CS406 Parallel Computation

Homework 4 - Parallelization of Nearest Neighbor Search

ORAN CAN ÖREN

December 2017

1 Task Description

The nearest neighbor problem is a variant of the k-nearest neighbors (kNN) problem, where k is substituted for 1. The problem is widely applicable, some of its instances are entropy estimation, clustering, and content-based image retrieval. There are multiple distance calculation methodologies, such as the euclidean distance and the Manhattan distance. Our task requires us to compute distances in terms of euclidean distances. The euclidean distance d, among two vectors $X = (x_1, x_2, ..., x_k)$ and $Y = (y_1, y_2, ..., y_k)$ in k-dimensional vector spaces is defined as follows.

$$d = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

The number of training and test points are provided in advance, specifically there are 19000 points in the train data and 1000 points in the test data.

2 Algorithm

The most straightforward algorithm solving the problem is the brute-force (BF) algorithm. For a given point X in a multidimensional space, BF queries rest of the points to find the nearest point. Various algorithms having better run-time complexities have been proposed earlier; however, BF is highly parallelizable and its GPU implementations have been experimentally shown to be performing faster than its competitor algorithms by significant factors [1].

3 Implementation Details

Following subsections discuss different implementations for the BF algorithm.

3.1 Parallel algorithm v1

The CUDA code for the first version of my implementation can be found in file "source1.cu".

3.1.1 the algorithm

The first algorithm I devised shared the workload of computing the distances for each test point, hence 1000 GPU threads were utilized as there are 1000 test points. For each test point v, the algorithm computed the euclidean distance of v with respect to each each test point $t \in T$. Each thread kept two unsigned integers, one for keeping the index of the nearest point $n \in T$ and another for keeping the distance between v and n to obtain the nearest neighbor of v in a greedy manner.

3.1.2 performance

CUDA kernel that's responsible for computing the nearest neighbors takes about 39 ms, yielding correct results for the provided dataset; however overall execution time of the program is about two seconds. Performance profiling has shown that the bottleneck is memory allocation for host and device pointers. It turns out that the cause for this 1.5 second delay for such a trivial operation is the lack of CUDA context prior to this CUDA function invokation. I've placed a function call for `cudaSetDevice()` to initialize the CUDA context within the process. Subsequently, the overall execution time decreased to 430 ms.

3.2 Parallel algorithm v2

The CUDA code for the second version of my implementation can be found in file "source2.cu".

3.2.1 the algorithm

To take advantage of the asynchronous processing capabilities of CUDA and overlap communication with computation, I've utilized multiple CUDA streams in

this implementation. Overall algorithm is the same as before, each thread is given the task to find the nearest tranining point to the test point it is assigned to. I utilized four CUDA streams in this version and assigned the work of finding nearest points of 250 testing points to each stream. During the time taken for the computation of a stream, the program reads another chunk of test points and copies this data to the GPU.

3.2.2 performance

Contrary to my expectations, this implementation took longer than algorithm v1. Kernel takes about 49 ms. This may be the case because of the small size of input data. The overhead of such parallelization is larger than the benefits of it.

Algorithm 1: Outline for the brute-force algorithm

```
   Input: X: array of training points
   Y: array of testing points
   Output: D: D[i] is the nearest x_j ∈ X to y_i
1  thread local closest_dist = ∞
2  for y_i ∈ Y in parallel do
3  |  for x_i ∈ X do
4  |  |  if distance(y_i, x_i) < closest_dist then
5  |  |  |  closest_dist = distance(y_i, x_i)
6  |  |  end
7  |  end
8  end
```

4 Execution Time Tables

Table 1: My caption

	Algorithm v1	**Algorithm v2**
Kernel Time	39 ms	44 ms
Total Time	406 ms	424 ms

References

[1] arXiv:0804.1448 [cs.CV]

[2] V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud, "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," 2010 IEEE International Conference on Image Processing, 2010.