Top-k Queries Processing With Uncertain Data on Graphics Processing Units

Haozhe Chang[1], Tingting Qin[2], Xiaoguang Liu[2], Gang Wang[2], Airu Yin[1]
[1]*College of Software, Nankai University, Tianjin, 300071, China*
[2]*College of I.T., Nankai University, Tianjin, 300071, China*
AngelClover@yahoo.cn, PaulaQin@163.com, {liuxguang,wgzwp,yinairu}@gmail.com

Abstract

Considering the complex uncertain database, top-k query processing in uncertain databases is semantically and computationally different from classical top-k processing. Score is not the only factor we should concern. The interplay between score and membership uncertainty makes computation complex. Powerful computing capability of Graphic Processing Unit(GPU) is needed in the processing of this kind of queries if we want to acquire the results as soon as possible. Using GPU with batch mode, we present a CPU-GPU cooperative computing framework to processing top-k queries in uncertain database. Two parallel GPU algorithms are designed to solve the problem specifically. Moreover, a "label-confidence" data format conversion is proposed to reduce CPU-GPU communication. We also suggest an error-correction method with the heap-based algorithm to improve the accuracy and correction of the results. Experimental results show that the CPU-GPU framework provides a better performance and it is quite efficiency in handling uncertain top-k problem.

1. Introduction

In recent years, many advanced technologies have been developed to store and record large quantities of data. In many cases, the data may contain errors or may only be partially completed. Uncertain databases have received more and more attention recently due to the enlargened number of applications which require the management of uncertain or fuzzy data. For example, sensor networks typically create large amount of uncertain data, maybe with certain probability. In some other cases, the data points may correspond to objects which are only vaguely specified, or mistaken by manual work which always be correct with probability. It has become an important issue to process uncertain data in many applications.

For a given uncertain dataset, there are many possible instances called *worlds*, and the *possible worlds* semantics has been widely used. In practical, a reliable set of k tuples is proved to be kinder than a disordered one. With an uncertain dataset, Top-k query processing can always give us a more reliable k-set.

Table 1. A sample of uncertain dataset

Time	Sensor Loc.	Temperature	Confidence
$11:30$	$L1$	$98°C$	0.4
$11:47$	$L2$	$95°C$	0.8
$11:51$	$L3$	$97°C$	0.6
$11:53$	$L4$	$99°C$	0.5

In this paper, we present two new GPU algorithms for Top-k Query processing.

The paper is organized as follows. An overview of the *Uncertain Top-k Query* problem will be given in Section 2, as well as the advantages in using GPU. Section 3 presents the GPU Top-k query algorithm. Section 4 presents two strategies based on the new algorithm. Experimental results are arranged at Section 5. Conclusion and future work is discussed in Section 6.

2. Uncertain Top-k Definition and GPU Architecture

In this section, we introduce two definition of uncertain top-k problem. Our algorithms are based on definition 2. We also introduce the GPU and CUDA architecture, which is also the base of our optimization.

2.1. Uncertain Top-k Definition

Top-k queries have been recently studied in the setting of uncertain data, which is shown in [1]. Given a ranking function, the goal is to find the top-k ranked tuples in a given uncertain dataset. In [10], *Soliman* et al. defined two types of top-k queries over an uncertain dataset, called U-Top$k$ and U-kRanks. In [4], *Hua* et al. defined a probabilistic threshold top-k query, denoted PT-k. We choose the first kind of definition U-top$k$ as an illustration.

Definition1 Uncertain Top-k query(U-Top$k$)[10]. Let $\mathcal{D}$ be an uncertain database with possible worlds space $\mathcal{W}$. For any $W \in \mathcal{W}$, let $\Psi(W)$ be the top-k tuples in W by the score attribute; if $|W| < k$, define $\Psi(W) = \emptyset$. Let T be any set of k tuples. The answer T^* to a U-Top$k$ query on $\mathcal{D}$ is $T^* = arg\,max_T \sum_w Pr[W]$. Ties can be broken arbitrarily.

Here is an example of possible world of heat sensor network. **Table 1** shows an uncertain dataset sample of a

Table 2. A sample of possible world

World	Probability
$PW^1 = \{L1, L2, L3, L4\}$	0.096
$PW^2 = \{L1, L2, L3\}$	0.096
$PW^3 = \{L1, L2, L4\}$	0.064
$PW^4 = \{L1, L3, L4\}$	0.024
$PW^5 = \{L2, L3, L4\}$	0.144

heat sensor network, with confidence values. There are 16 possible worlds for all the heat sensors. Here, a possible world is a set of heat sensor readings associated with a probability of the set, which is computed based on both the existence of all the tuples in the possible world and the absence of all tuples in the dataset that are not in the possible world, assuming mutual independence among the tuples. Every k-set of tuples constructs a possible world, which was shown in **Table 2**, with $k \geq 3$. Considering the 5-th possible world $\{L2, L3, L4\}$, in which the existence probabilities are $0.8, 0.6, 0.5$, and the absence probability of $L1$ is $(1 - 0.4)$. Therefore the probability of the possible world is $0.144 (= (1 - 0.4) \times 0.8 \times 0.6 \times 0.5)$.

In order to make clear of the possibility, we come to this definition:

Definition2 Let $\mathcal{D} = (D, p, f)$ be an uncertain dataset. For any $W \subseteq D$, let $\Psi_k(W)$ be the top-k elements in W by the score function f; if $|W| < k$, define $\Psi(W) = \emptyset$. Let T be any set of k tuples. The answer T^* to a U-Top$k$ query on $\mathcal{D}$ is $T^* = arg\,max_T Pr_{W \sim \mathcal{D}}[\Psi_k(W) = T] = arg\,max_T \sum_{\Psi_k(W)=T} Pr[W|\mathcal{D}]$. Ties can be broken arbitrarily.[2]

The top-k result gives us a quite good idea of where the k highest temperature occurs. As a convention, we assume that all the scores are distinct and $\mathcal{D}$ is given in the decreasing score order, for example, $f(1) \geq f(2) \geq ... \geq f(n)$. Thus, the probability of a set T of size k being the top-k elements $Pr_{W \sim \mathcal{D}}[\Psi_k(W) = T]$ comes out to be

$$\prod_{j \in T} p(j) \prod_{j \leq l(T), j \notin T} (1 - p(j))$$

where $l(T)$ is the last element in T. The problem becomes finding the set containing k elements T^* that maximizes the above quantity. Ties can be broken by choosing a smaller $l(T)$ or a smaller label.

Neglecting the naive enumerative algorithm, there are several excellent CPU algorithms proposed in recent years. In [10], *Soliman* et al. proposed a $O(n\dot{k})$ dynamic programming methods avoiding redundant calculation by compressing the optimal subspace. After transformed this problem using definition 2, *Yi* et al. proposed a heap-based algorithm, which takes $O(nlogk)$ operations to get the answer, in [12]. Then there outcomes many variants of the top-k query problem. For example, *Chen* and *Yi* proposed a problem maintaining top-k tuples in a dynamic set, using a binary tree, in $O(klogklogn)$ time to handle an update and in $O(logn + j)$ time to handle a top-j query, in [2]. *Jin* and *Yi* proposed sliding-window top-k queries and solve it in [6]. They also suggested an optimizing algorithm with x-relations which runs in near linear or low polynomial time and cover both types of top-k queries which are mentioned in definition 1 and 2 respectively in uncertain databases[13].

2.2. GPU And CUDA Architecture

In top-k queries in uncertain database, the processing speed and the degree of parallelism are significant factors people concerned. In this paper, we proposed a CPU-GPU cooperative processing framework which will be introduced in section 3.2 in detail. Using GPU brings advantages in top-k queries processing in uncertain database. A GPU is a collection of multiprocessors at the hardware level. And each multiprocessor has several elements to support thousands of threads simultaneously, named scalar processors (SP).

In this paper, we use Compute Unified Device Architecture (CUDA) [9] platform released by Nvidia Inc. CUDA supports a programming interface for parallel general purpose computing . In CUDA, threads are organized into *thread blocks* and distinguished by *threadIdx*. Another important feature is that memory space has different hierarchies with different access speeds. It brings us a challenge to store data effectively to upgrade the access efficiency.

3. GPU Algorithm for Uncertain Top-k Queries

As we known, the massive on-chip parallelism of GPU may greatly reduce the processing time if the computing process is good designed. But there is the extra cost in transmitting data from CPU to GPU. In order to make use of the powerful computing ability of GPU, and liberate CPU from complicated computing task, we propose a GPU algorithm to solve the uncertain top-k query problem.

3.1. Data format conversion

We propose a “label-confidence” data format conversion to reduce communication between CPU and GPU, which is generally the bottleneck of GPU algorithms. We label tuples by natural numbers, so each tuple is represented by a unique label number, and in our GPU algorithm, each tuple can be converted into a combination of a label number and a confidence value.

For example, we regards the temperature of the heat sensor as a score function, then the dataset in **Table 1** will be converted into **Table 3**. When we transmit this dataset to the GPU, only the label and the confidence of each tuple are transmitted to the GPU memory. After processing the queries on GPU, CPU will gets the top-k set of a query, in the form of labels. By contracting to the original dataset, we can easily acquire the top-k set.

Table 3. A sample of label-confidence format

Label	Time	Sensor Loc.	Temperature	**Confidence**
2	$11:30$	$L1$	$98°C$	0.4
4	$11:47$	$L2$	$95°C$	0.8
3	$11:51$	$L3$	$97°C$	0.6
1	$11:53$	$L4$	$99°C$	0.5

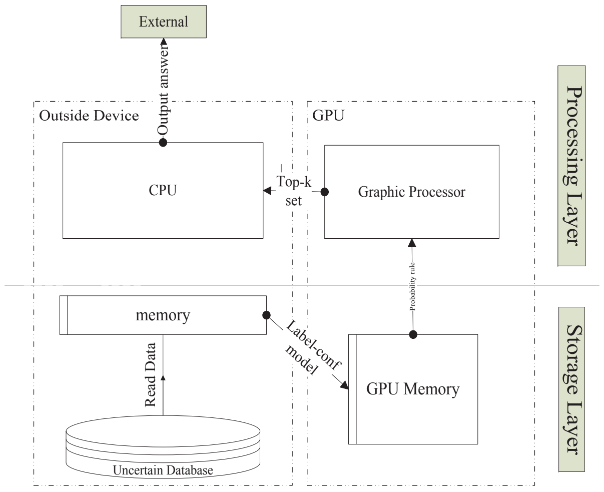

Figure 1. GPU processing framework

3.2. CPU-GPU Framework

Since uncertain data is likely to be stored in a traditional database, most of current uncertain database system prototypes rely on relational DBMSs for efficient retrieval and query processing. In [10], *Soliman* et al. propose a novel processing framework that leverages RDBMS storage, indexing, and query processing techniques to compute uncertain top-k query answers. However, GPU can not directly access RDBMS residing in the CPU host, so we propose a CPU-GPU cooperative processing framework consisting of two main parts(CPU and GPU part), and two layers(storage and processing layer), as Figure 1 shown.

Using this framework, we can tackle the top-k query with GPU effectively. In next section, two GPU algorithms based on this framework will be introduced, and the methods handle the problem will be introduced as well.

4. GPU Algorithms

GPU is used in many applications, such as inverted list compression, list intersection, and top-k scoring. For the classic top-k query problem, many distributed algorithms have been proposed [8], [11]. While there exist many algorithms calculating the top-k query, such as efficient exact algorithm, fast sampling algorithm, Poisson approximation based algorithm, and a typical vector sample algorithm proposed in [5], [3]. All the algorithms mentioned above are designed for CPU. In this paper, we proposed two algorithms based on GPU, according to the framework discussed on the last section.

4.1. The algorithm in CPU part

In the CPU part, we will transform the dataset from the uncertain databases to label-confidence pairs. Detail algorithm is shown in **Algorithm 1**.

Algorithm 1: CPU part

```
begin
    for each batch of queries do
        read the batch of queries
        for each query do
            $T \longleftarrow$ dataset corresponding to the query
            convert $T$ into label-confidence pair $P$
        end
        upload the batch of $P$ from CPU to GPU
        invoke GPU kernel
        download the results from GPU to CPU
    end
end
```

The GPU kernel implements the actual uncertain top-k query processing.

4.2. The GPU Dynamic Programming Algorithm

In [10], *Soliman* et al. proposed a $O(nk)$ dynamic programming method. Our GPU algorithm is similar to it. Since GPU processes queries in batching mode, we assign each query to a unique GPU block. Since the dynamic programming(DP) algorithm can be regarded as a process calculating elements in a $n \times k$ matrix, we assign each column of the matrix to a distinct GPU thread in the query's host block. Therefore, each row of the matrix can be calculated by k GPU threads in parallel, and rows are still calculated one by one. Therefore, the ideal complexity of the GPU algorithm is $O(n)$.

4.3. GPU Heap-based Algorithm

In [12], *Yi et al.* proposed an efficient serial uncertain top-k algorithm. Our second GPU uncertain top-k algorithm is based on it. This algorithm uses a minimum heap to maintain k tuples with largest confidences. It initially constructs the heap using any k tuples, and then iterates remaining tuples. For each tuple, its confidence is compared with the confidence of the heap root. If the tuple has the lower confidence, it is simply discarded, otherwise it will take the place of the heap root. In each iteration, the probability of the world composed of the tuples in the new heap is calculated.

Algorithm 2: GPU *DP* algorithm

```
Input: $confidence$
Output: $value$,$result\_set$
begin
    $bid \longleftarrow$ block index number in grid
    fetch the $bid$-th query
    $tid \longleftarrow$ thread index number in block
    if $tid > k$ then
        return
    end
    if $tid == 0$ then
        $dp(0, tid) \longleftarrow 1$
    end
    else
        $dp(0, tid) \longleftarrow 0$
    end
    value $\longleftarrow 0$
    __syncthreads()
    for $i = 0..n - 1$ do
        $dp(i + 1, tid) \longleftarrow max(dp(i, tid - 1) \times confidence(i), dp(i, tid) \times (1 - confidence(i)))$
        record the $footprint$ - the choice of $min$ operation
        if tid==$K$ then
            $value \longleftarrow max(value, dp(i + 1, tid))$
        end
    end
    if $tid == k$ then
        $result\_set \longleftarrow \phi$
        $result\_set \longleftarrow$ construct top-$k$ results from the $footprint$
        return $value$ and the $result\_set$
    end
end
```

If it is larger than the previous largest probability, it and the world are recorded. *Yi et al.* have proved the correctness of this algorithm [12].

Since parallel heap maintaining is inefficient, our GPU heap-based algorithm assigns each query to a unique GPU thread rather than a GPU block. That is, this algorithm uses a “query partition” rather than “query parallel” strategy. As described above, if the tuple P_i takes the place of the heap root P_j, the probability should be recalculated. It should be divided by $confidence(j)$ an $1 - confidence(i)$ and multiplied by $confidence(i)$ and $1 - confidence(j)$, that is, eliminating P_j from the world, and including P_i in. Since division operation will bring rounding error, we propose a probability tree to avoid division operations. This tree has the same structure as the confidence heap. Each node in this tree records the probability of the world composed of the tuples in its sub-tree. Therefore, when the heap is updated, the new probability can be recalculated using only

Algorithm 3: GPU *heap-based* Algorithm

```
Input: $confidence$
Output: $value$,$result\_set$
begin
    $bid \longleftarrow$ block index number in grid
    $tid \longleftarrow$ thread index number in block
    constructs a minimum heap $H$ using the first $k$ tuples $P_0$ $P_{k-1}$
    constructs the probability tree $T$ according to $H$
    res $\longleftarrow$
    the probability of the world constructed by tuples in $H$
    value $\longleftarrow res$
    result_set $\longleftarrow currentworld$
    for i=k..n-1 do
        if confidence(i) > confidence(root of $H$) then
            delete the root of $H$
            insert $P_i$ to $H$
            adjust $T$ accoding to $H$
            res $\longleftarrow$
            the probability of the new world
            if res>d value then
                value $\longleftarrow value$
                result_set $\longleftarrow newworld$
            end
        end
    end
end
```

multiplication operations in this tree. Moreover, to avoid the float number precision lost in multiply operation, we substitute the corresponding log value for the possibility. Furthermore, we can use a method in numerical calculation to reduce the influence of precision lost, for example, Kahan summation algorithm[7].

5. Experimental Results

We implemented our GPU framework on the **NVIDIA GeForce GTX 480**, which has 2GB memory and the CUDA version is 3.1. For the two algorithms we proposed in the last section, the implementation of the corresponding CPU version is accomplished also. All of these experiments are performed on a **Linux server (RedHat 5.1)** with **Intel i7 930 CPU (2.8GHz)**.

We created synthetic datasets each with $100,000$ cases. For each case, n denotes the number of tuples, while k denotes the size of required top-k answer. We performed four groups of experiments, shown in **Figure 2**. Where the DP-GPU line describes the processing time of **Algorithm 2**, Heap-GPU line describes **Algorithm 3**, DP-CPU and Heap-CPU lines describe the processing time of the corresponding algorithms implemented on CPU. The dataset of each group

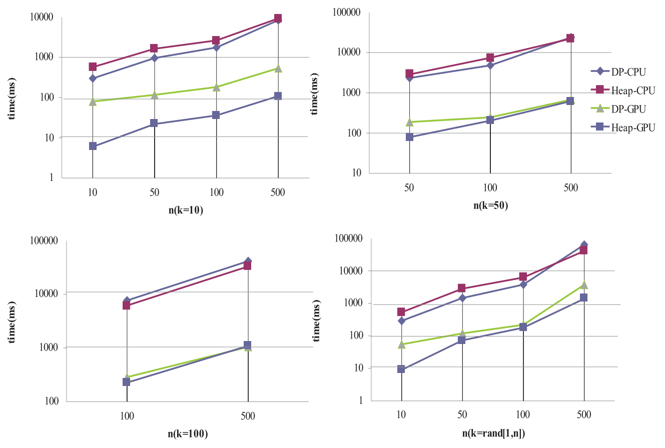

Figure 2. Processing time with a fixed $k(=10)$

determined a different upper bound of n: $10, 50, 100, 500$, and got a random number n, guaranteeing $n \geq k$. In our first three groups of experiments, we fixed $k = 10, 50, 500$, respectively, while that in the fourth group of experiments, we randomly choose k in the range $[1, n]$, shown in the last group. The probability of each tuple is uniformly distributed in $[0, 1]$ with the precision 0.001.

All the four groups suggest that, processing the top-k query on GPU is much faster than the classic CPU algorithms. On GPU, two algorithms run nearly the same. The heap-based algorithm is obviously faster with a quite larger batch when k or n is small. When k is small, the rate is about 200, and flops to 100 when $k \geq 500$.

In our GPU heap-based algorithms, in order to get an accurate answer, we use another heap and Kahan summation algorithm to maintain the probability of possible world, so that it looks a bit slower. But in general application, k is usually not big, and we will get a better effect if use the GPU heap-based algorithm. In this case, it suggests that the GPU heap-based algorithm is better, while in other cases, the difference of cost time between two GPU algorithms seems not obvious.

6. Conclusion

Top-k queries is arguably one of the most important queries in uncertain databases. This paper based on one of the uncertain top-k query problem's definitions mentioned above, proposed a CPU-GPU cooperative computing framework and two parallel GPU algorithms to tackle the problem. Besides, a "label-confidence" data format conversion is also proposed and implemented to decrease the transmission cost from CPU to GPU. Moreover, an error-correction technique is also suggested to improve the accuracy and correction of the results. Experimental results confirms the efficiency and scalability of our framework and algorithms.

7. Acknowledgments

This paper is partially supported by NSFC of China (60903028, 61070014, 61170184), Key Projects in the Tianjin Science & Technology Pillar Program (11ZCK-FGX01100).

References

[1] C. Aggarwal and P. Yu. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, pages 609–623, 2009.

[2] J. Chen and K. Yi. Dynamic structures for top-k queries on uncertain data. *Algorithms and Computation*, pages 427–438, 2007.

[3] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: On score distribution and typical answers. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 375–388. ACM, 2009.

[4] M. Hua, J. Pei, W. Zhang, and X. Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. of ICDE*, volume 8. Citeseer, 2008.

[5] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 673–686. ACM, 2008.

[6] C. Jin, K. Yi, L. Chen, J. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *Proceedings of the VLDB Endowment*, 1(1):301–312, 2008.

[7] W. Kahan. Pracniques: further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965.

[8] S. Michel, P. Triantafillou, and G. Weikum. Klee: a framework for distributed top-k query algorithms. In *Proceedings of the 31st international conference on Very large data bases*, pages 637–648. VLDB Endowment, 2005.

[9] C. Nvidia. Programming guide, 2008.

[10] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 896–905. IEEE, 2007.

[11] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data, June*, pages 09–12. Citeseer, 2008.

[12] K. Yi, F. Li, G. Kollios, and D. Srivastava. improved top-k query processing in uncertain databases. *Technical report, ATT Labs, Inc.*, 2007.

[13] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *Knowledge and Data Engineering, IEEE Transactions on*, 20(12):1669–1682, 2008.