Assignment 2: GPU KNN using CUDA

Steven M. Hernandez
Department of Computer Science
Virginia Commonwealth University
Richmond, VA USA
hernandezsm@vcu.edu

I. INTRODUCTION

CUDA provides programmers an interface to develop high performance parallel code through kernels which are run in parallel on GPUs. Through this method, we explore the speed capabilities of such a highly parallel GPU based system when compared to basic sequential CPU implementations as well as threaded parallel CPU implementations for use in K-Nearest Neighbor (KNN) classification. For all datapoints within this report, we use the dataset contained in the provided *medium.arff* file along with the value for $k = 5$.

II. DISCUSSION

In working with CUDA we find that the algorithm developed for calculating KNN on the CPU does still result in an improvement in Speedup compared to the threaded CPU version with 2048 threads (table I).

TABLE I
TOTAL TIME TAKEN TO CALCULATE KNN

Method	Time (ms)	Accuracy (%)	Speedup
Sequential	8433	49.59	N/A
CPU (1 thread)	8446	49.59	0.998
CPU (2 threads)	4402	49.59	1.916
CPU (4 threads)	2270	49.59	3.175
CPU (8 threads)	1192	49.59	7.075
CPU (16 threads)	695	49.59	12.13
CPU (2048 threads)	351	49.59	24.03
GPU	244	49.59	34.56

A. Profiling

Profiling reveals an extraordinary performance for calculation. Table II shows the times taken for individual elements of the overall CUDA program revealing our core logic taking a very low percentage of the overall time of the program. First, do notices the overall execution time is higher when profiling than specified previously in table I. This can be attributed to the overhead required of the profiler. In addition to this, the table does not reveal all parts of the program, instead simply revealing the most interesting and relevant points of note.

As we can see, table II shows that our first cudaMalloc takes up a huge portion of time when compared to not only the amount of time it takes to run the actual kernel calculation, but also when compared to other cudaMalloc calls this amount of time is staggering. This extra time was not necessarily a result of the large amount of space being allocated; instead this first cudaMalloc only allocated enough space for a single integer element. The fourth cudaMalloc listed in the table was the cudaMalloc which was allocating the greatest amount of space, but again, we can see that the time taken for this allocation is much smaller, we have no reason to look at optimizing the number of actual allocations. Instead, this large time can be chalked up to overhead in first setting up CUDA and communicating with GPUs initially. As a result, there appears to be no way to improve this element of the program.

Because of all this, we have little incentive to optimize the program overall for the given datasets *small.arff* and *medium.arff*. Of course, as we begin reaching much higher number of elements, this is when we would look to these optimizations; when running the kernel begins overshadowing the time taken for initialization of CUDA.

TABLE II
TIME TAKEN FOR SIGNIFICANT PORTIONS OF THE CUDA GPU PROGRAM THROUGH PROFILER (497 MS TAKEN OVERALL)

Step	Time Taken (ms)	Percent of Runtime
cudaMalloc # 1	325	65%
cudaMalloc # 2	0.007	<1%
cudaMalloc # 3	0.004	<1%
cudaMalloc # 4	0.100	<1%
cudaMemcpy (HtoD) All	0.070	<1%
Launch Kernel	4	<1%
Kernel Running	29	5.8%

B. Note on Accuracy

Please do note an error from assignment 1 resulted in incorrect accuracy calculations, thus resulting in different accuracy values between this report and the previous report. The issue was using the class as a dimension in calculation of the nearest neighbors, thus resulting in a slightly higher accuracy values from the previous report. As a result, this was fixed and all values ($Accuracy$, $CPU\ time$ and $Speedup$) were recalculated for table I.

III. CONCLUSION

CUDA provides us a method of developing highly parallel code which can give us performance improvements over standard CPU parallel methods such as threading. In this report it is found the KNN algorithm to be a simple algorithm to implement and run on GPUs through CUDA.