Dr. Top-k: Delegate-Centric Top-*k* on GPUs

Anil Gaihre
Stevens Institute of Technology

Da Zheng
Johns Hopkins University

Scott Weitze
Stevens Institute of Technology

Lingda Li
Brookhaven National Laboratory

Shuaiwen Leon Song
University of Sydney and UW Seattle

Caiwen Ding
University of Connecticut

Xiaoye S. Li
Lawrence Berkeley National Laboratory

Hang Liu
Stevens Institute of Technology

ABSTRACT

Recent top-k computation efforts explore the possibility of revising various sorting algorithms to answer top-k queries on GPUs. These endeavors, unfortunately, perform significantly more work than needed. This paper introduces ***Dr. Top-k***, a Delegate-centric top-k system on GPUs that can reduce the top-k workloads significantly. Particularly, it contains three major contributions: First, we introduce a comprehensive design of the delegate-centric concept, including maximum delegate, delegate-based filtering, and β delegate mechanisms to help reduce the workload for top-k up to more than 99%. Second, due to the difficulty and importance of deriving a proper subrange size, we perform a rigorous theoretical analysis, coupled with thorough experimental validations to identify the desirable subrange size. Third, we introduce four key system optimizations to enable fast multi-GPU top-k computation. Taken together, this work constantly outperforms the state-of-the-art.

ACM Reference Format:
Anil Gaihre, Da Zheng, Scott Weitze, Lingda Li, Shuaiwen Leon Song, Caiwen Ding, Xiaoye S. Li, and Hang Liu. 2021. ***Dr. Top-k***: Delegate-Centric Top-k on GPUs. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458817.3476141

1 INTRODUCTION

Formally, top-k algorithms find the top k elements from an input vector V. Here, the criteria could be the top k largest or smallest, or any other conditions of interest. For simplicity, we assume the default criterion in this paper to be the top k largest. k-selection algorithm slightly differs from the top-k algorithm, as k-selection only identifies the k^{th} largest element from V. These two algorithms serve as building blocks for a variety of applications, such as, High Performance Computing (HPC) [26, 51], Information Retrieval (IR) [8, 11], deep learning training [3, 48, 49], big data [14, 15, 27], and data mining [25, 34, 54]. A textbook implementation of top-k exploits priority queue, i.e., min-heap. That is, a priority queue at the size of k will slide through the input vector. For each encountered element that is larger than the minimum from the priority queue, we substitute the minimum of the priority queue by this encountered element. Eventually, this priority queue captures the top-k largest elements for the input vector V.

Recently, the interest in deploying top-k computation on GPUs has surged for three major reasons. First, GPUs offer superior processing power and memory throughput comparing to the other processing hardware [17, 23, 47]. For instance, the most recent A100 GPU [28] features an astonishing 312 Tera Floating Point Operations Per Second (TFLOPS) computing capability and 2,039 GB/s memory throughput. Second, both the existing leading supercomputers [45] and future exascale ones (e.g., Aurora [46], Frontier [32] and El Capitan [41]) use GPUs as the major computing resources. Third, the majority of the applications that exploit top-k, such as IR [24], deep learning [1], data mining [36, 53], and database applications, e.g., PG Strom [35], Ocelot [7], and MapD [38] are offloaded atop GPUs, deploying top-k on GPUs could avoid copying data back and forth between GPU and CPU for top-k computation.

While priority queue-based top-k is the most efficient approach for single- or multi-core systems [55], it requires to maintain many local priority queues to expose massive parallelism to GPUs. Unfortunately, maintaining such many priority queues would experience expensive global synchronization overhead when merging these local priority queues into a final global one. Consequently, pertinent top-k applications do not adopt priority queue-based top-k. Instead, they use sort-and-choose approach for top-k computing on GPUs [6, 18, 33, 44, 48]. However, as shown in Figure 17, the GPU-based sort-and-choose top-k [6] takes much longer time than GPU-based top-k algorithms.

Revising sorting algorithms to compute top-k becomes a popular trend because, at most, only a subset of data needs to be sorted in the top-k problem. Along this direction, bitonic top-k [42] presents a revised bitonic sort algorithm [20] that focuses on the top-k elements when merging $2k$ elements together. Since this rudimentary design can only reduce the workload by half, [42] further proposes to read $8k$ elements and reduce it to k while using GPU shared memory to cache the intermediate results. Due to the limited capacity of shared memory on GPUs, bitonic top-k can only work for very small k (i.e., $k \leq 256$). Another notable attempt [2] revises bucket sort by discarding all buckets that do not include the k^{th} elements at each iteration, similarly for radix top-k. Despite these designs in [2] have the chance of reducing more workloads, they would suffer from unstable workload reductions (see Figure 6).

To reduce *more* workload in a *stable* manner, we introduce ***Dr. Top-k***, a delegate-centric system that partitions the input vector

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476141

into subranges, extracts the delegate from each subrange, and uses the top-k of the delegates to rapidly reduce the workload for the overall top-k computation on the input vector. It is essential to note that the popular IR algorithm, i.e., Block Maximum WAND (BMW) [11] also uses the delegate concept for search engine designs. In contrast, ***Dr. Top-k*** has a more comprehensive design and innovative usage for the delegate concept. Taken together, it can help state-of-the-art top-k algorithms to improve their performance significantly with the following three contributions:

First, we introduce a comprehensive delegate-centric design, which includes maximum delegate, top-k delegate-based filtering, and β delegate mechanisms to help reduce the workload for top-k up to more than 99%. Specifically, we (i) partition the input vector into a collection of subranges and extract the maximum delegate from each subrange to construct a delegate vector, and (ii) perform top-k on the delegate vector. Since *only those subranges whose maximum delegates belong to the top-k of the delegate vector can contribute to the top-k for the input vector*, we further (iii) concatenate those qualified subranges to construct a concatenated vector, and (iv) perform top-k on the concatenated vector. To further reduce the workload for step (iv), we use the minimum of the top-k of the delegate vector to filter out smaller elements from the qualified subranges. Not limited there, we extend the maximum delegate to β delegate to reduce the workload for concatenation and second top-k. In particular, we will extract the top β delegates, instead of merely the maximum, from each subrange. Afterward, we introduce a new rule using which we only concatenate subranges whose entire β delegates are taken.

Second, we deduce the optimal subrange size with both theoretical soundness and experimental validation. Note, a proper subrange size is crucial for ***Dr. Top-k*** to achieve a good performance; on the one hand, a small subrange size would lead to too many subranges. In this context, the delegate vector construction and first top-k would suffer from heavy workloads. On the other hand, when the subrange size is too large, we would have too few subranges. In this case, the majority of these subranges will be eligible for the second top-k. We hence skip too few subranges, leading to limited workload reduction for concatenation and second top-k. In Section 5.2, our theoretical analysis derives that the total time consumption of ***Dr. Top-k*** is a convex function of subrange size, which we also verify in our experiment. We further extract the optimal subrange size for a wide range of $|V|$ and k.

Third, we deploy ***Dr. Top-k*** atop multiple GPUs with four key system optimizations. First, we introduce a warp-centric delegate vector construction mechanism to achieve near-peak GPU global memory throughput. Second, although our delegate-centric design can help all existing top-k algorithms, we identify that the best ***Dr. Top-k*** assisted top-k algorithm changes along with the climbing of k. We further introduce a flag-based strategy to avoid random memory access during in-place radix top-k. Third, we identify that delegate vector construction suffers from low thread utilization and an exceeding usage of CUDA (an acronym for Compute Unified Device Architecture) shuffle instructions when k becomes relatively large. As a result, we introduce a novel *coalesced load to shared memory and strided compute approach* to improve the thread utilization, as well as curb the usage of CUDA shuffle instructions. This optimization has reduced the delegate vector construction time consumption from 31.4 ms in Figure 10 to 9.5 ms in Figure 15 for $k = 2^{24}$ and $|V| = 2^{30}$. Finally, we scale top-k across multiple GPUs to handle gigantic input vectors, and achieve sustained scalability.

During evaluation, we notice that the recent top-k efforts [2, 42] only test their systems on synthetic datasets, limiting the impacts of top-k. This paper hence builds a benchmark that contains three real-world applications, i.e., k-nearest neighbor search [21], website degree centrality [12], and COVID fear related Twitter dataset [19] for top-k. Our evaluation shows that ***Dr. Top-k*** can outperform the state-of-the-art on both synthetic and real-world datasets.

The remainder of this paper is organized as follows: Section 2 discusses the background and related work which motivate the overview in Section 3. Section 4 presents the delegate-centric top-k design and compares it against BMW. Section 5 deploys our top-k on multi-GPU systems with GPU-specific optimizations. Section 6 evaluates ***Dr. Top-k*** and we conclude in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Graphics Processing Units

Streaming processors and threads. Designed with NVIDIA Volta architecture, V100S [30, 31] is powered by 80 streaming processors (SMs). Each SM is equipped with 64 CUDA cores, yielding a total of 5,120 cores running at 1.5 GHz. During execution, a GPU thread runs on one CUDA core, and an SM schedules a group of 32 consecutive threads known as warp in a Single Instruction Multiple Thread (SIMT) manner. Note, all the threads in a warp can use shuffle instructions to exchange data. A collection of consecutive warps further formulate a Cooperative Thread Arrays (CTAs) or a block. All the CTAs are called a grid.

Memory architecture. V100S is equipped with 32 GB global memory with 1,134 GB/s as the peak throughput. All the SMs share an L2 cache of 6,144 KB. Each SM owns a private 96KB configurable shared memory, also used as the L1 cache. All the threads in a CTA can use shared memory to communicate with the help of the CUDA __syncthreads() primitive. It is desirable to use shared memory to cache intermediate data because it is around one order of magnitude faster than the global memory [42].

2.2 Related Work

This section discusses the closely related projects for ***Dr. Top-k*** that includes priority queue-based top-k [42], sorting-based top-k [6, 48], bucket top-k [2], radix top-k [2] and bitonic top-k [42].

Priority queue approach. A natural way to compute top-k would be to maintain a priority queue that only keeps the top-k elements while scanning through the input vector. While this idea is well-suited for single- or multi-threaded systems, implementing it on massively parallel GPUs remains elusive. Mainly, a parallel implementation would involve the maintenance of many local priority queues and the eventual merging of these priority queues into a global one. This adds the challenge of frequent read and write and global synchronization across the threads when merging them.

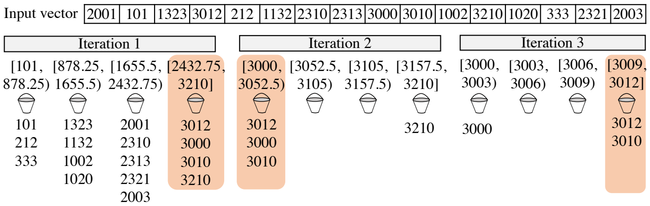


Figure 1: Bucket top-2 computation for an input vector with 16 elements. The highlighted bucket is of interest at each iteration.

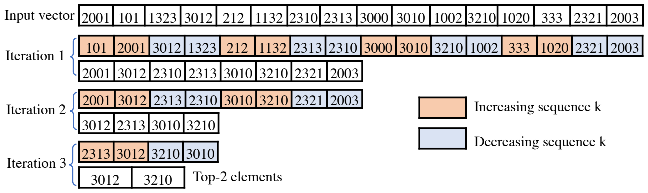


Figure 2: Bitonic top-2 on the same input vector as Figure 1.

Sort-and-choose is an alternative approach that is more friendly for parallel implementation. Basically, we sort the input vector elements using sorting algorithms, like in THRUST [6] and choose the top-k elements. But *this implementation turns out to do more work than necessary.* At least, there is no need to sort the elements that are outside of the range of $1^{st} - k^{th}$ elements. Alabi et. al. [2] also show their top-k algorithms, e.g., radix and bucket top-k outperforms the sort-and-choose designs.

Top-k algorithms. Bucket, radix, and bitonic top-k are introduced to alleviate the aforementioned inefficiency problems faced by sorting. In contrast to their corresponding sort-and-choose approach, the top-k algorithm distributes the input vector into different subranges, like a bucket in bucket top-k, and only focuses on the subrange that will lead to the k^{th} element of the input vector. Below we explain how these designs work with examples.

I. Bucket top-k first obtains the *min* and *max* values from the input vector. Afterward, it divides this *min* − *max* value range into several buckets, with each of which accounting for a disjoint equal value range. In the second step, this method scans through the input vector, puts each element into the corresponding bucket, and tracks the number of elements in each bucket. This way, one can easily figure out which bucket contains the k^{th} elements. As mentioned earlier, top-k operation discards the buckets that do not contain the k^{th} element. This method continues until the bucket of interest only has one element, i.e., the k^{th} one.

Figure 1 exemplifies how bucket top-k works for an input vector of 16 elements. We first derive the *min* and *max* as 101 and 3210, respectively. Therefore, we can divide this value range into four buckets, that is, [101, 878.25), [878.25, 1655.5), [1655.5, 2432.75), [2432.75, 3210]. Scanning through the entire input vector, one can obtain all elements that belong to each bucket as shown in iteration 1 of Figure 1. Since we are searching for the 2nd largest element from the input vector, our next iteration only focuses on the [2432.75, 3210] bucket, which is the largest bucket that contains four elements. Consequently, iteration 2 of Figure 1 divides the [2432.75, 3210] value range into four buckets and scans through the elements from the [2432.75, 3210] bucket of iteration 1 to generate new element distributions in iteration 2 of Figure 1. While Figure 1 only includes three iterations due to space constraints, this process is supposed to continue until the bucket of interest only contains one element.

II. Radix top-k is similar to bucket top-k but exploits the digits (i.e., radixes) of each element to determine which bucket a value belongs to. The key is that *the position of the bucket needs to indicate their order* so that we can derive the bucket of interest for the next iteration. Consequently, radix top-k starts from the Most Significant Digit (MSD) to the Least Significant Digit (LSD). Following this manner, for instance, if we process 3 bits at one iteration, we will need eight buckets, that is, 000, 001, 010, 011, 100, 101, 110, 111. And all the elements from bucket '111' are larger than those of '110'. Similarly for other buckets. At the end of each iteration, we only focus on the bucket that contains the k^{th} elements to proceed.

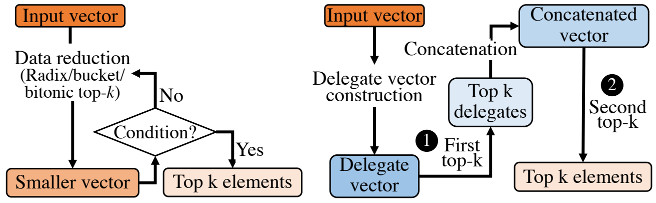
III. Bitonic top-k. Improving from the traditional bitonic sorting algorithm, bitonic top-k [42] proposes to discard k elements by selecting the top-k elements from a bitonic sequence of size $2k$. Therefore, the workload is always reduced by half. Figure 2 demonstrates how bitonic top-2 behaves for the same input vector in Figure 1. Particularly, this algorithm sorts every two consecutive elements in the input vector, as shown in Iteration 1. Afterward, it merges the adjacent two sequences – {101, 2001} and {3012, 1323} – and gets the top-2 from these four elements, that is, {2001, 3012}, similarly for remaining sequences. This process continues until Iteration 3, where we obtain the final top-2, as {3012, 3210}.

Some of the other related projects worth mentioning are a GPU-based bucket sorting [10] that takes samples from different regions in the input vector to achieve a good workload balancing. The work partitions the input vector into several subranges, performs a local sort in each subrange, and selects multiple samples from each subrange. These samples are collectively processed to guide data reordering on the original vector so that each bucket would end up with a similar amount of workloads. The top-k at [22] performs a priority queue based k-selection algorithm in register memory in GPUs. As the registers per thread in the GPU are limited to a few numbers, similar to [42], the performance degrades for k ≥ 1024. A recent work [37] uses sampling to make bucket select more immune of skewed data distribution. Particularly, this work samples splitters from the original vector. Then, these splitters are sorted and used to assign bucket ranges. While this work tries to adjust the bucket boundaries in order to reduce workload in each level, *our work directly reduces the original input vector for not only bucket top-k but also other ones, e.g., radix and bitonic top-k algorithms.*

3 CHALLENGES AND OVERVIEW

The state-of-the-art GPU-based top-k designs, as shown in Figure 3(a), directly work on the input vector when reducing the elements. This data reduction process continues until a desirable condition is met. Despite that such designs can outperform the traditional priority queue and sorting-based approaches, they still face the following two challenges:

- The performance of bucket and radix top-k is unstable. That is, they are sensitive to the value distribution of the data. For



(a) Bucket/radix/bitonic top-k workflow (b) ***Dr. Top-k*** workflow

Figure 3: Workflow of (a) Bucket/radix/bitonic top-k vs (b) *Dr. Top-k*.

instance, the radixes of interest might carry most of the elements from one iteration over to the next. Figure 4 presents the performance variation of ***Dr. Top-k*** on three different distributions Uniform (UD), Normal (ND) and Customized Distributions (CD), where the data distributions are rigorously defined in Section 6. We observe both radix and bucket top-k [2] experience performance variations when changing data distributions. And bitonic top-k [42] performs stably across different data distributions.

- While bitonic top-k can stably reduce the workload, it only reduces the workload by half at one iteration. To further reduce the workload, bitonic top-k requires tremendous shared memory to store the intermediate results. This is problematic for GPUs due to limited shared memory capacity. Figure 2 demonstrates how bitonic top-k reduces the workload only by half at an iteration when it selects top-2 elements from each bitonic sequence of length 4 at an iteration. For instance at iteration 1, from the first bitonic sequence of 4 elements {101, 2001, 3012, 1323} top-2 elements i.e. {2001, 3013} are selected to be written into new vector for next iteration. Similarly, remaining bitonic sequences in the vector go through same process. This leads the vector length to reduce from 16 to 8 at iteration 1.

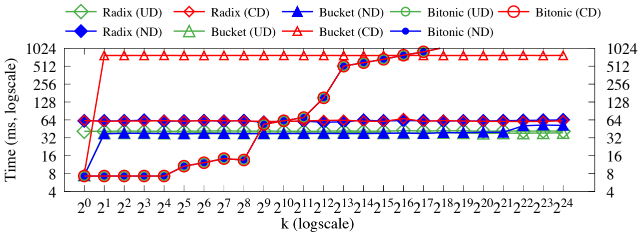


Figure 4: The performance inconsistency of different top-k versions on different distribution defined at Section 6.

Dr. Top-k, as shown in Figure 3(b), introduces the delegate-centric concept where top-k computation only happens on delegate and concatenated vectors which are small fraction of the original input vector. This warrants both stable and larger workload reductions during top-k computation on ***Dr. Top-k***. (i) Our workload reduction is *stable* regardless of the value distribution of the input vector. That said, for a given k and $|V|$, the workload is determined (detailed in Section 5.2). (ii) ***Dr. Top-k*** on average reduces a greater portion of the workload, compared to top-k algorithms such as bucket, radix, and bitonic top-k. This is because the total size of the delegate and concatenated vectors is smaller than the input vector, which is the input for the bucket/radix/bitonic top-k algorithms. This is evident in Section 6.2. It should be noted that *instead of being regarded as an alternative algorithm to the existing top-k algorithms,* ***Dr. Top-k*** *can help reduce workloads for all existing top-k algorithms, including bucket, radix, and bitonic top-k as long as we change the first and second top-k (❶, and ❷) into these algorithms.*

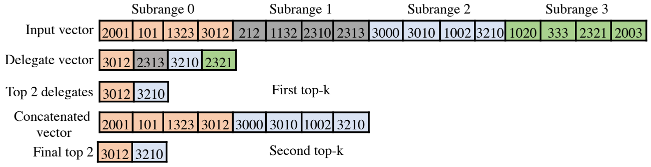


Figure 5: Maximum delegate-based top-2 computation for the same input vector V in Figure 1.

4 *DR. TOP-K*: DELEGATE-CENTRIC TOP-*K*

4.1 Maximum Delegate

Rule 1. *For a given vector V, $\exists D \in V$, such that D is the delegate vector containing the maximum elements of all subranges S_i. If the maximum m_i from S_i is not among the top-k elements in D, then S_i will not contribute any elements to the top-k of V.*

Rule 1 indicates that we can use the delegate of a subrange to decide whether to omit the entire subrange during top-k computation. Figure 5 presents an example about how to use Rule 1 to find the top k, (i.e., $k = 2$) elements from the same input vector as Figure 1. Essentially, we first divide the input vector into four subranges, with each of which containing four elements. Second, we extract the delegate, i.e., maximum element from each subrange to formulate a delegate vector, i.e., {3012, 2313, 3210, 2321}. The first top-k finds 3012 and 3210 as the top 2 elements from the delegate vector. This implies that only the subranges that contain 3012 and 3210 are qualified for concatenation. Therefore, we concatenate these two subranges and conduct the second top-k on the concatenated vector – {2001, 101, 1323, 3012, 3000, 1002, 3210}. Our second top-k derives the final top-2 as {3012, 3210}.

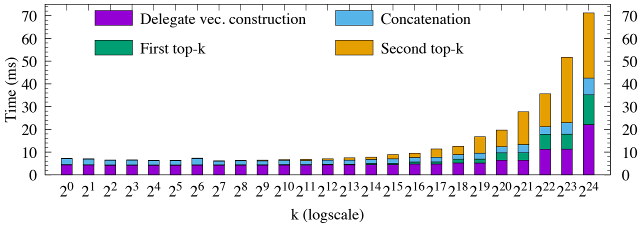


Figure 6: *Dr. Top-k* assisted radix top-k time consumption breakdown with respect to the increase of k for UD dataset on Section 6.

Leveraging Rule 1, we implement the initial version of ***Dr. Top-k***. Figure 6 demonstrates the time consumption breakdown of ***Dr. Top-k*** accelerated radix top-k for $|V| = 2^{30}$ unsigned integers with k ranging from 2^0 to 2^{24}. *For $k \leq 2^{15}$, the time consumption delegate*

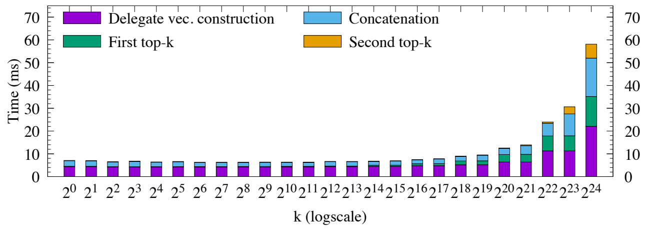


Figure 7: *Dr. Top-k* with delegate top-k enabled filtering for the UD dataset on Section 6.

vector construction is ~4.2 ms, which means we achieve 84% of the peak throughput of the V100S GPU, albeit delegate vector construction also performs additional shuffle instructions. When $k > 2^{15}$, the time consumption of ***Dr. Top-k*** also increases, which is reflected in all the four steps of ***Dr. Top-k***.

4.2 Delegate Top-k Enabled Filtering

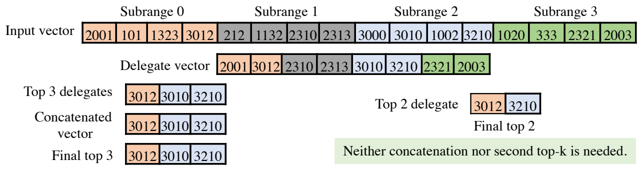
Although the maximum delegate of a subrange is in the top-k of the delegate vector, not all the elements in the qualified subrange will be eligible for the second top-k. This section uses the top-k of the delegate vector to remove elements from the qualified subrange during the concatenation, leading to further workload reduction for the second top-k through the following rule.

Rule 2. *The k^{th} element in the delegate vector is the minimum possible element the final k^{th} element can become.*

This rule can be derived as follows: the minimum of the top-k of the input vector V will be no less than that of the delegate vector D, i.e., $min(topk(D)) \leq min(topk(V))$. Therefore, only the elements that are larger than the $min(topk(D))$ are possible to get into $topk(V)$, hence are qualified for the concatenation. Here, $topk(V)$ denotes the top-k elements in V, similarly for $topk(D)$. We can use the example from Figure 5 to assist the understanding of this Rule 2. Here, the minimum element from the top-2 of the delegate vector is 3012. Our prior ***Dr. Top-k*** takes the entire subranges whose maximums are in the first top-k into consideration. This is, in fact, wasteful. For instance, the elements that are smaller than 3012 in both subranges 0 and 2, that is, 2001, 101, 1323, 3000, 3010, and 1002, will never become one of the elements in the final top-2. Hence, none of them should be copied to the new concatenated vector. Eventually, the concatenated vector is merely {3012, 3210}.

To implement this delegate top-k enabled filtering approach, we disseminate the minimum of the top-k from the delegate vector across all threads. Afterward, the threads are dispatched to work on the qualified subranges identified by the first top-k. When performing scan on those qualified subranges, only the elements that are larger than the minimum of the top-k of the delegate vector are stored in the concatenated vector. As the number of eligible elements from each subrange is unknown beforehand, each thread needs to use atomic operation [13, 16] to obtain the position to store the eligible element.

Figure 7 demonstrates the effectiveness of delegate top-k enabled filtering for the same dataset in Figure 6. Comparing Figures 7 and 6, one can observe that the benefits of this optimization for the second top-k is substantial, especially when $k \geq 2^{19}$. Using $k = 2^{24}$ as an example, we reduce the second top-k time consumption from 28.7 ms in Figure 6 to 6.1 ms.



(a) β=2 delegate for top-3 computation.

(b) β=2 delegate for top-2 computation.

Figure 8: Top-k with β delegate on the same input vector in Figure 1. Particularly, it shows different workload reductions for (a) $\beta = 2$ delegate for top-3 query and (b) $\beta = 2$ delegate for top-2 query.

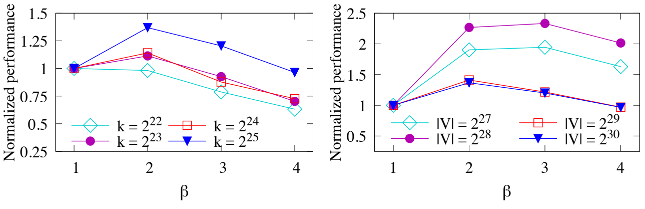
4.3 β Delegate

While delegate top-k enabled filtering can tremendously reduce the workload for second top-k, it still has two weaknesses that require further improvements: First, one might need to perform extensive atomic operations to build the concatenated vector. Second, we still need to scan through the qualified subranges to omit the elements that are smaller than the minimum of the top-k of the delegate vector. Now we introduce β delegate that allows ***Dr. Top-k*** to safely avoid the entire subrange without scanning any elements which would be qualified for second top-k if without β delegates. Below, we formally introduce the β delegate rule.

Rule 3. *In a subrange S_i, we select the top β elements as β delegates. If not all of the β delegates in S_i would qualify as top-k of the delegate vector D, the rest of the elements from this subrange will not qualify for the top-k of the input vector V. Note, $\beta \in N$ and $\beta > 1$.*

Figure 8 describes how to use Rule 3 to answer the top-3 and top-2 queries with $\beta = 2$. In this case, our delegate vector contains two delegates from each subrange. For top-3 in Figure 8(a), since we only take one delegate from subrange 0 and both delegates from subrange 2, we obtain the concatenated vector as {3012, 3010, 3210}. Note, even the concatenated vector is the same as the top 3 delegates, we still need to scan through the entire subrange 3 to omit the ineligible elements. Finally, the second top-k computes on the concatenated vector to figure out the final top-3. Figure 8(b) presents a special benefit of β delegate. That is, we might not need the concatenation and second top-k computation. In this case, since the top-2 of the delegate vector does not take all the β delegates from any subrange, Rule 3 suggests that neither concatenation nor second top-k is necessary.

Note, β delegate will lead to more workloads for the first top-k and delegate vector construction. To reduce the workload for the first top-k, we let the first top-k skip the final iteration when locating the exact bucket or radix of interest. Because β delegate and delegate top-k enabled filtering can substantially reduce the workload for concatenation and second top-k, this skipping, which helps first top-k noticeably, will lead to negligible performance drop for the subsequent concatenation and second top-k steps. For performance improvements on delegate vector construction, Section 5.3 will introduce our novel optimizations shortly.



(a) $|V| = 2^{30}$

(b) $k = 2^{25}$

Figure 9: The performance dynamics with respect to the change of β when (a) varying k at $|V| = 2^{30}$, and (b) varying $|V|$ at $k = 2^{25}$.

An appropriate β is important for ***Dr. Top-k***, our empirical study in Figure 9 suggests that $\beta = 2$ performs the best. For better visualization, we normalize the performance of various tests towards $\beta = 1$. In Figure 9(a), we find that $\beta = 2$ is the desirable configuration which increases the performance up to 1.41 when $k = 2^{24}$ from $\beta = 1$. Although Figure 9(b) observes slightly better performance when $\beta = 3$ for smaller $|V| = 2^{29}$ and 2^{30}, we find $\beta = 2$ always yield good performance across both figures.

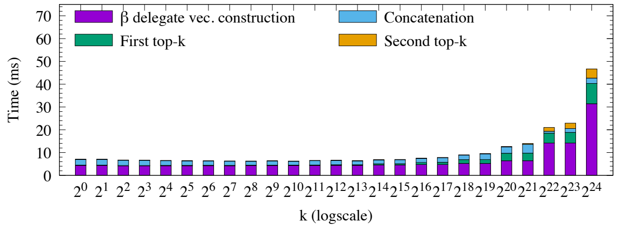


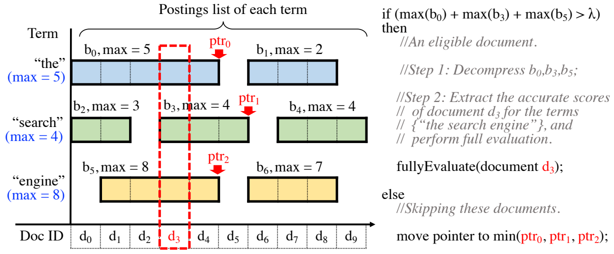
Figure 10: *Dr. Top-k* with β delegate and delegate top-k enabled filtering optimization for UD dataset on Section 6.

Figure 10 shows the time consumption breakdown of β delegate optimization. Using k=2^{24} as an example, although β delegate spends 31.4 ms for delegate vector construction and 8.9 ms for first top-k, it reduces the time consumption for concatenation and second top-k from 16.8 ms and 6.1 ms of Figure 7 to 2.3 ms and 4 ms, respectively. Overall, we reduce the time consumption from 58 ms of Figure 7 to 46.7 ms for $k = 2^{24}$.

4.4 Discussion: *Dr. Top-k* vs BMW Algorithm

This section compares our ***Dr. Top-k*** algorithm with a closely related IR algorithm, BMW [11], which is a variant of the popular Weak AND (WAND) algorithm [8]. Briefly, the BMW algorithm aims to find the top-k most related documents for a query term.

Figure 11 presents an example for BMW algorithm. For clarity, we first describe the settings of our example: (i) there are ten documents, i.e., d_0 - d_9; (ii) the query contains three terms: "the search engine", and (iii) the score of a term in a document is the number of occurrence of the query term in that document. BMW first puts the documents that contain each term together, subsequently sorts them by the document ID and partitions them into blocks, e.g., the term "the" contains two blocks $b_0 = \{d_0 - d_4\}$, and $b_1 = \{d_6 - d_8\}$. For each block, BMW stores the maximum score, e.g., the maximum score of block b_0 is 5. Assuming BMW is working on document d_3,



```
if (max(b_0) + max(b_3) + max(b_5) > λ)
then
    //An eligible document.

    //Step 1: Decompress b_0,b_3,b_5;

    //Step 2: Extract the accurate scores
    //  of document d_3 for the terms
    //  {"the search engine"}, and
    //  perform full evaluation.

    fullyEvaluate(document d_3);

else
    //Skipping these documents.

    move pointer to min(ptr_0, ptr_1, ptr_2);
```

Figure 11: BMW algorithm for a query {"the search engine"}.

the right side of Figure 11 is the pseudocode of BMW. Specifically, BMW first evaluates whether the block maximums of the three blocks that contain document d_3 would be bigger than the threshold λ. If this condition is true, BMW will perform a full evaluation on document d_3 and move on to the next document d_4. Otherwise, BMW will skip all the documents that exactly share the same block maximum with document d_3. In this context, using each block's maximum to estimate whether we should skip a document is similar to ***Dr. Top-k***, which uses the maximum of a subrange for delegate.

Distinction. While BMW leverages the block maximum to skip computations when extracting the top-k documents, ***Dr. Top-k*** designs and exploits the delegate concept more comprehensively from three aspects. First, ***Dr. Top-k*** introduces a delegate-centric processing concept while BMW still uses a regular element-centric concept. Here a regular element is a document. Particularly, ***Dr. Top-k*** uses the delegate to decide whether an entire subrange (i.e., a block in BMW) is eligible or not. However, BMW processes one document at a time. Using document d_3 in Figure 11 as an example, even d_3 is qualified, BMW still needs to perform the eligibility check for d_4. Further, for ineligible documents, BMW can only skip the documents that share the same or fewer terms than d_3. Second, we further introduce β delegate to help remove more subranges and delegate top-k enabled filtering to reduce some regular elements from the qualified subranges. Both of these designs are novel compared to BMW. Third, as we will discuss shortly in Section 5, ***Dr. Top-k*** also includes subrange size tuning and GPU-aware optimizations, which are also novel compared to the BMW algorithm.

5 *DR. TOP-K* IMPLEMENTATION AND OPTIMIZATIONS

5.1 GPU-based *Dr. Top-k* Design

Warp-centric delegate vector construction first divides the entire input vector into smaller subranges at the length of 2^α, where α is an integer. Afterward, each warp of GPU threads is assigned to extract each subrange delegate in three phases. Using maximum delegate as an example, every thread first records the maximum element when scanning through a specific subrange. Second, all the threads in each warp use *shuffle instruction*, i.e., __shfl_sync(), to communicate and derive the maximum element in the current subrange. During the third phase, ***Dr. Top-k*** writes each subrange's maximums and the subrange IDs to the delegate vector in the global memory. The size of the delegate vector is $\frac{|V|}{2^\alpha}$.

Warp-centric concatenation. This step concatenates the eligible subranges into a new concatenated vector where the second top-k performs on. Particularly, a warp of threads is responsible for moving the subrange elements into the concatenated vector. Because ***Dr. Top-k*** uses delegate top-k enabled filtering, the eligible elements per subrange are unclear. We resort to atomic operations to calculate the location for each eligible element.

First and second top-k. Once ***Dr. Top-k*** formulates delegate or concatenated vector, it will perform top-k on them. While both top-k algorithms work on a relatively small vector, the first top-k presents two unique features. First, this top-k algorithm has to work on a delegate vector that comes in the format of *(key, value)* pair. Here, the *key* is the delegate element from each subrange, and the *value* indicates which subrange this delegate element comes from, which is essential for the concatenation step. Second, the first top-k algorithm has to be a top-k operation instead of k-selection because one needs to extract all the top-k subranges for concatenation. We hence have to revise the radix and bucket k-selection algorithms of [2] to support top-k.

Choice of top-k algorithms. Despite the fact that ***Dr. Top-k*** can help all existing top-k algorithms, we notice that the best ***Dr. Top-k*** will favor different top-k algorithms when k changes. Particularly, (i) when k is small, all top-k algorithms will enjoy comparable performance gains over their baseline algorithms. However, for radix and bucket top-k, they prefer in-place designs that always work on the input vector V as instead of out-place variants that copy the derived candidates to a new array for the follow-up iteration. (ii) When k is large, the performance of ***Dr. Top-k*** assisted bitonic top-k will lag behind. Specifically, bitonic top-k needs a large shared memory space to cache the intermediate results and achieve desirable performance, which will experience low occupancy hence poor performance when $k > 256$. As shown in Figure 4, when k goes beyond 256, the performance of bitonic top-k degrades significantly. This makes ***Dr. Top-k*** assisted bitonic perform worse than other ***Dr. Top-k*** assisted ones.

Optimized in-place radix top-k. Since existing in-place radix top-k algorithm [2] requires to modify the ineligible element from the input vector into a value that is assured to fall out of the value range of interest (e.g., zero), this results in excessive random memory accesses. *We introduce a single flag variable to indicate the radixes of interest.* This flag tracks the radixes that are eligible for the next iteration. Subsequently, once an element is loaded from global memory, we will perform flag == (flag & loadedElement) between the loaded element and the flag variable. Only when the condition is evaluated as true, we consider this loaded element as a qualified element. As shown in Figure 12, our optimized in-place radix top-k is on average 10.7× faster than the state-of-the-art [2].

5.2 α Tuning

A proper subrange size is crucial for ***Dr. Top-k*** to achieve good performance. On the one hand, a small subrange size would lead to too many subranges. In this context, the delegate vector construction and the first top-k would suffer from heavy workloads. On the other hand, when the subrange size is large, there are too few subranges. In this case, the majority of these subranges will be eligible for the second top-k. We hence skip too few subranges, leading to limited workload reduction for concatenation and second top-k. Rule 4 helps ***Dr. Top-k*** derive an optimal subrange size.

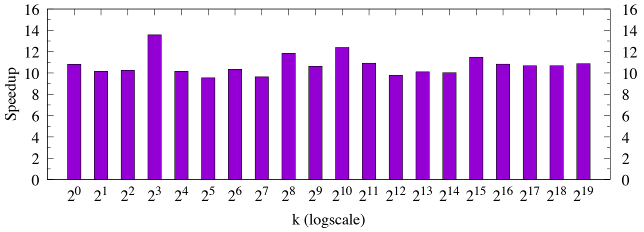


Figure 12: *Dr. Top-k* in-place radix top-k speedup over GGKS in-place radix top-k on uniformly distributed vector of size $|V| = 2^{21}$.

Rule 4. *For **Dr. Top-k**,*

$$\alpha = \frac{1}{2} \cdot [Const + \log_2(|V|) - \log_2(k)]$$

*leads to the optimal subrange size 2^α, where $|V|$ is the number of elements in the input vector V, k is the number of top elements **Dr. Top-k** aims to find. $Const = \log_2[6 \cdot (C_{global} + C_{shfl})] - \log_2(6 \cdot C_{global})$, where C_{global} and C_{shfl} are the clock cycles for one global memory access and one CUDA shuffle instruction, respectively.*

Proof. The time consumption of ***Dr. Top-k*** is:

$$T = T_{Delegate} + T_{FirstK} + T_{Concat} + T_{SecondK}, \quad (1)$$

where $T_{Delegate}$, T_{FirstK}, T_{Concat} and $T_{SecondK}$ are the time consumption of delegate vector construction, first top-k, concatenation, and second top-k, respectively.

Global memory access and intra-warp communication are the key factors determining the time consumption of ***Dr. Top-k*** for two reasons. First, one global memory access or intra-warp shuffle operation takes a much longer time than a single arithmetic and logic operation on GPUs, according to Nvidia profiler [29]. Second, the number of arithmetic and logical operations is similar to that of global memory accesses across all four stages of ***Dr. Top-k***. Using delegate vector construction as an example, each thread loads one element from global memory and compares, i.e., a logic operation, it with the current maximum in a register. In this case, one memory access leads to one arithmetic or logic operation. As a result, we mainly use global memory access and shuffle instructions to estimate the time consumption. We perform our analysis for maximum delegate for simplicity and assume all global memory accesses have equal latency, i.e., C_{global}.

$T_{Delegate}$: Delegate vector construction reads $|V|$ elements and write $\frac{|V|}{2^\alpha}$ delegates. After thread local comparison, each subrange resorts to CUDA __shfl_sync instruction to derive the maximum for the entire subrange. Since one warp contains 32 threads, $\sum_{1 \le i \le 5} \frac{32}{2^i} = 31$ shuffle instructions are needed. Therefore, the communication complexity is $31 \cdot \frac{|V|}{2^\alpha} \cdot C_{shfl}$, where C_{shfl} is the cost of a shuffle instruction. Together, delegate vector construction time is:

$$T_{Delegate} = \underbrace{(1 + \frac{1}{2^\alpha}) \cdot |V| \cdot C_{global}}_{Global\ memory\ access} + \underbrace{\frac{31 \cdot |V|}{2^\alpha} \cdot C_{shfl}}_{Intra-warp\ comm.} \quad . \quad (2)$$

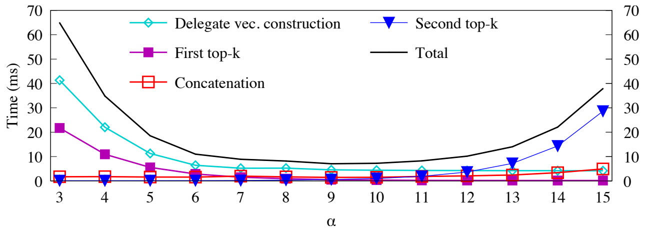


Figure 13: The runtime of *Dr. Top-k* with respect to the change of α, where $k = 2^{13}$ and the UD dataset from Section 6.

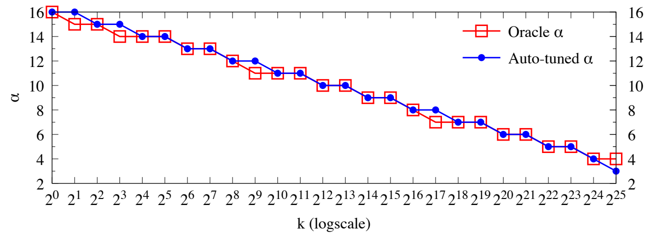


Figure 14: Performance of oracle α vs. auto-tuned α.

T_{FirstK}: Now, we analyze the time consumption of our optimized in-place radix top-k. According to our study, 8-bit per digit yields the optimal performance for in-place optimized radix top-k. A 32-bit unsigned integer hence experiences four iterations of scans. At each iteration, we always load all the elements. An additional iteration over the vector is used to identify the top-k elements. Finally, we write k elements, which are also the indices of the eligible subranges. Therefore, the time consumption of the first top-k is:

$$T_{FirstK} = \frac{5 \cdot |V| \cdot C_{global}}{2^{\alpha}} + 2 \cdot k \cdot C_{global}. \tag{3}$$

T_{Concat}: The concatenation step reads k indices for the subranges that are eligible for the second top-k and copies those subranges from the input to the concatenated vector for the second top-k. The time consumption for concatenation is:

$$T_{Concat} = k \cdot C_{global} + 2 \cdot k \cdot 2^{\alpha} \cdot C_{global}. \tag{4}$$

$T_{SecondK}$: The second top-k takes as input the output from the concatenation step and conducts in-place radix top-k to derive the eventual top-k. Consequently, this step is mainly about reading the entire outputs from concatenation. Similar to the analysis for the first top-k, which reads the concatenated vector by four times, the time consumption of the second top-k is:

$$T_{SecondK} = 4 \cdot k \cdot 2^{\alpha} \cdot C_{global}. \tag{5}$$

Taken Equations (2) – (5) together, we arrive at the total time consumption of ***Dr. Top-k*** as shown as in Equation 6.

$$\begin{aligned} T &= T_{Delegate} + T_{FirstK} + T_{Concat} + T_{SecondK} \\ &= 31 \cdot |V| \cdot 2^{-\alpha} \cdot C_{shfl} + \\ &(6 \cdot |V| \cdot 2^{-\alpha} + 6 \cdot k \cdot 2^{\alpha} + 2 \cdot k + |V|) \cdot C_{global}. \end{aligned} \tag{6}$$

Given Equation 6 ignores various hardware scheduling, arithmetic, and logical operation latency, we introduce $\Delta(\alpha, k, |V|)$ (at Equation 7), which is a positive function of α, k and $|V|$, to make up the impacts. We assume the magnitude of $\Delta(\alpha, k, |V|)$ is smaller than that of T.

$$T_{\textbf{\textit{Dr. Top-k}}} = T + \Delta(\alpha, k, |V|). \tag{7}$$

*We first prove $T_{\textbf{\textit{Dr. Top-k}}}$ is a convex function, which makes it easy to obtain the optimal α for **Dr. Top-k**.* According to [50], in order to demonstrate the convex nature of $T_{\textbf{\textit{Dr. Top-k}}}$, the second derivative of $T_{\textbf{\textit{Dr. Top-k}}}$ with respect to α should be positive.

$$\begin{aligned} \frac{\partial^2 T_{\textbf{\textit{Dr. Top-k}}}}{\partial^2 \alpha} &= (31 \cdot C_{shfl} + 6 \cdot C_{global}) \cdot |V| \cdot \ln^2(2) \cdot 2^{-\alpha} \\ &+ 6 \cdot k \cdot C_{global} \cdot \ln^2(2) \cdot 2^{\alpha} + \Delta''(\alpha, k, |V|). \end{aligned} \tag{8}$$

According to the assumption in Equation 7, the magnitude of $\Delta''(\alpha, k, |V|)$ will be smaller than the remaining factors in Equation 8. For positive values of k, $|V|$, C_{global} and C_{shfl}, we obtain:

$$\frac{\partial^2 T_{\textbf{\textit{Dr. Top-k}}}}{\partial^2 \alpha} > 0. \tag{9}$$

Hence, $T_{\textbf{\textit{Dr. Top-k}}}$ is convex function of α.

Our study in Figure 13 also suggests that ***Dr. Top-k*** is a convex function of α. Particularly, the time consumption of delegate vector construction and first top-k decrease along with the increase of α. Meanwhile, concatenation and second top-k increase. Altogether, the total time consumption decreases then increases with respect to the increase of α. Finally, since $T_{\textbf{\textit{Dr. Top-k}}}$ is convex, the optimal value of α can be obtained by:

$$\frac{\partial T_{\textbf{\textit{Dr. Top-k}}}}{\partial \alpha} = 0. \tag{10}$$

Solving Equation 10, we obtain:

$$\boxed{\alpha = \frac{1}{2} \cdot [\log_2(|V|) - \log_2(k) + const]}, \tag{11}$$

where, $const = \log_2(6 \cdot C_{global} + 31 \cdot C_{shfl}) - \log_2(6 \cdot C_{global}) + \Delta'(\alpha, k, |V|)$.

Figure 14, from an evaluation perspective, exhibits the performance alignment of the auto-tuned α and the oracle α across a wide range of k for the $|V| = 2^{30}$ unsigned integers dataset, where we set const = 3 according to performance tuning. □

5.3 Delegate Vector Construction Optimization

After we optimize the first and second top-k computations and concatenation steps, delegate vector construction becomes the next bottleneck for ***Dr. Top-k***. This is especially true when k is relatively large. According to Equations 11: α decreases with respect to the increase of k. For instance, when $|V|=2^{30}$, and $k = 2^{24}$, the optimal $\alpha = 4$. This implies that the input vector is partitioned into a large number of small subranges which would lead to two problems: *(i) the small subrange size fails to saturate a GPU warp; and (ii) too many subranges will lead to an overwhelming number of shuffle instructions for delegate communication.*

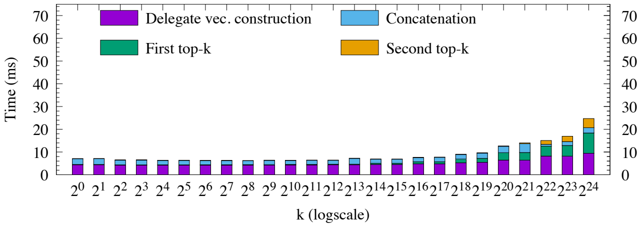


Figure 15: After delegate vector construction optimization, *Dr. Top-k* time consumption breakdown for UD dataset on Section 6.

We introduce a novel **coalesced loading to shared memory and strided computing approach** to remedy this small subrange size problem ($\alpha \leq 5$). This method consists of two phases: (i) one warp moves 32 subranges into the shared memory for delegate extraction. Here, each subrange is loaded from global memory into the shared memory by a warp in a coalesced manner. Since the subrange size is small, the shared memory pressure remains low. Subsequently, (ii) each thread of the warp individually works on the entire subrange to extract the delegate. This design ensures that all the threads of a warp have workloads, and no shuffle instruction is needed to communicate and decide the subrange delegates. This design helps the β delegate tremendously, which would otherwise needs approximately $\beta\times$ more shuffle instructions to extract the β delegates. We use padding to avoid shared memory bank conflict.

Figure 15 shows the improvement brought by the delegate vector construction optimization for different values of k. Comparing to Figure 10, one can find out that the delegate vector construction time is dramatically reduced for larger values of k, making the sampling time always close to merely the time consumption of scanning the input vector. Especially, for $k = 2^{24}$, we observe the time consumption of delegate vector construction decreases from 31.4 ms to 9.4 ms. And the total time consumption is reduced from 46.7 ms of Figure 10 to 24.7 ms.

5.4 Distributed *Dr. Top-k*

In the distributed GPU setting, we partition the input vector V into disjoint sub-vectors of equal length. To fit in the GPU memory, we require the length of each sub-vector to be no longer than 2^{30}: (i) when $(\#GPUs) \times 2^{30} \geq |V|$, we partition V into #GPU number of sub-vectors and let each GPU account for one sub-vector. (ii) When $(\#GPUs) \times 2^{30} < |V|$, we partition V into $\frac{|V|}{2^{30}}$ number of sub-vectors. In this case, one GPU accounts for more than one sub-vector; hence will load the unloaded sub-vectors from outside of GPUs. We schedule each GPU to compute the top-k for its own sub-vectors to arrive at one top-k per GPU. Subsequently, each GPU sends its top-k to the primary GPU to calculate the final top-k.

Figure 16 presents the workflow of multi-GPU ***Dr. Top-k*** which contains three major steps: ❶ It enables all the participating GPUs to work on their local sub-vectors to compute local top-k. ❷ It gathers these locally computed top-k's to the primary GPU. ❸ It enables primary GPU to compute the global top-k. For inter-GPU communication, we use Message Passing Interface (MPI) [43]. Particularly, we use MPI asynchronous (⟵ *Asynch. MPI* in the figure) communication among the processes to gather local top-k's from all the GPUs to the primary GPU.

While relying on the primary GPU to compute the final top-k works for a small number of GPUs, e.g., 16 in our evaluation, we anticipate hierarchical reduction [52] would excel when ***Dr. Top-k*** scales to a large number of GPUs. Particularly, for a multi-node setting, where each node installs multiple GPUs, the hierarchical scheduling method first derives the top-k across all the GPUs in each node. Afterward, all the nodes will send their top-k to the primary GPU to compute the final top-k.

It is also worthy of noting that we attempt to reduce the workload for the second top-k by enabling an MPI communication for the top-k^{th} element of the first top-k (the ↔ symbol in Figure 16). With the k^{th} delegate across all GPUs, we anticipate this will help filter out more unpromising elements hence reduce the workload. However, since this method requires all GPUs to have the maximum of k^{th} elements before launching the concatenation kernel, this introduces synchronization overhead. Additionally, in Figure 15, we also notice the cost of second top-k remains low throughout for a wide range of k, leaving relatively small room for improvement. In summary, the overhead of synchronizing the k^{th} elements from first top-ks' exceeds the benefits of a smaller concatenated vector, we disable this technique in our final version of distributed ***Dr. Top-k***.

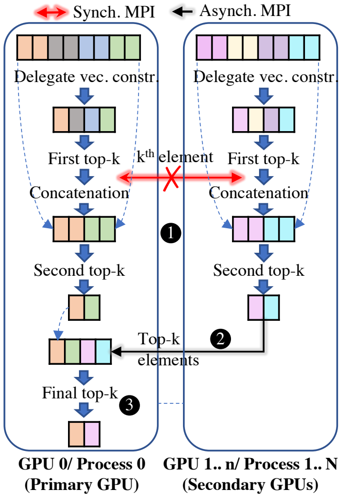


Figure 16: Workflow of multi-GPU *Dr. Top-k*.

6 EVALUATION

We implement ***Dr. Top-k*** with ~1,500 lines of C++ and CUDA code, extending the state-of-the-art bitonic, bucket and radix top-k projects [2, 42]. We compile the source code using NVIDIA CUDA 10.1 nvcc compiler with the optimization level as O3. We use two platforms to evaluate the performance of ***Dr. Top-k***. Platform I is a server with two Intel Xeon "Cascade Lake-SP" CPUs (@3.8 GHz) and 4 Tesla V100S GPU running Ubuntu Server 18.04. Platform II consists of i7-8700 CPU @ 3.20GHz with one Titan Xp running Ubuntu Server 16.04. All the reported execution time is an average of five runs. The default size of the input vector V is $|V| = 2^{30}$, and each data entry is an unsigned integer. V is generated by the following distributions.

- **Uniform distribution dataset (UD)** is generated following U[0, 2^{32}-1], meaning the value ranges from 0 to $2^{32} - 1$.

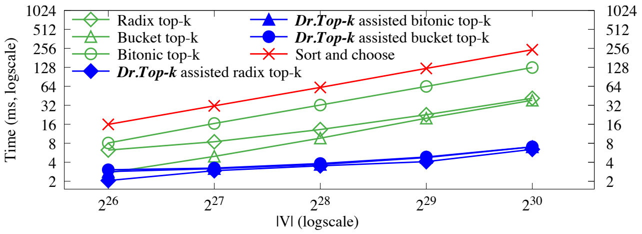


Figure 17: The time consumption of *Dr. Top-k* versus various top-k algorithms with respect to the increase of $|V|$.

- **Normal distribution dataset (ND)** is produced with the normal distribution $N[10^8, 10]$, where the mean and standard deviation are 10^8 and 10, respectively.
- **Customized distribution dataset (CD)** is produced to increase the number of iterations in bucket top-k. The values are generated in the range of [0, 2^{32}-1] such that every bucket other than the bucket containing the k^{th} element will always have at least one element in every iteration and majority of the element is present in the bucket with the k^{th} element.

Unless stated differently, we present the experiments on platform I for the UD dataset.

6.1 *Dr. Top-k* vs. State-of-the-art

This section reports the performance gains brought by ***Dr. Top-k*** to the state-of-the-art with respect to the change of $|V|$ and k.

Dr. Top-k* for different input vector V sizes.** Figure 17 demonstrates the time consumption of different versions of top-k for $k = 1024$ on V whose sizes vary from 2^{26} to 2^{30}. *The general trend is that **Dr. Top-k** becomes more beneficial when the input vector size is bigger*, because delegate vectors can help reduce more workloads when V gets larger. Particularly, when $|V| = 2^{30}$, radix, bucket, bitonic and sort and choose top-k consume 41.3 ms, 38.4 ms, 127.0 ms and 243.2 ms, respectively. Our ***Dr. Top-k assisted radix, bucket and bitonic top-k designs reduce the time to 6.4 ms, 7.0 ms and 7.0 ms, respectively.

Dr. Top-k* assisted radix top-k.** As shown in Figure 18, in general, ***Dr. Top-k yields bigger performance gains on the normal and customized distribution datasets. Particularly, we observe 1.7× - 10× and 1.1× - 10.1× speedups, respectively on normal and customized distribution, while 1.7× - 6.6× on uniform distribution. It is also important to note that the impact of ***Dr. Top-k*** decreases with respect to the increase of k. For instance, when k=2^{24}, ***Dr. Top-k*** only gives 1.7× speedups for both the uniform and normal distribution datasets and 1.1× speedup for customized distribution. This is caused by the fact that ***Dr. Top-k*** requires more delegates to figure out the useful subranges when k becomes larger, leading the first top-k to consume significant time. Section 6.2 conducts a thorough study on the workload reduction trend for varying k.

Dr. Top-k* assisted bucket top-k.** Figure 18 also shows the speedup of ***Dr. Top-k assisted bucket top-k over the bucket top-k alone algorithm on the normal, customized and uniform distributions. The trends are analogous to radix top-k but with two differences. First, bucket top-k performs fairly well when $k = 1$ because bucket top-k first finds the maximum value. Then the query is completed. Thanks to near bandwidth performance of delegate vector construction and single delegate needed to be selected for first top-k for $k = 1$, ***Dr. Top-k*** assisted bucket top-k only needs to work on the first top-k, leading ***Dr. Top-k*** assisted bucket top-k to perform faster, i.e., by 1.1× for all normal, customized and uniform distributions. Second, in bucket top-k, the speedup of ***Dr. Top-k*** on customized distribution outpaces the uniform and normal distribution. Particularly, we observe the speedups from 1.1× - 118.6× for customized distribution while 1.1× - 6.1× and 1.1× - 6.2× for normal and uniform distribution respectively.

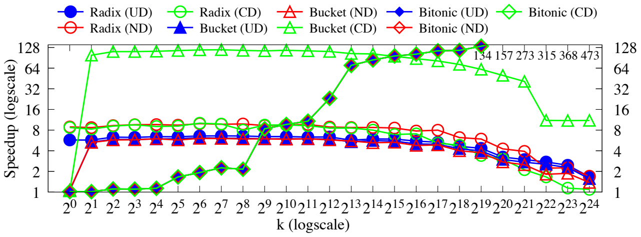


Figure 18: The speedup of *Dr. Top-k* over the-state-of-the-art for a varying k on synthetic datasets.

Dr. Top-k* assisted bitonic top-k.** Figure 18 further includes the speedup of ***Dr. Top-k assisted bitonic top-k over the bitonic top-k stand alone algorithm [42]. Note, the original source code [5] from bitonic top-k project [42] experiences shared memory overflow when k goes beyond 256. We modify the source code to enable it for $k > 256$. Particularly, the speedup of ***Dr. Top-k*** climbs from 1.1× when k=2^0 to 473× when k=2^{24}. Since the performance of bitonic top-k is independent from the data distribution, the speedups over the normal, uniform and customized distributions are the same. Note, for visualization, we limit the y-axis to [0, 128] in Figure 18. Hence the speedups that are beyond 128 for bitonic top-k are marked as numbers in Figure 18.

Dataset	Abbr.	|V|	Application domain
ANN_SIFT1B [21]	AN	536,870,912	k-Nearest Neighbor
ClueWeb09 [9, 40]	CW	1,073,741,824	Sparse Networks
TwitterCOVID-19 [19]	TR	1,073,741,824	Social Networks

Table 1: Real-world datasets.

Real-world datasets contains three datasets: ANN_SIFT1B [21], ClueWeb09 [39] and TwitterCOVID-19 [19]. (i) ANN_SIFT1B dataset contains 1 billion vectors, each of which is at 128 dimensions and describes an image. We use the first vector from the ANN_SIFT1B dataset to calculate the euclidean distances between this vector and the 1 billion vectors. Afterward, the distance array is the input vector for top-k. (ii) The ClueWeb09 is a webpage graph which contains 4,780,950,910 webpages and 7,939,635,651 links. We derive the degrees of the webpages and use that as the input vector for top-k. (iii) TwitterCOVID-19 [19] dataset consists of COVID-fear related scores of the tweets related to the COVID-19 pandemic from 28 January 2020 to 1 January 2021. The original 132 million public twitter posts are duplicated on the vector of 1 billion size to achieve same distribution. *Top-k computation can help (i) derive the k-NN of*

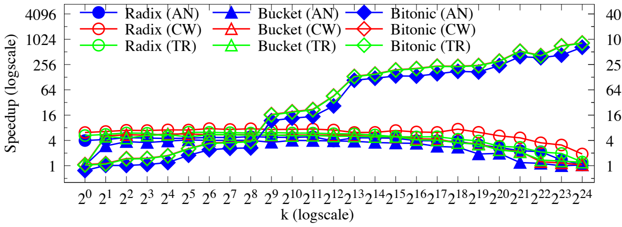


Figure 19: The speedup of *Dr. Top-k* over the-state-of-the-art for a varying k on real-world graph.

a query vector [21], (ii) rank the vertices by degree [12] and (iii) k least fearful tweets related to the COVID19 pandemic in [19] dataset. Since bitonic top-k cannot work on $|V|$ that is not at size of power of 2, and GGKS radix top-k suffers from $|V| \geq 2^{31}$, we cut the sizes of (i) and (ii) datasets into 536,870,912 and 1,073,741,824, respectively.

Figure 19 shows the speedup of ***Dr. Top-k*** assisted top-k algorithms over the state-of-the-art projects on the real-world datasets. In general, for the same top-k algorithm, ***Dr. Top-k*** enjoys higher speedups on CW dataset than AN, because CW is larger. This aligns with our finding in Figure 20. On average, ***Dr. Top-k*** assisted radix, bucket and bitonic top-k, respectively, perform 6.7×, 4.6 × and 173.7× faster than their corresponding top-k algorithms on CW. AN dataset observes an average speedup of respectively 4.2×, 3.3× and 127.1× over the state-of-the-art top-k algorithms. Similarly, TR dataset observes an average speedup of respectively 4.8×, 4.1× and 170.2× over the state-of-the-art top-k algorithms.

6.2 *Dr. Top-k* Workload Statistics

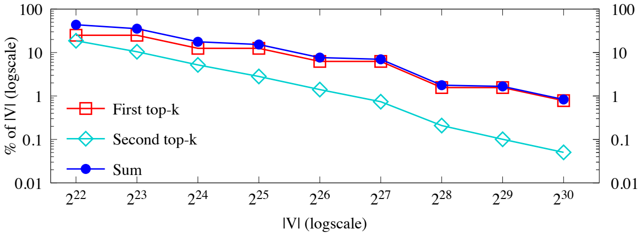


Figure 20: Workload dynamics of the first top-k, second top-k and their sum with respect to the increase of $|V|$. Here, we set $k = 2^{19}$.

Figure 20 plots the workload dynamics for the first top-k, second top-k and their sum with respect to varying sizes of the input vector $|V|$. Particularly, the workloads are the sizes of the delegate vector and the concatenated vector for the first and second top-k, respectively. *We observe that the ratio of the delegate vector over $|V|$ decreases significantly when $|V|$ increases, so does that for concatenated vector.* Specifically, the sum of the delegate and concatenated vector sizes is 76.06% of $|V|$ at $|V| = 2^{22}$ and 0.83% at $|V| = 2^{30}$. *This workload reduction trend demonstrates the scalable nature of **Dr. Top-k**, that is, **Dr. Top-k** performance improves when the problem size |V| increases.*

Figure 21 demonstrates the vector size of the first top-k and second top-k in ***Dr. Top-k*** assisted radix top-k across different k values. Apparently, for a given input vector size, the increase of k will lead to larger vector sizes for both the first and second top-k. As the vector sizes increase to a higher ratio of the input vector, the speedup of ***Dr. Top-k*** over the state-of-the-art also decreases. Another fact is that the workload of the first top-k dominates the entire workload for ***Dr. Top-k*** because β delegate will lead to more delegates. Further, β delegate and delegate-based filtering together can significantly reduce the workload for the second top-k. Particularly, the ratio of the sum of both vectors over the input vector climbs from 0.0015% to 15.91% with the increase of k.

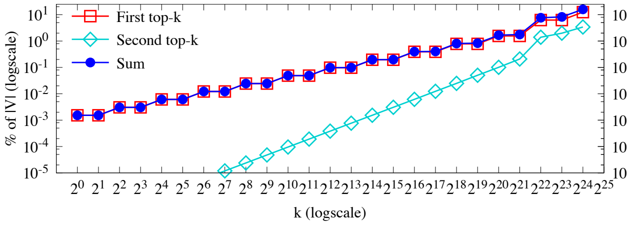


Figure 21: Workload dynamics of the first top-k, second top-k and their sum with respect to the increase of k. Here, we set $|V| = 2^{30}$.

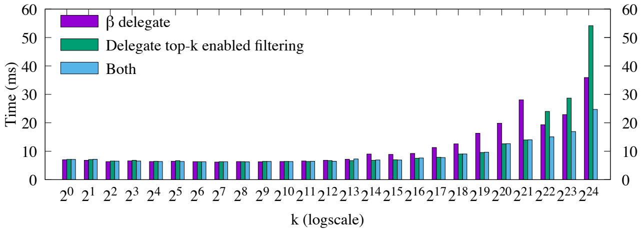


Figure 22: The performance impacts of delegate top-k enabled filtering vs β delegate.

Figure 22 studies the separate and combined effects of delegate top-k based filtering and β delegate, given both of them are proposed to reduce the workload for concatenation and second top-k. Here, we include delegate vector construction optimization. When k is small, one can observe that delegate top-k enabled filtering yields better performance gains over β delegate, $k = 2^{20}$ in particular. However, once k becomes bigger, the β delegate optimization starts picking up the momentum. Overall, delegate top-k enabled filtering combined with β delegate always offers the best performance. Particularly, for $k = 2^{24}$, the time consumption of delegate top-k enabled filtering, β delegate and the combined one are 54.2 ms, 35.9 ms 24.7 ms.

6.3 *Dr. Top-k* Scalability

Table 2 demonstrates the scalability of ***Dr. Top-k*** assisted radix top-k for vector sizes |V| of 2^{30} - 2^{33} on up to 16 V100 GPUs (4 compute nodes). The table includes the communication overhead among the GPUs, vector reloading overhead, and total time. Overall, we can observe that ***Dr. Top-k*** achieves desirable scalability in various settings. When the partitioned sub-vector can fit in 1 - 16 GPUs in $|V| = 2^{30}$, the speedup goes up to 3.4× on 16 GPUs. In the remaining columns ($|V| \geq 2^{31}$) of the table, we observe superlinear speedup. The reason is that when the # of GPUs is low, we cannot fit all

#GPU (#Nodes)	$\|V\| = 2^{30}$			$\|V\| = 2^{31}$			$\|V\| = 2^{32}$			$\|V\| = 2^{33}$		
	Communication (ms)	Reload Overhead (ms)	Total time (ms) (speedup)	Communication (ms)	Reload Overhead (ms)	Total time (ms) (speedup)	Communication (ms)	Reload Overhead (ms)	Total time (ms) (speedup)	Communication (ms)	Reload Overhead (ms)	Total time (ms) (speedup)
1 (1)	0	0	6.1 (1x)	0	373.14	384.93 (1x)	0	1238.13	1261.51 (1x)	0	2898.54	2944.99 (1x)
2 (1)	0.11	0	3.7 (1.6x)	0.46	0	6.22 (61.7x)	0.06	524.41	536.218 (2.3x)	0.08	1586.81	1788.3 (1.7x)
4 (1)	0.11	0	2.5 (2.4x)	0.29	0	3.7 (104.0x)	0.12	0	8.8 (143.3x)	0.07	1056.02	1067.68 (2.8x)
8 (2)	0.19	0	1.96 (3.1x)	0.29	0	2.71 (141.7x)	0.73	0	4.36 (289.2x)	1.32	0	7.97 (369.4x)
16 (4)	0.31	0	1.80 (3.4x)	0.32	0	2.07 (185.9x)	0.82	0	2.68 (470.5x)	1.43	0	4.01 (734.2x)

Table 2: Scalability of *Dr. Top-k* with varying $|V|$ and $k = 128$.

the sub-vectors in GPU before computation. Therefore, ***Dr. Top-k*** loads certain partitions during computation. And the total time includes sub-vector loading time. Whereas, when the GPU number increases to 16, the data can fit in GPUs. Thanks to a relatively low communication cost in asynchronous communication for the top-k elements, we observe a maximum communication time of 1.43 ms at 16 GPUs $|V| = 2^{33}$ configuration. Similarly, for a large input vector $|V| = 2^{33}$ and with a single GPU configuration, the reload overload can go up to 2898.54 ms. Note, we cannot include the multi-GPU settings for the state-of-the-art tools because they do not support multi-GPU features.

6.4 Global Memory Transactions Analysis

#global memory transactions	Radix top-k		Bucket top-k		Bitonic top-k	
	GGKS [2]	Dr.Top-k	GGKS [2]	Dr.Top-k	Bitonic [42]	Dr.Top-k
#load ($\times 10^9$)	3.07	1.34	4.04	1.44	11.45	1.35
#store ($\times 10^9$)	2.01	0.003	1.36	0.003	2.09	0.007

Table 3: Number of global load and store transactions in different versions in top-k. We test on UD dataset with $|V| = 2^{30}$ and $k = 2^7$.

Table 3 showcases the number of global memory load and store transactions of different versions of top-k on the UD dataset with $|V| = 2^{30}$ and $k = 2^7$. We use nvprof [29] profiler for profiling the results. From the table, we observe a reduction of load transactions by 2.3×, 3.1× and 8.5×, respectively when ***Dr. Top-k*** assists radix, bucket and bitonic top-k. Similary, ***Dr. Top-k*** helps reduce the global memory store transaction by 766.8×, 516.9× and 298.6×, respectively on radix, bucket and bitonic top-k.

6.5 *Dr. Top-k* on Different GPUs

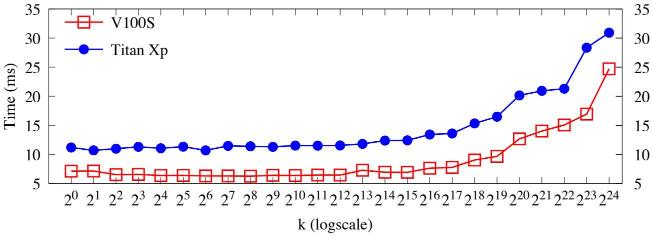


Figure 23: *Dr. Top-k* on V100S Vs Titan Xp.

Figure 23 compares the ***Dr. Top-k*** radix top-k on the V100S and Titan Xp GPUs. Clearly, the time consumption patterns of ***Dr. Top-k*** on both the GPUs are similar for a range of k. Overall, the performance of ***Dr. Top-k*** on V100S is better than in Titan Xp by a factor of 1.3× - 1.8×. This roughly aligns with the ratio of the reported peak throughput difference between V100S [31] and Titan Xp [4] which are 1,134 GB/s and 547.7 GB/s.

6.6 *Dr. Top-k* vs BMW

Figure 24 presents the ratio $= \frac{\text{BMW workload}}{\textbf{\textit{Dr. Top-k}}\text{ workload}}$, where we use the sum the workloads of first and second top-k as ***Dr. Top-k*** workload. Overall, we observe the ratio to be, on average, 212× in ND and 6× in UD, which suggests that ***Dr. Top-k*** reduces 212× and 6× more workload than BMW. The reason is that BMW still works on each regular item even after deriving the delegate while ***Dr. Top-k*** uses a delegate to skip an entire subrange directly.

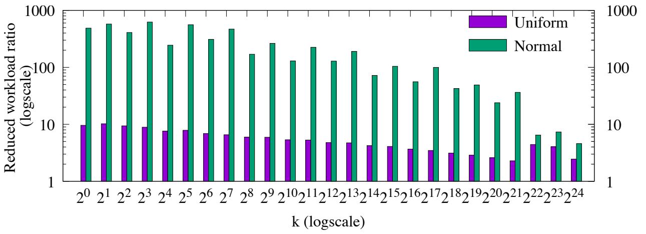


Figure 24: The ratio of fully evaluated workload (after workload reduction) of BMW to that of *Dr. Top-k*.

7 CONCLUSION

We introduce ***Dr. Top-k*** with three contributions: First and foremost, ***Dr. Top-k*** introduces a comprehensive delegate-centric concept to help tremendously reduce the workload for top-k computations. Second, we introduce a practical way to partition the input vector into proper sized subranges with theoretical support. Finally, we deploy our project atop distributed GPUs to handle extreme large input vectors along with various system optimizations. Taken together, ***Dr. Top-k*** assisted top-k algorithms constantly outperform the state-of-the-art.

ACKNOWLEDGEMENT

This work was in part supported by NSF CRII Award No. 2000722, CAREER Award No. 2046102, Australia Research Council (ARC) Discovery Project DP210101984, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration (NNSA). Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE, NNSA, ARC, or NSF.

REFERENCES

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M. Tensorflow: A System for Large-scale Machine Learning. In *Symposium on Operating Systems Design and Implementation* (2016).

[2] Alabi, T., Blanchard, J. D., Gordon, B., and Steinbach, R. Fast k-Selection Algorithms for Graphics Processing Units. *Journal of Experimental Algorithmics 17* (2012), 4–2.

[3] Amazon AWS. U.S. Department of Energy and Cray to Deliver Record-Setting Frontier Supercomputer at ORNL. Available at https://www.ornl.gov/news/us-department-energy-and-cray-deliver-record-setting-frontier-supercomputer-ornl. Accessed: 2020, June 15.

[4] Angelini, C. Nvidia Titan Xp. Available at https://www.tomshardware.com/reviews/nvidia-titan-xp,5066.html. Accessed: 2020, March 15.

[5] Anil Shanbhag. Bitonic select source code. Available at https://github.com/anilshanbhag/gpu-topk. Accessed: 2020, July 16.

[6] Bell, N., and Hoberock, J. Thrust: A Productivity-Oriented Library for CUDA. In *GPU computing gems Jade edition*. Elsevier, 2012, pp. 359–371.

[7] Bress, S., Köcher, B., Heimel, M., Markl, V., Saecker, M., and Saake, G. Ocelot/HyPE: Optimized Data Processing on Heterogeneous Hardware. *Proceedings of the VLDB Endowment 7*, 13 (2014), 1609–1612.

[8] Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. Efficient Query Evaluation Using a Two-level Retrieval Process. In *International Conference on Information and Knowledge Management* (2003), ACM, pp. 426–434.

[9] Clarke, C. L., Craswell, N., and Soboroff, I. Overview of the TREC 2009 Web Track. Tech. rep., DTIC Document, 2009.

[10] Dehne, F., and Zaboli, H. Parallel Sorting for GPUs. In *Emergent Computation*. Springer, 2017, pp. 293–302.

[11] Ding, S., and Suel, T. Faster Top-k Document Retrieval Using Block-Max Indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2011), pp. 993–1002.

[12] Doo, M., and Liu, L. Extracting Top-k Most Influential Nodes by Activity Analysis. In *International Conference on Information Reuse and Integration* (2014), IEEE, pp. 227–236.

[13] Gaihre, A., Liu, H., and Li, X. GSOFA: Scalable Sparse LU Symbolic Factorization on GPUs. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[14] Gaihre, A., Luo, Y., and Liu, H. Do Bitcoin Users Really Care About Anonymity? An Analysis of the Bitcoin Transaction Graph. In *International Conference on Big Data (Big Data)* (2018), IEEE, pp. 1198–1207.

[15] Gaihre, A., Pandey, S., and Liu, H. Deanonymizing Cryptocurrency with Graph Learning: The Promises and Challenges. In *Conference on Communications and Network Security (CNS)* (2019), IEEE, pp. 1–3.

[16] Gaihre, A., Wu, Z., Yao, F., and Liu, H. XBFS: eXploring Runtime Optimizations for Breadth-First Search on GPUs. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (2019), ACM.

[17] Gokhale, M., Cohen, J., Yoo, A., Miller, W. M., Jacob, A., Ulmer, C., and Pearce, R. Hardware Technologies for High-performance Data-intensive Computing. *Computer 41*, 4 (2008), 60–68.

[18] Guo, C., Chen, H., Li, C., and Wu, T. A Memory Access Reduced Sort on Multi-core GPU. In *International Conference on High Performance Computing and Communications; International Conference on Smart City; International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (2018), IEEE, pp. 586–593.

[19] Gupta, R. K., Vishwanath, A., and Yang, Y. Global Reactions to COVID-19 on Twitter: A Labelled Dataset with Latent Topic, Sentiment and Emotion Attributes. *arXiv preprint arXiv:2007.06954* (2020).

[20] Ionescu, M. F., and Schauser, K. E. Optimizing Parallel Bitonic Sort. In *Proceedings 11th International Parallel Processing Symposium* (1997), IEEE, pp. 303–309.

[21] Jégou, H., Douze, M., and Schmid, C. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011).

[22] Johnson, J., Douze, M., and Jégou, H. Billion-scale Similarity Search with GPUs. *Transactions on Big Data* (2019).

[23] Li, A., Liu, W., Wang, L., Barker, K., and Song, S. L. Warp-consolidation: A Novel Execution Model for GPUs. In *International Conference on Supercomputing* (2018).

[24] Liu, Y., Wang, J., and Swanson, S. Griffin: Uniting CPU and GPU in Information Retrieval Systems for Intra-Query Parallelism. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2018).

[25] Malkov, Y. A., and Yashunin, D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence 42*, 4 (2018), 824–836.

[26] Muneer. arrayfirerequest. Available at https://groups.google.com/g/arrayfire-users/c/oDtQcI7afZQ/. Accessed: 2021, Mar 17.

[27] Nguyen, H. T. H., and Cao, J. Trustworthy Answers for Top-k Queries on Uncertain Big Data in Decision Making. *Information Sciences 318* (2015), 73–90.

[28] NVIDIA. NVIDIA A100 TENSOR CORE GPU. Available at https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/a100-80gb-datasheet-update-nvidia-us-1521051-r2-web.pdf. Accessed: 2021, Jan 31.

[29] Nvidia Inc. Nvidia Profiler nvprof. Available at https://docs.nvidia.com/cuda/profiler-users-guide/index.html. Accessed: 2020, July 16.

[30] Nvidia Inc. NVIDIA TESLA V100 GPU Architecture Whitepaper. Available at https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf. Accessed: 2020, July 16.

[31] Nvidia Inc. NVIDIA V100 TENSOR CORE GPU. Available at https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf. Accessed: 2020, July 16.

[32] Oak Ridge National Laboratory . AmericaâĂŹs First Exascale Supercomputer to be Built by 2021. Available at https://www.olcf.ornl.gov/frontier/. Accessed: 2021, Jan 31.

[33] Obeya, O., Kahssay, E., Fan, E., and Shun, J. Theoretically-efficient and Practical Parallel In-place Radix Sorting. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures* (2019), pp. 213–224.

[34] Pandey, S., Li, L., Hoisie, A., Li, X. S., and Liu, H. C-SAW: A Framework For Graph Sampling and Random Walk on GPUs. In *International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), IEEE.

[35] PG-Strom. PG-Strom v3.0 Official Documentation. Available at https://heterodb.github.io/pg-strom/release_v3.0/. Accessed: 2021, May 24.

[36] Ren, J., Zhang, M., and Li, D. HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory. *Advances in Neural Information Processing Systems 33* (2020).

[37] Ribizel, T., and Anzt, H. Parallel Selection on GPUs. *Parallel Computing 91* (2020), 102588.

[38] Root, C., and Mostak, T. MapD: A GPU-powered Big Data Analytics and Visualization Platform. In *ACM SIGGRAPH 2016 Talks*. Association for Computing Machinery, 2016, pp. 1–2.

[39] Rossi, R. A., and Ahmed, N. K. WEB-CLUEWEB09. Available at http://networkrepository.com/web-ClueWeb09.php. Accessed: 2021, Feb 4.

[40] Rossi, R. A., and Ahmed, N. K. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015).

[41] Sebastian Moss. El Capitan Supercomputer to Feature AMD Chips, Break 2 Exaflops Barrier. Available at https://www.datacenterdynamics.com/en/news/el-capitan-supercomputer-feature-amd-chips-break-2-exaflops-barrier/. Accessed: 2021, Jan 31.

[42] Shanbhag, A., Pirk, H., and Madden, S. Efficient Top-K Query Processing on Massively Parallel Hardware. In *Proceedings of the 2018 International Conference on Management of Data* (2018), ACM, pp. 1557–1570.

[43] Snir, M., Gropp, W., Otto, S., Huss-Lederman, S., Dongarra, J., and Walker, D. *MPI–the Complete Reference: the MPI core*, vol. 1. MIT press, 1998.

[44] Stehle, E., and Jacobsen, H.-A. A Memory Bandwidth-Efficient Hybrid Radix Sort on GPUs. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), ACM, pp. 417–432.

[45] Sverdlik, Y. IBM, Nvidia Build "World's Fastest Supercomputer" for US Government. Available at https://www.datacenterknowledge.com/supercomputers/ibm-nvidia-build-world-s-fastest-supercomputer-us-government. Accessed: 2020, April 1.

[46] The Verge. Americas First Exascale Supercomputer to be Built by 2021. Available at https://www.theverge.com/2019/3/18/18271328/supercomputer-build-date-exascale-intel-argonne-national-laboratory-energy. Accessed: 2021, Jan 31.

[47] Vetter, J. S., Brightwell, R., Gokhale, M., McCormick, P., Ross, R., Shalf, J., Antypas, K., Donofrio, D., Humble, T., and Schuman, C. Extreme Heterogeneity 2018-Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity. Tech. rep., USDOE Office of Science Washington, DC (United States), 2018.

[48] Wang, L., Wu, W., Zhang, J., Liu, H., Bosilca, G., Herlihy, M., and Fonseca, R. FFT-based Gradient Sparsification for the Distributed Training of Deep Neural Networks. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (2020), pp. 113–124.

[49] Wang, S., Liu, H., Gaihre, A., and Yu, H. EZLDA: Efficient and Scalable LDA on GPUs. *arXiv preprint arXiv:2007.08725* (2020).

[50] Weisstein, E. W. Second Derivative Test. Available at https://mathworld.wolfram.com/SecondDerivativeTest.html. Accessed: 2021, Aug 26.

[51] Xueyuan Zhao. Tensorflow. Available at CalculatetopKonGPUwithk-selectionalgos/. Accessed: 2021, Mar 17.

[52] Yang, C. Tree-based Allreduce Communication on MXNet. *Tech. Rep.* (2018).

[53] Zhao, W., Tan, S., and Li, P. SONG: Approximate Nearest Neighbor Search on GPU. In *International Conference on Data Engineering (ICDE)* (2020), IEEE, pp. 1033–1044.

[54] Zheng, B., Xi, Z., Weng, L., Hung, N. Q. V., Liu, H., and Jensen, C. S. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proceedings of the VLDB Endowment 13*, 5 (2020), 643–655.

[55] Zois, V., Tsotras, V. J., and Najjar, W. A. Efficient Main-memory Top-K Selection for Multicore Architectures. *VLDB Endowment 13*, 12 (2019).