Improving Kernel Density Estimation Calculation Time by Parallel Processing.

Adel Kamara, UMES **Keith Benning, Messiah College**
Dr. Arthur J. Lembo, Jr., Salisbury University (faculty mentor)

Abstract:

Kernel Density Estimation is a process in which points plotted onto a chart are assigned a percentage of their value to surrounding cells in the chart, creating a large overlapping of data similar to a large Venn diagram of the points. This allows observers to determine where the points of data cluster the most on a geographic image. Kernel Density Estimation, or KDE, provides far better visual results than a simple plotting of points, but uses up far more computational resources.

This research project was designed to explore the potential speed improvements and processing savings brought on by implementing a KDE system in CUDA, a programming language developed by NVIDIA to run parallel programs on their series of high power graphics processers. In order to correctly test the effect of parallelization, code also had to be written to test KDE data sets both in series and in parallel.

The experiment confirmed that using parallel processing improves run time. However, it also showed that the number of calculations determined by bandwidth used the most processing time in comparison to the number of points or size of the grid.

Methods and Data:

Both serial and CUDA implementations of KDE were implemented using C. Each implementation was evaluated based on the number of points used, the size of the raster grid, and the bandwidth distance used to search for points to include in the analysis.

The programs read a list of x,y coordinates as part of the KDE input, and stores the points in an array for evaluation. The KDE function computes the density of each focus cell based on the number of points contained in the surrounding cells as defined by the bandwidth. As an example, a bandwidth of 1 produces a 3 by 3 square of cells., while a kernel of 2 produces a 5 by 5 square of cells. Each increase in bandwidth dramatically increases the number of computations for the focus cell. The number of calculations for all cells in a grid is illustrated as bellow, where Δ is the bandwidth, W is the density and N is the number of cells, and X_i is the *ith* cell within the bandwidth. X is the focal cell.

$$\mathbf{f\hat{}(x)} = \frac{1}{n\Delta}\sum_{i=1}^{n} w\left(\frac{x - X_i}{\Delta}\right)$$

Upon completion of the computations, the program then writes the final grid into the output file.

Both serial and CUDA implementations were tested with a 1000x1000 grid with point samples of 10000, 50000, 100000, and 200000 to determine the sensitivity of processing a progressively larger number of points. Another test was performed with 50000 points on 3 different grid cell sizes of 100,1000, and 5000 to evaluate the sensitivity of the grid size. Finally, a 5000x5000 cell raster was evaluated with bandwidth sizes of 1 (8 computations per point), 10 (99 computations per point), and 100 (9999 computations per point) to evaluate the sensitivity of the bandwidth size.

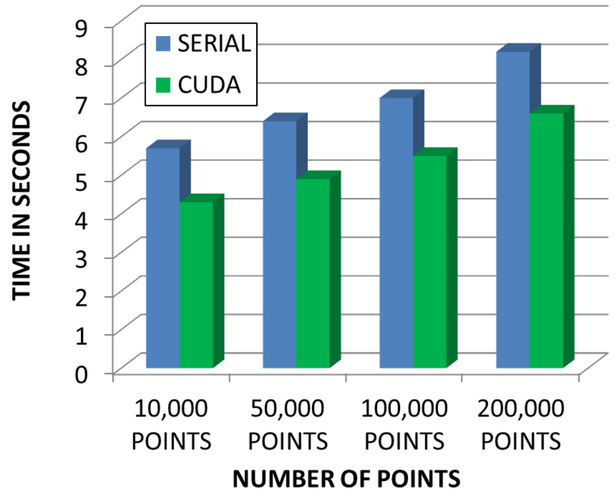

Figure 1: processing time for KDE on a 1000x1000 cell raster shows that CUDA processing was 27% faster for various amounts of points.

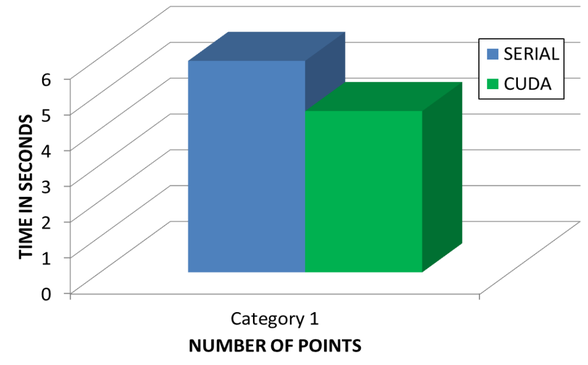

Figure 2: processing time for KDE on a 5000x5000 cell raster shows that CUDA processing was 45% faster for 50,000 points.

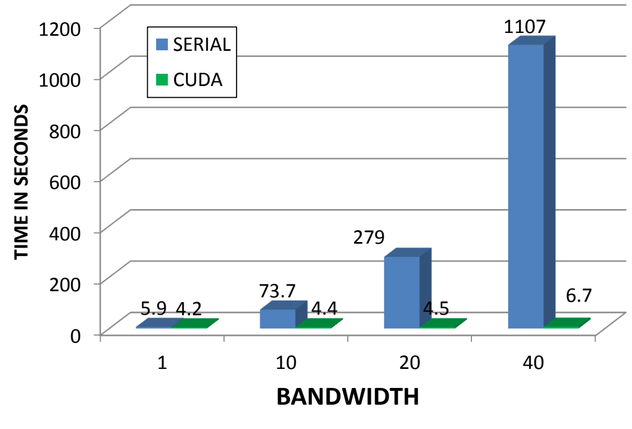

Figure 3: processing time for KDE on a 1000x1000 cell raster with 10,000 points and various bandwidth sizes. CUDA processing was dramatically faster when the bandwidth was increased.

Analysis of Results:

The parallel code performed better than the serial code in terms of processing a larger number of points, and a larger grid. However, the most dramatic difference in processing time came when the bandwidth was increased, tenfold each time. Increasing the bandwidth increases the number of calculations required to process each cell. Serial code can only process one cell at a time, whereas parallel can process multiple cells and their calculations at a time using the large number of threads CUDA is capable of running at the same time.

These results seem to indicate that the greatest performance achievement comes within the context of the massive calculations performed when increasing the bandwidth. As shown in Figures 1 and 2, moderate improvements to KDE processing are achieved using CUDA, primarily because there are relatively few computations per data element. Figure 3 shows the dramatic increase in processing time when the bandwidth is increased, and along with the bandwidth increase, an increase in the number of computations. When tasks can be divided up between multiple processing threads, the improvements in processing time are exponential and somewhat startling.

Conclusion:

Parallelization is a revolutionary way of programming mathematically based tasks, improving processing speed of programs that have problems that can be broken up into individual tasks that can be done at the same time.

Future experimentation might be geared towards improving upon the systems established by this research project, both by improving the efficiency of both sets of code through better programming, and increasing the functionality of the software and its algorithms to obtain finer KDE graphs once the program has finished executing. In particular, improvement on the KDE algorithm for larger data sets would lead to faster, more accurate results than what is currently in this project.

Bibliography:

- "CUDA by Example: An Introduction to General-Purpose GPU Programming", Jason Sanders, Addison-Wesley Professional, 2010
- Hotspot Analysis Based Partial CUDA Acceleration of HMMER 3.0 on GPGPUs. International Journal of Soft Computing and Engineering, 2, 91-95. Retrieved June 9, 2014, from http://www.ijsce.org/attachments/File/v2i4/D0894072412.pdf
- Hotspot mapping using Kernel Density Estimation (KDE). (2005, August 12). Retrieved June 9, 2014, from http://www.cadcorp.com/pdf/BR-Hotspot-Mapping-Using-KDE.pdf
- Kernel Density Estimation. (2014, January 1). . Retrieved August 1, 2014, from https://onlinecourses.science.psu.edu/stat464/node/88