Current Trends in Technology and Sciences
ISSN : 2275-0535
9th Symposium on Advances in Science & Technology (9thSASTech 2014), Mashad, Iran

Presentation of a Parallel Algorithm for Nearest Neighbor Search on GPU Using CUDA

Neda Mohammadi
Shiraz University of Technology, Computer Engineering and IT Department, Shiraz, Fars, Iran,
E-Mail:N. Mohammadi@sutech. ac. ir

Abstract **– Nearest Neighbor Search Algorithm has many applications in various sciences, for example KNN classification techniques are used often in industry and in many scientific applications. It has been applied in areas such as medical imaging, entropy estimation, data mining, machine learning and content based image retrieval. The high computational complexity in nearest neighbor algorithm is a challenge for runtime. Although presenting and solving of this problem and for small data is easy, when the database is large The fundamental problem in fast processing of data occurs. In areas such as, data mining where the nearest neighbor search algorithm is applied for it, several technologies have been used to classify the data. Several technologies in order to data classification are introduced which with increasing amount of data choosing appropriate technology for classifying them is important. CUDA technology was provided by NVIDIA and also this technology provided an opportunity for developers so that by using of their system graphics card, data in parallel they performed minimal cost and easy computational processing data. The concept of GPGPU and CUDA technology for nearest neighbor search is used. We will compare parallelism implementation of this algorithm on GPU With accessing to it's shared memory with serial implementation of algorithm on CPU and while the program without access to shared memory on a graphics processing unit runs. It is shown that Parallel implementation of the algorithm on GPU with accessing to the shared memory in compared to the other methods discussed here is heavy computational processes in parallel method.**

Keyword **– Nearest Neighbor, CUDA, graphics processing unit, Shared Memory, Parallelization**

1. INTRODUCTION

Nearest Neighbor Search Algorithm is used to find similar items in many issues. For solving such issues, Nearest neighbors of an object should be found In a metric space. It has been applied in areas such as data mining, entropy estimation[1, 2, 3]. in these areas, amount of data for processing constantly Increases. Particularly in data mining[3], Searching for implementations which can respond to large data collections is done. Parallel Processing provided a scalable solution for nearest neighbor search algorithm In the case of high volumes data. graphics processing unit(gpu) contains hundreds parallel processor core that can simultaneously manage thousands of threads. The purpose of the gpu creation is to effectively fulfillment required calculations for 3D graphics that often are simple. cuda provide opportunity for using of gpu's power in parallelization non-graphics computing processing. Cuda technology can be written in c language and implemented on graphic processing units so that in this way, scalable and cost- effectivity solutions provided for parallel implementation of the heavy computational processes. we propose a parallel implementation of the cuda for NN search and evaluate its runtime performance. Algorithms that are implemented by the graphics processing unit dependens on various hardware. All algorithms are implemented on a system by Intel core 2 duo processor 2. 40 GHZ with 3GB memory and windows 7. the graphics card used is a Nvidia Geforce 9300 MG and expected that Algorithm's run on GPU with CUDA causes enhancement in it's speed. The following, section 2, some related work reviewed, Section 3 explains Nearest Neighbor Algorithm in more detail and describes GPU and CUDA. Then describes, how it is implemented in Serial on CPU and how NN Algorithm is implemented in Parallel On GPU without accessing to GPU's Shared memory. Section 4 describes how the NN Algorithm is implemented in parallel with accessing CUDA shared memory. Results and analysis are presented in Section 5. Section 6 concludes the paper.

2. RELATED WORK

There are some pervious researches on improving performance on NN Algorithm[1, 4, 5]. in 2008 a method for improving algorithm performance is presented[4], that used a parallel Algorithm for finding K-Nearest Neighbor in run-time with using CUDA and creates trees with degree of K, but using of this method has low scalable. In[1] an algorithm called 'brute force' for implementation KNN is proposed, the results indicate that implementation of KNN with this method is faster than serial on CPU and also is faster than implementation using kd-tree. In[5] a randomized algorithm called LSH is presented that Neighbors are calculated by estimating. The author has reported that this method reduces the time complexity but There is little probability that algorithm in finding nearest neighbor to fail and Does not guarantee the correct answer. In Here, serial pseudo-code NN algorithm is used in [1] as a starting point and then in order to speed enhancement and better performance,

algorithm in term of implementation complexity is parallelized.

3. NEAREST NEIGHBOR SEARCH ALGORITHM

$P=\{i_1, i_2, i_3, \ldots, i_n\}$ was defined as a set of n points in 3D space, for each query point q as q € P, NN Algorithm find nearest neighbor to this query point of this set of points. This algorithm has great importance in computer science (pattern recognition, search in multimedia, data mining)[5]. for calculating distance between the points, the Euclidean distance is used. Of Course, for calculating distance, any other method can be used. Figure 1 show KNN where K=1.

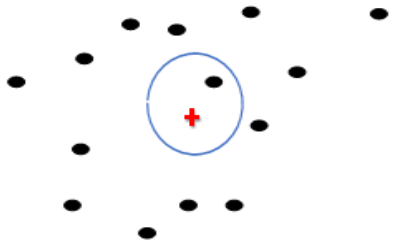

Fig. 1. KNN search problem for k=1. black points is reference point and red plus is query point. circle shows nearest point to the query point

serial Pseudocode nearest neighbor search algorithm in [1] is shown in figure 2.

```
//curPoint is queryPoint

For(int curPoint=0;curPoint<count;curPoint++)

For(int i=0;i<count;i++)

Compute all distance between query curpoint and r_j,
j€[1, count]
```

In this Pseudocode Curpoint variable is as a query point which its closest neighbor should be found. Count variable expresses the total number of set of points.

The main point in this algorithm is its high time complexity. With respect to the internal loop nearest neighbor search for a point of time complexity is calculated and with regard to external loop, finding the nearest neighbor operations are performed for all points. so in figure 2, complexity of algorithm is of order $O(N^2)$. equation 1 was used for calculating distances between two point in three dimensional coordinate.

$\text{Sqrt}((x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2)$ (1)

3. 1. Parallel Programming

Modern graphics processing units for the production and processing of high-quality 3D graphics are designed. These graphics processing units are very common on personal computers and improve overall system performance in terms of reducing runtime. In the graphics processing units a lot of the same calculations on a number of different data items to be run one after another(SIMD). For example, it may same computing for a every pixel in a frame or every vertex in a page be calculated. Now modern graphics processing units are including hundreds of processor cores and are able to manage the thousands of threads Simultaneously [6]. In November 2006, Nvidia introduced CUDA (Compute Unified Device Architecture), is a general purpose parallel computing architecture and provides opportunity for developers that their C language programs run on graphics processing units whereas this graphics processing unit previous was used only for graphic processes. Advantages such as low cost of graphics processing units and Easy to learn CUDA has caused it to be widely used in the scientific community. Some research shows that Algorithms have been implemented in parallel with the CUDA is hundreds of times faster than the serial implementation of this algorithm[1, 4, 7].

3. 2. Parallelism Nearest Neighbor Search Algorithm

The aim is reducing runtime for nearest neighbor search algorithm using the technology of CUDA. Also according to advancement of technology and the availability of modern graphics cards, Conditions is provided that addition to increasing system response speed become optimal use of system hardware. The response time is, the time between the beginning of the operations finding the nearest neighbor Until the end of the operation. As already mentioned, Many factors, including the type and number of processor, graphic card's type, and more importantly, the parallelism of the program is effective in achieving this goal. According to the pseudocode in Figure 2 for finding the nearest neighbors to each point there are two loop that The time complexity of the serial execution is of the order $O(N^2)$. from this pseudocode can be concluded that finding the nearest neighbor has good capability For parallelization. This must be done in a manner that threads communication overhead with global memory as far as possible may be reduced and does not overcome on runtime. The internal loop can not be parallel because the calculation of the nearest neighbor to a query point is independent of calculation of The nearest neighbor for the rest.

3. 3. Parallelism in global memory

According to equation 1, Kernel obtain distance of each point to query point. There is many way for problem fraction and giving it for running to threads. So problem was broken and was given to threads. The number of threads equal to the number of points of collection. Thus, each thread is mapped to a point and search other points in order to find it's nearest neighbor.

```
__global__ void FindNearestglobal(Float3*
points, int* indices, int count)
{
if(count<=1) return;
Idx=threadIdx. x+blockIdx. x*blockDim. x;
Determine a defult value for SmallestSoFar
For(i=0;i<count;i++)
 If(i=Idx) continue;
 Calculate Distance reference Point I with
 query point
 if distance is less of smallestSoFar
 smallestSofar=distance
 save I in indices[Idx]
 end If
end For

}
```

Fig. 3. Pseudocode for parallel execution threads accessing to global memory

Now, according to the pseudocode in figure 3 will be discussed in detail. Points array including three-Dimentional points coordination in the indices array that keeps its nearest neighbors. like pseudocode in figure 2 Count variable represents total number of points. when Points array in main method was initialized by programmer with random numbers. kernel calculation nearest neighbor to each point maps to a thread. the number of points is equal to the number of threads. in order to evaluation time complexity, implementation of this algorithm on a graphics processing unit without accessing to shared memory is compared with condition that serial algorithm runs on processor. Each block contains 320 thread that has been set by the programmer. In For loop each thread is bound to find the point closest and stores index of nearest neighbor in element of Indices array. order of complexity in figure 3 due to a for loop is O(N).

4. SUGGESTED METHOD

One of GPU facilities is shared memory per block. shared memory size is small, but its speed is high. more shared memory is used for communication threads of a block together or storage and retrieval data that threads need constantly. if this memory does not exist for communicating threads together or storage and retrieval of needed data, constantly have to access to GPU's global memory. since speed of global memory is low causes a sharp reduction in speed of execution whole program.

4.1. Parallelism using shared memory of graphic processing unit

Now to describe nearest neighbor algorithm parallelism using GPU's shared memory. Pseudo-code in figure 4, shared point array is consists of the threads in a block. every time, kernel transfers a block from global memory to shared memory. every thread in a block represents a point of set of points and obtains its nearest point among shared memory's threads within block. in this way, all blocks are copied in shared memory one after another and current threads obtain its distance with all points of set, the procedure is continued as long as each thread finds its nearest point. During the implementation, shared memory is shared among block threads. so shared memory is applied as a rapid method for communication between threads of a block. It is necessary to notice when shared memory is applied as a communication and interaction between threads, The use of shared resources in parallel programming causes Hazard. Generally, semaphore is often used for avoiding of hazard.

```
__global__ void FindNearestShared(Float3* points,
int* indices, int count){
__shared__ SharesPoint[blocksize]
Obtain ThreadIndex and save in Idx
Determin a defult value -1 for indexOfNearest
Determin a defult value -1 for distanceToNearest
Define a thisPoint variable
If(Idx< count)
Thispoint=points[Idx]
For each block in grid
{ if threadIdx in currentBlock<count
Copy thread value in sharedPoints Array
__syncthreads()
If( threadIdx< count)
For each thread in block calculate distance
with other threads in block
if(dist<distanceToNearest&& threadIdx
in currentBlock<count && threadIdx in
currentBlock!=idx)
DistanceTonearest=dist;
        indexOfnearest=thread in currentBlock
end if
end for
end if
__syncthreads()
End for
If(ind<count)
Indices[idx]=indexOfNearest
}
```

Fig. 4. Pseudo-code algorithm parallelism by using of GPU's shared memory

Using semaphore causes part of the program runs a serial that reduce or neutralize the aim of graphic processing unit which it is algorithm parallelism for increasing speed of algorithm execution. when two threads want a value of shared memory to modify in two different value, The

competitive situation arises. In CUDA to avoid competition and create hazard, instead of semaphore used sync thread method. This method acts as a barrier, until all thread of a block reach to this point, any threads in a block will not pass it.

5. Implementation Results

To evaluate the suggested algorithm, the number of set of points 100, 2000, 6000, 8000, 10000 are considered and for each of those sets, program is executed 20 times. As already mentioned, Points array of Random Numbers in range [-5000, +5000] is generated. Because these numbers are often used in real application. Once in suggested method and two mentioned method in section 3, Algorithm's execution time for a number of different points in 20 times is measured, Arithmetic mean of each of these 20 numbers can be considered as the runtime.

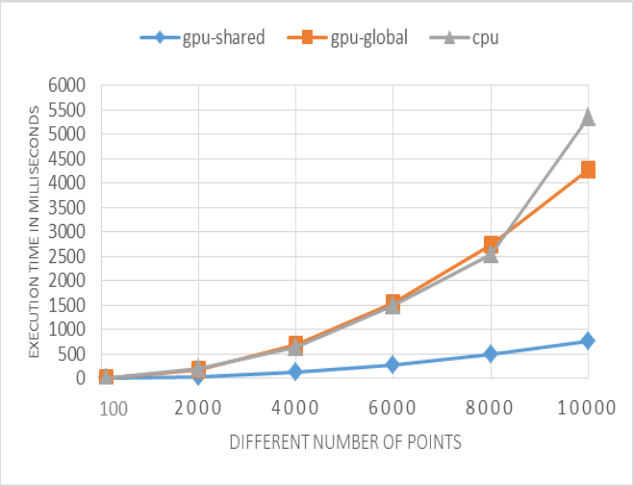

Fig. 5. Three method implementation runtime for number of different set of points

As seen in figure 5, Suggested method for large amount of data in comparing to another methods has much higher efficient in terms of time complexity. Parallelism method by CUDA on the graphics processing unit, in the case the threads have great access to the global memory and shared memory is not used, runtime doesn't improve. in order to compare the effectiveness of the suggested method and serial, the acceleration coefficient is used. Accelerating coefficient as the ratio of the time required to solve the problem with serial to the time needed to solve the problem with the suggested method is defined. Also acceleration coefficient for the ratio serial time to the parallel implementation of the algorithm without access to a GPU's shared memory is calculated. Equation 2 is used for calculating the results of the acceleration coefficient.

$$S(n)= t_s/t_p \quad (2)$$

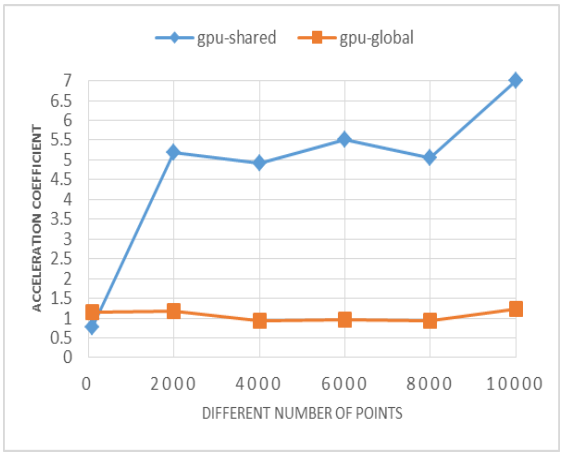

Fig. 6. Acceleration coefficient for the proposed method and the parallel without the use of shared memory than serial methods for different number of points

6. Summery and Conclusion

According to the obtained results that can be clearly seen, the suggested algorithm reducing the execution time especially when the data volume is high has a considerable impact. As previously mentioned, nearest neighbor search problem is easily for low-volume data But when the data size is large, algorithm's time complexity is high. Therefore, the suggested method has the best implementation in terms of running costs, efficient use of the hardware and scalability and Unlike the LSH method returns a definitive answer as nearest neighbor.

Reference

[1] Garcia, V., Debreuve, E., & Barlaud, M "Fast k nearest neighbor search using GPU". In Computer Vision and Pattern Recognition Workshops, CVPRW'08. IEEE Computer Society Conference on, Pp. 1-6. (2008, June).

[2] Nickolls, J., Buck, I., Garland, M., & Skadron, K. . "Scalable parallel programming with CUDA". Pp. 40-53. (2008)

[3] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., & Steinberg, D. " Top 10 algorithms in data mining". Knowledge and Information Systems, *14*(1), Pp. 1-37. (2008)

[4] Ryoo, S., Rodrigues, C. I., Baghsorkhi, S. S., Stone, S. S., Kirk, D. B., & Hwu, W. M. W "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA". In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. Pp. 73-82. . (2008, February).

[5] Nourzad, P, "Find the nearest neighbor in high dimentional spaces", The introduction of context-sensitive Hashing Algorithm. (2010)

[6] CUDA, C. Programming Guide: "CUDA Toolkit Documentation".

[7] Neumann, A. " Parallel reduction of multidimensional arrays for supporting online analytical processing (olap) on a graphics processing unit (GPU)"

AUTHOR'S PROFILE

Neda Mohammadi. She is studying for a master's degree in Shiraz University of Technology. Her research interests include presentation of solutions for algorithms parallelism which have high time complexity, CUDA programing on GPU, web service for transaction composition