Supplementary Material

A *iQAN* Algorithm

Algorithm 2: *iQAN*: Intra-Query Parallel ANNS

```
Input: graph $G$, starting point $P$, query $Q$, queue capacity
       $L$, number of workers $T$
Output: $K$ nearest neighbors of $Q$
1  expansion width $W \leftarrow 1$
2  global priority queue $S \leftarrow$ an empty queue
3  local priority queues $LS \leftarrow T$ empty queues
4  array $U \leftarrow$ the vector to store update positions of workers
5  compute $dist(P, Q)$ and add $P$ into $S$
6  while true do
7      divide all unchecked vertices from $S$ into $LS$
8      if all $LS$ are empty then break
9      foreach worker $t$ out of $W$ in parallel do
10         while $LS[t]$ has unchecked vertices and doMerge is
             false do
11             $v \leftarrow$ the first unchecked vertex in $LS[t]$
12             mark $v$ as checked
13             foreach neighbor $u$ of $v$ in $G$ do
14                 if $u$ is not visited then
15                     mark $u$ as visited
16                     compute $dist(u, Q)$
17                     add $u$ into $LS[t]$
18             if $LS[t]$.size() > $L$ then $LS[t]$.resize($L$)
19             update $U[t]$
20             if $t$ is the checker then
21                 $\bar{u} \leftarrow$ average positions of elements in $U$
22                 if $\bar{u} \geq L \cdot R$ then doMerge $\leftarrow$ true
23                 else doMerge $\leftarrow$ false
24                 assign the next checker in a round-robin
                     way
25     merge $LS$ into $S$
26     if $S$.size() > $L$ then $S$.resize($L$)
27     if $W < T$ then $W \leftarrow 2W$
28 return the first $K$ vertices in $S$
```

Algorithm 2 describes the overall algorithm of *iQAN*. At the beginning of each global step, the global queue evenly divides its unchecked candidates among all workers. After that, each worker performs a local best-first search based on its own local queue (Line 10 to Line 24). In a local search step, a worker expands its best-unchecked candidate and updates its private queue accordingly. A worker continues expansion until the *checker* indicates a sync or it has no unchecked candidates left locally. In the path-wise parallelism, the visiting map is shared by all workers to indicate if a vertex has been visited. Since multiple threads may access the shared visiting map concurrently, locking or lock-free algorithms are required to ensure a vertex is visited only once. Fortunately, the ANNS algorithm remains correct even if a vertex is calculated multiple times because the local candidates are guaranteed to be merged back to the global priority queue. Therefore, multiple threads do not need to exclusively update the visiting map, which is then called a *loosely synchronized visiting map* and can reduce communication overhead. It is implemented using a bit vector instead of a byte array for faster access.

B Details of Datasets

Table 2. Characterization of datasets. `Dim.` denotes the dimension of a feature vector, `#base` denotes the number of points, and `#queries` denotes the number of queries.

Dataset	Dim.	#base	#queries
SIFT1M	128	1M	10K
GIST1M	960	1M	1K
DEEP10M	96	10M	10K
SIFT100M	128	100M	10K
DEEP100M	96	100M	10K
BIGANN	128	1B	10K
DEEP1B	96	1B	10K

Table 2 summarizes the details of the datasets we use in the evaluation.

C Latency Speedups on Skylake

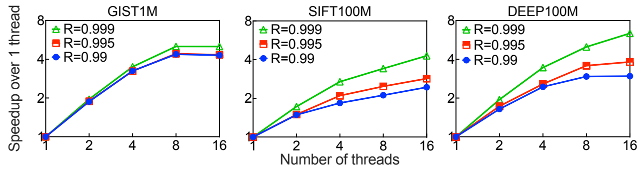

Figure 18. Speedups of *iQAN* over the sequential baseline on Skylake.

Figure 18 shows that *iQAN* on Skylake achieves average 4.4× and 5.2× speedups over the sequential baseline with 8- and 16-thread at recall 0.999, respectively.

D Comparison with a GPU Implementation.

Table 3. Latency comparison of *iQAN* and Faiss-GPU on five datasets. `Lt.` means *Latency*. `OOM` means *out of memory*. Faiss-GPU's index format is IVFFLat. *iQAN* uses 32 threads.

Datasets	Faiss-GPU w/ IVFFlat		*iQAN*-32T on KNL	
	R@100	Lt. (ms.)	R@100	Lt. (ms.)
SIFT1M	0.52	0.87	**0.91**	**0.61**
GIST1M	0.36	7.25	**0.90**	**1.21**
DEEP10M	0.62	5.79	**0.90**	**0.96**
SIFT100M	OOM	OOM	**0.90**	**2.00**
DEEP100M	OOM	OOM	**0.90**	**1.91**

We also compare *iQAN* with a GPU-based large-scale ANNS algorithm [30] in the Faiss library [1]. The GPU experiments are conducted on an NVIDIA Tesla P100 with CUDA 10.2. Faiss is set to have one query in every batch because we focus on reducing the online query latency to meet stringent latency requirements. Table 3 shows the latency comparison results on five datasets. *iQAN* uses 32 threads on KNL. For the SIFT100M and DEEP100M, Faiss-GPU complains of out-of-memory errors. For other datasets, *iQAN* outperforms Faiss-GPU with 1.4× to 6.0× speedup and much better recall, which indicates that *iQAN* can effectively achieve faster latency on CPUs than GPU-based algorithms.

E Effects on memory bandwidth.

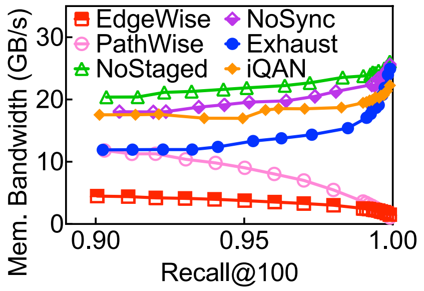

Figure 19. Memory bandwidth on KNL using 32 threads.

Figure 19 shows that *iQAN* achieves a much higher memory bandwidth utilization than *EdgeWise*. For example, the memory bandwidth increases from 1.6 GB/s to 21.1 GB/s at recall 0.999. This bandwidth improvement comes from *iQAN*'s design to expose more parallelism during neighbor expansion while carefully eliminating synchronization overhead and redundant computations. To see how each optimization affects the memory bandwidth utilization, we also include the results of intermediate configurations. *PathWise* overall achieves better memory bandwidth utilization than *EdgeWise*. This is because it exposes more parallelism during the neighbor expansion process. However, as the recall target increases, the memory bandwidth of *Pathwise* declines gradually towards *EdgeWise*. This is because *PathWise* does not apply redundancy-aware synchronization and still merges local candidates to the global priority queue in every iteration. As a result, the synchronization overhead becomes a dominant factor and limits the memory bandwidth utilization when the search is under the high recall region. *NoStaged*, *NoSync*, *Exhaust*, and *iQAN* all improve the memory bandwidth utilization considerably because they all reduce frequent synchronizations. *NoStaged* and *NoSync* have slightly higher memory bandwidth utilization than *iQAN* because *NoStaged* and *NoSync* either use the maximum expansion width during the entire search process or perform no synchronization across worker threads, which leads to more (redundant) data loads during neighbor expansion. Therefore, *iQAN* results in shorter query latency than *NoStaged* and *NoSync* even though it requires slightly less memory bandwidth utilization.