multi-threading significantly improved total runtime.

The comparison of fstream methods vs. memory mapped files for file input showed two results. For tiles that spanned the full length of the data set (and were thus read serially) little difference was seen between the two methods. However, with smaller sized tiles mapping was much faster. We believe this is because file mapping allows the OS to handle I/O calls and caching.

Conclusions and Future Considerations:

While there are improvements to gained by multi-threading GIS users would probably see better results by upgrading to solid state drives. Comparing QGIS's runtime to that of the multi-threaded CUDA algorithm, for non-cached files, the speed up isn't even double.

The main issue is the "calculations-to-I/O-calls" ratio. To see truly significant improvements this ratio has to grow. This can happen in one of two ways, either improve the speed of the I/O calls through hardware, or implement functions that require more computations per unit of data.

	Howard Non-threaded	Howard
Mean	22.07	30.82
Variance	16.18	44.45
Observations	42	321
t Stat	-12.08919343	
p	0.000	

Table 1: A comparison of multi-threading vs. a non-threaded implementation showed that the introduction of multi-threading significantly improved the results

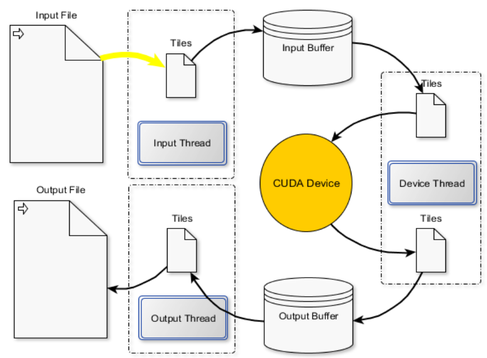

Figure 1. The program architecture for CUDA execution has three primary threads: One thread reads in the file and populates the input buffer, another process the tiles, and the third outputs the results. The input bottle-neck is highlighted.

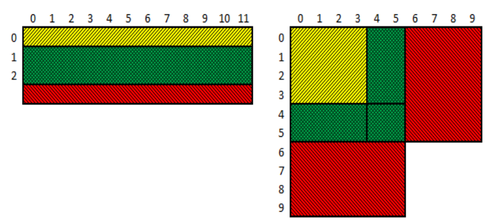

Figure 2. Alternate tiling schemes illustrate how reading can be performed with varying tiling sizes, to facilitate greater efficiency for data access.

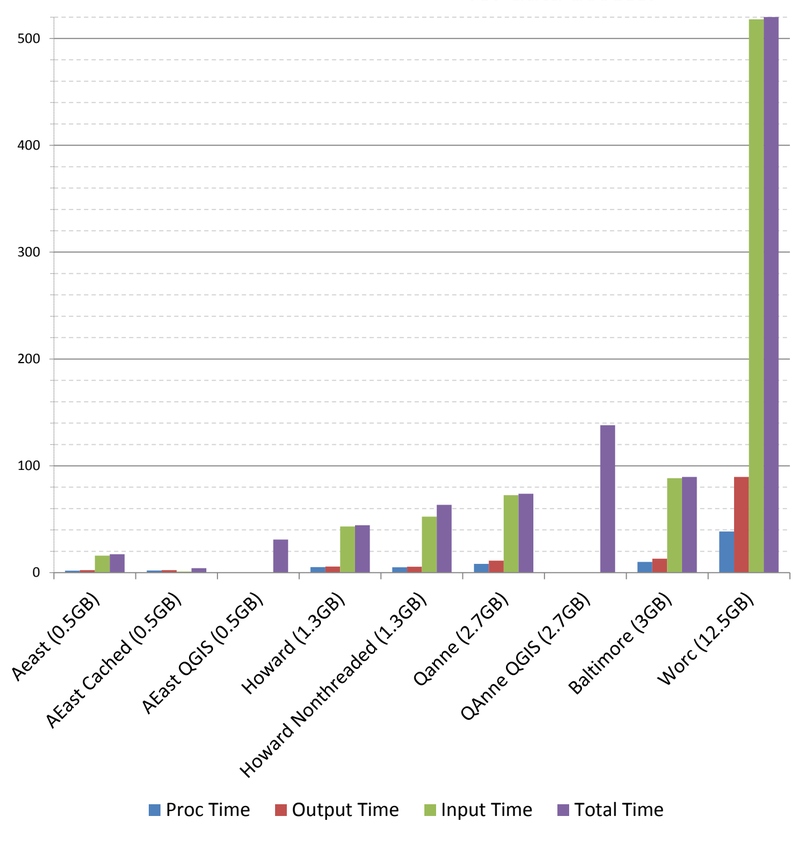

Figure 3:
Analysis of various algorithm architectures. This graph illustrates just how much time is spent reading in the file, compared to processing and output.