Eric Sheeder
CSCI 440

Homework Assignment 2 – Sparse Matrix Vector Multiplication

For this assignment, we were asked to parallelize an algorithm for multiplying a spare matrix by a column of randomly generated data.

When multiplying dense matrices/vectors together, you will generally access the next element in memory for your next computation, giving it good spatial locality when it comes to memory access. With a sparse matrix, though, you will generally skip over the empty values in the matrix, meaning that the next element you access is typically not next to the element previously accessed, which gives poor memory access efficiency.

To counteract this memory access problem, we instead store the values of our sparse matrix into 3 single-dimension arrays: one for the value, one for the column that the value belongs in, and another array that stores one value for each row, which gives the index number of the non-zero value that starts the row. When accessing the non-zero elements sequentially, they are right next to each other in the value array, giving us better spatial locality.

To do the actual multiplication, my program allocates memory for an array of “products”, which stores the multiplication components of the dot-product that we need to do for matrix-vector multiplication. In other words, products[0] will store the first non-zero value in matrix A multiplied by the row in column B it corresponds to, products[1] will be the second non-zero value multiplied by its corresponding B column, and so on. Once this products array is populated with data, 1 block is allocated for each row in matrix A to compute a single value for the output vector. Each block performs an inclusive segmented scan to sum the components for the row, and then stores this value in the final output vector.

To compile the program, type nvcc -I /usr/local/cuda-samples/common/inc -o matrixMul Eric_Sheeder_SpMV.cu on the command line. To run it, type ./matrixMul <sm1>.txt, where <sm1>.txt is the name of the input text file. The results will be stored in output.txt, which contains the CPU computations, GPU computations, and the difference between them. If the difference is less than 0.00001, then the listed difference is just “0.0” or “-0.0”. This is to make it easier to see at a glance if larger differences are present, meaning there is an error in the calculation somewhere. As far as I know, the two methods give essentially the same answer.