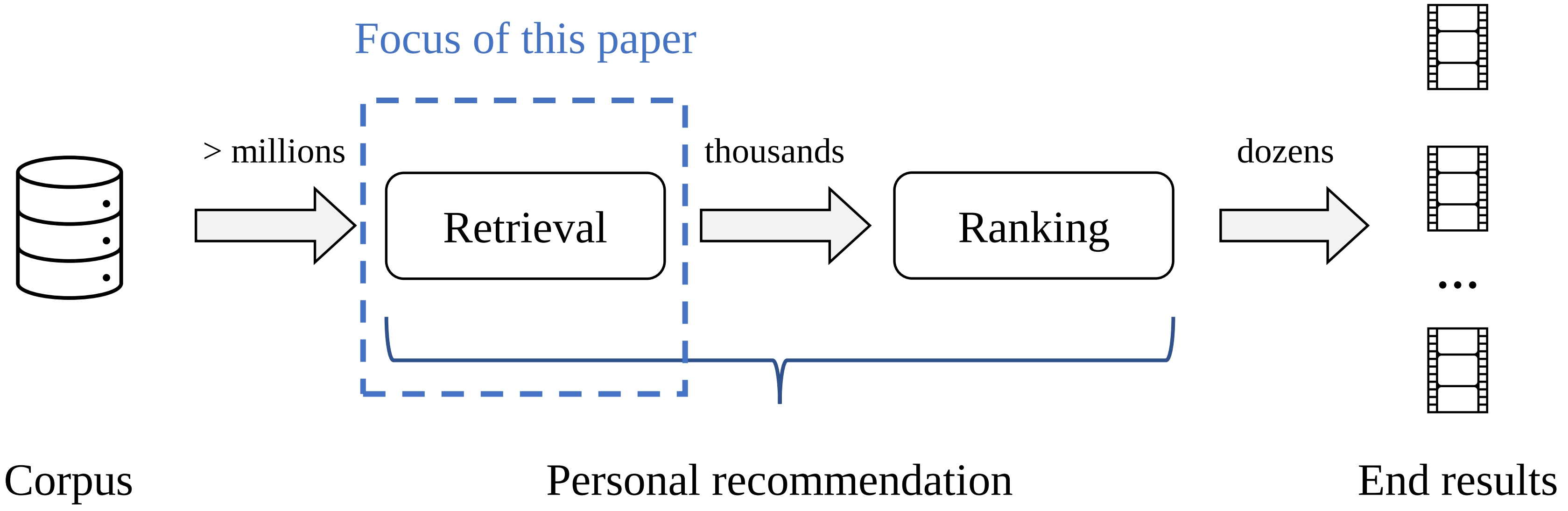
FAERY : An FPGA-accelerated Embedding-based Retrieval System

Chaoliang Zeng, Layong Luo, Qingsong Ning, Yaodong Han, Yuhang Jiang, Ding Tang, Zilong Wang, Kai Chen, Chuanxiong Guo

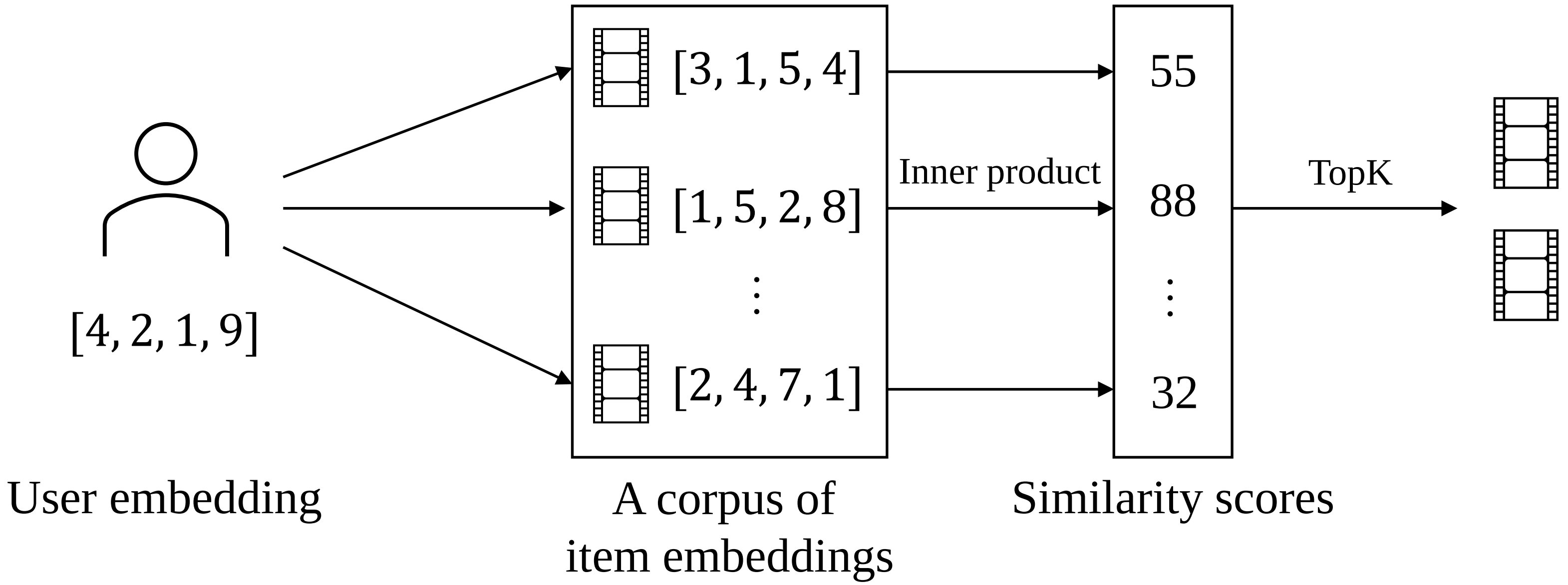
Hong Kong University of Science and Technology

ByteDance

Recommendation System



Embedding-based Retrieval (EBR)

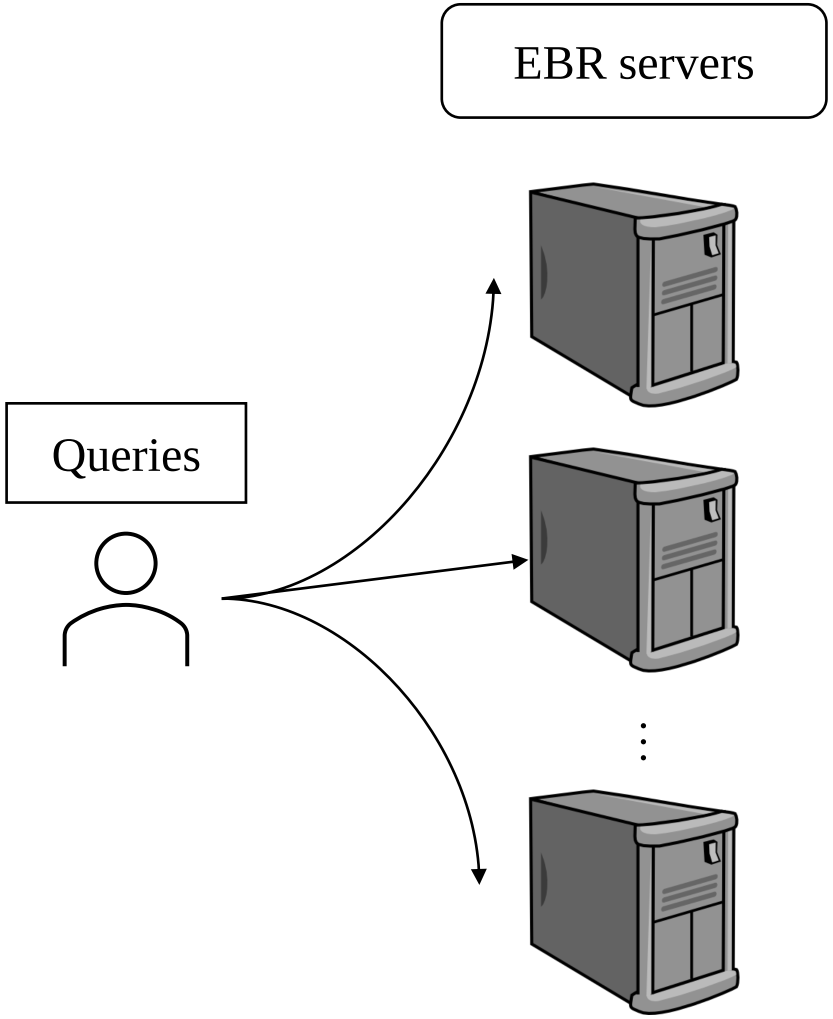



```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

Memory-intensive

Compute-intensive

Requirement of EBR



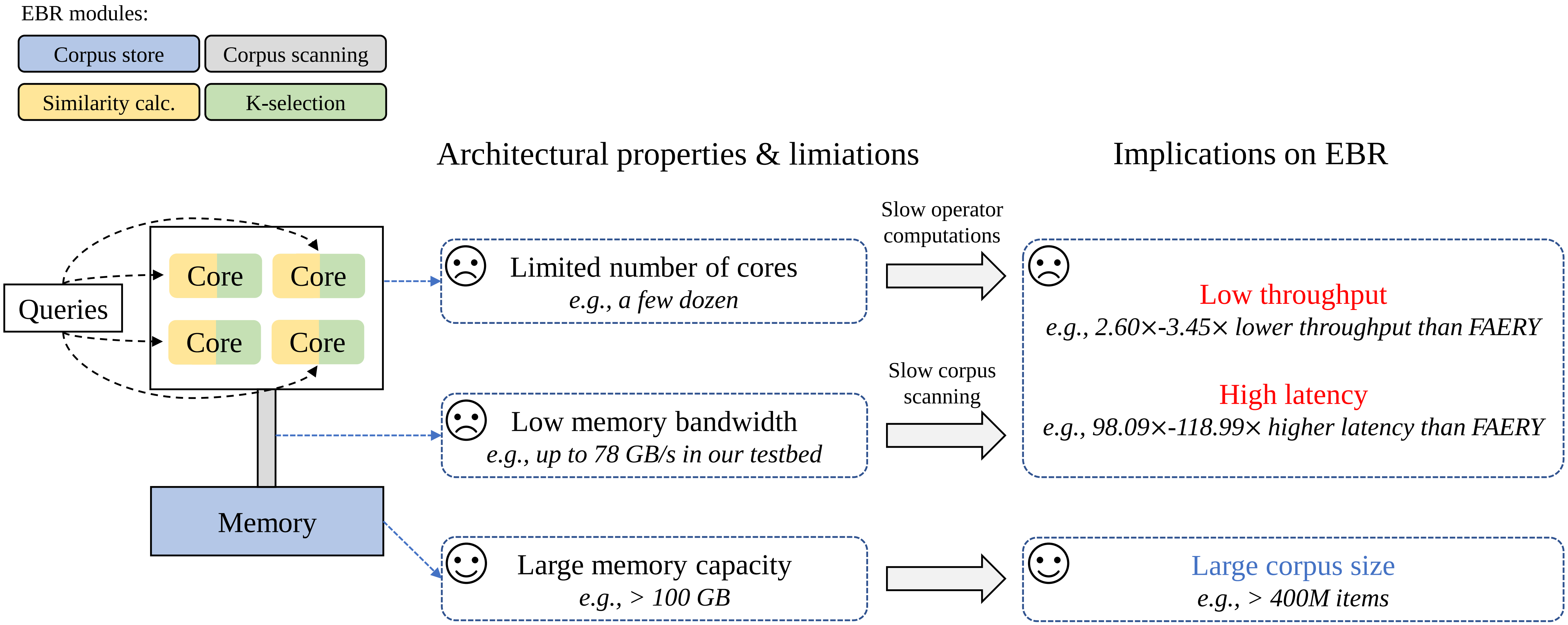
- High throughput
  - A small number of servers (low cost) to serve a target QPS

- Low latency
  - Good user experience
    - Low user's overall waiting time
    - *or*
    - High quality of recommendation results

  - EBR system usually has a latency SLA (e.g., 10 ms)

Both throughput and latency (thus latency-bounded throughput) matter!

Existing Work: CPU-based EBR

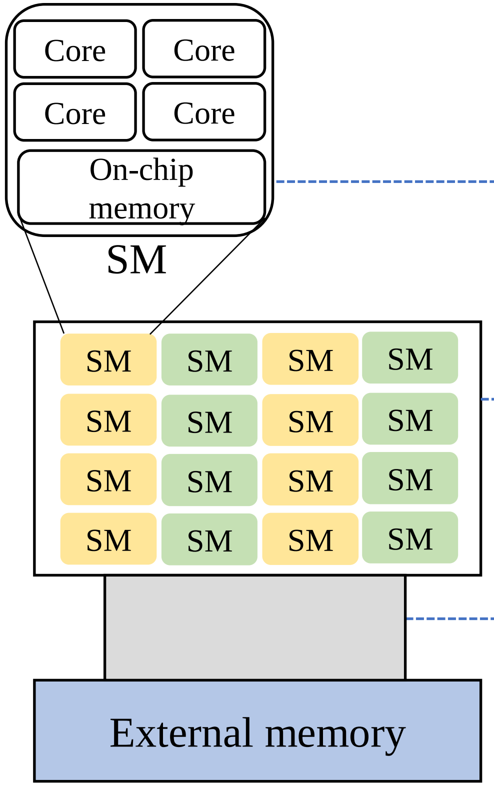


Existing Work: GPU-based EBR

EBR modules:

Corpus store | Corpus scanning

Similarity calc. | K-selection



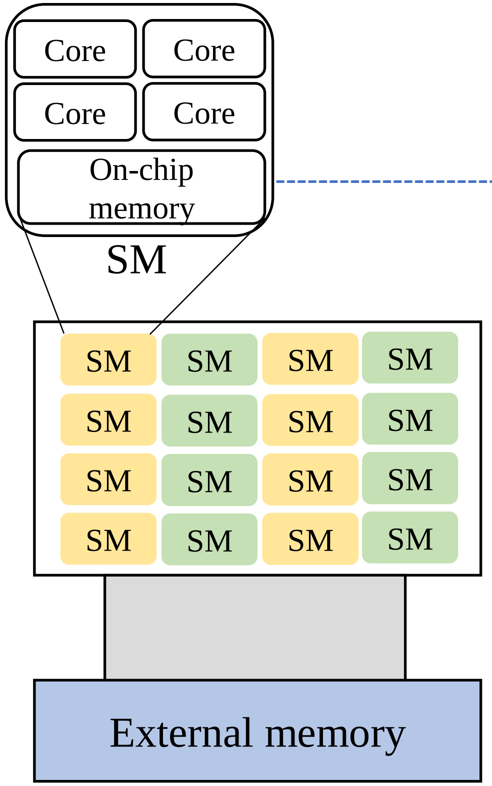
Architectural properties & limiations		Implications on EBR
☹ Explicit resource boundary between SMs - *Small & exclusive on-chip memory in an SM* - *Cross-SM comm. via external memory*	⇨	☹ High latency *Inefficient K-selection*
☺ Many streaming multiprocessors (SMs) with massive SIMT cores	⇨	☺ High throughput *Fast similarity calculation*
☺ High external memory bandwidth *e.g., 300 – 900 GB/s*	⇨	☺ High throughput *Fast corpus scanning*

Existing Work: GPU-based EBR (GPU-e)

EBR modules:

Corpus store | Corpus scanning

Similarity calc. | K-selection



Architectural properties & limiations

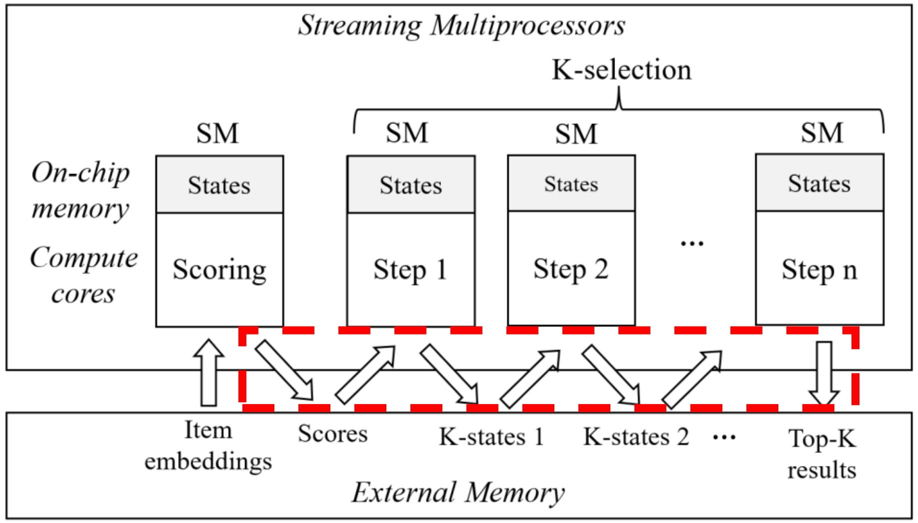
☹ Explicit resource boundary between SMs

- *Small & exclusive on-chip memory in an SM*
- *Cross-SM comm. via external memory*

Multiple memory traversals per query leads to high latency, worsening in batch

e.g., 12.36x latency increase when batch size is increased from 1 to 16 with a 4M corpus

Implications on EBR



GPU-e: K-selection with external memory

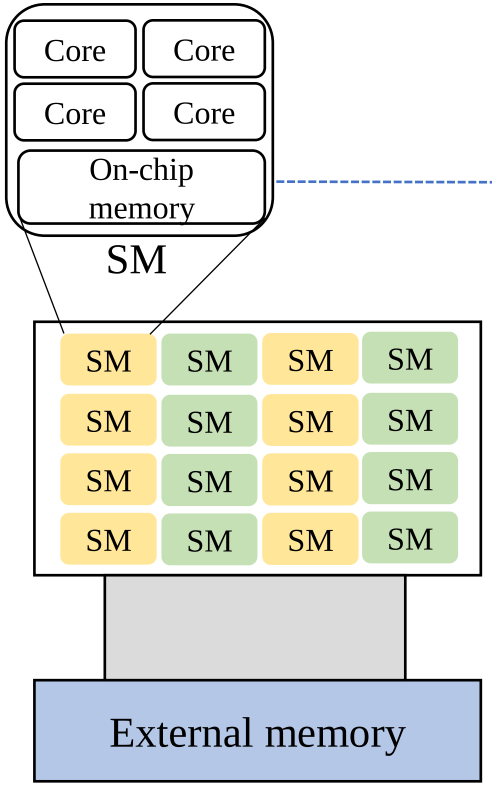
A. Shanbhag, etc. "Efficient Top-K Query Processing on Massively Parallel Hardware." *2018 ICMD.*

Existing Work: GPU-based EBR (GPU-o)

EBR modules:

Corpus store | Corpus scanning

Similarity calc. | K-selection



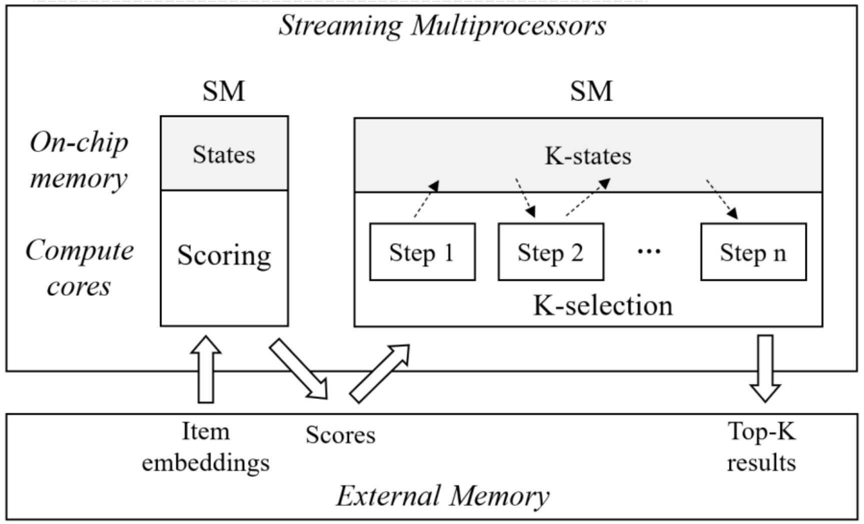
Architectual properties & limiations

☹ Explicit resource boundary between SMs

- *Small & exclusive on-chip memory in an SM*
- *Cross-SM comm. via external memory*

- Small on-chip SRAM leads to *a small K value* *(e.g., 2048 in WarpSelect)*
- K-selection in a single SM leads to *high latency* *(e.g., 9.48×-18.81× higher latency than FAERY)*

Implications on EBR



GPU-o: K-selection with on-chip memory

J. Johnson, etc. "Billion-scale Similarity Search with GPUs." *2017 arXiv*.

Summary of Existing Work

- None of existing EBRs achieve high throughput and low latency simultaneously
  - CPU-based EBR:
    - **Pros:** Large memory capacity for large corpus size
    - **Cons:** Limited memory bandwith and limited number of CPU cores results in high latency and low throughput
  - GPU-based EBR:
    - **Pros:** High memory bandwidth for fast corpus scanning and massive SIMT cores for fast similarity calculation
    - **Cons:** Explicit resource boundary between SMs results in poor pipeline support and thus high latency and low latency-bounded throughput

What should a practically ideal EBR architecture that achieves maximal latency-bounded throughput look like?

Practically Ideal EBR Architecture

```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

Practically Ideal EBR Architecture

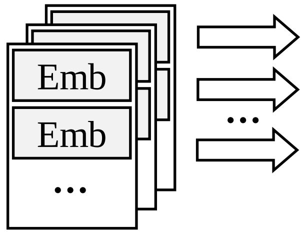
Corpus store & corpus scanning

```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

① ***Large external memory***

② ***High memory bandwidth***

② ***High memory bandwidth***



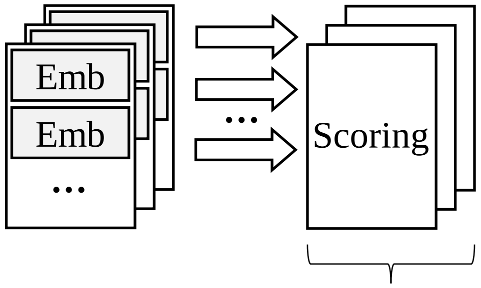
① ***Large external memory***

Practically Ideal EBR Architecture

Similarity calculation

```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

③ ***Data parallelism***



③ ***Data parallelism***

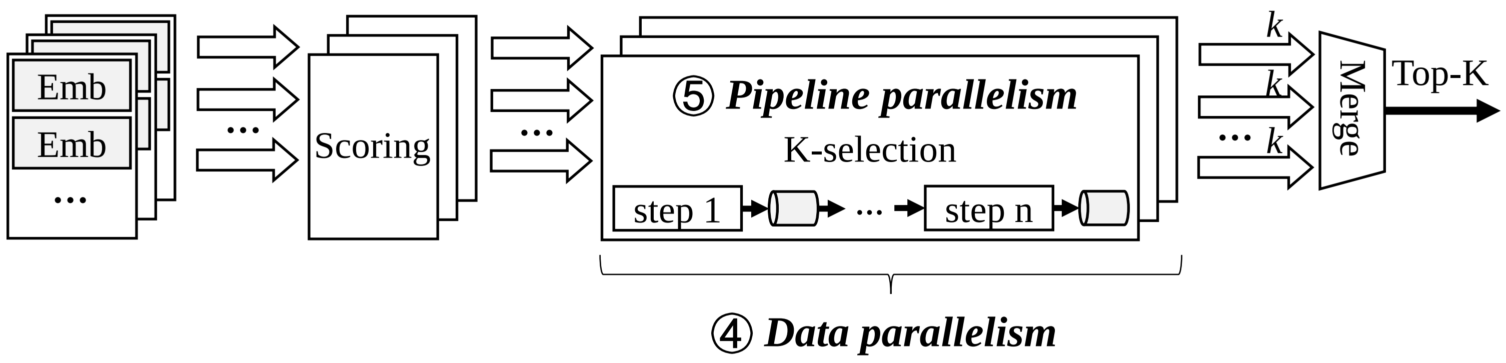
Practically Ideal EBR Architecture

K-selection

```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

④ ***Data parallelism***

⑤ ***Pipeline parallelism***

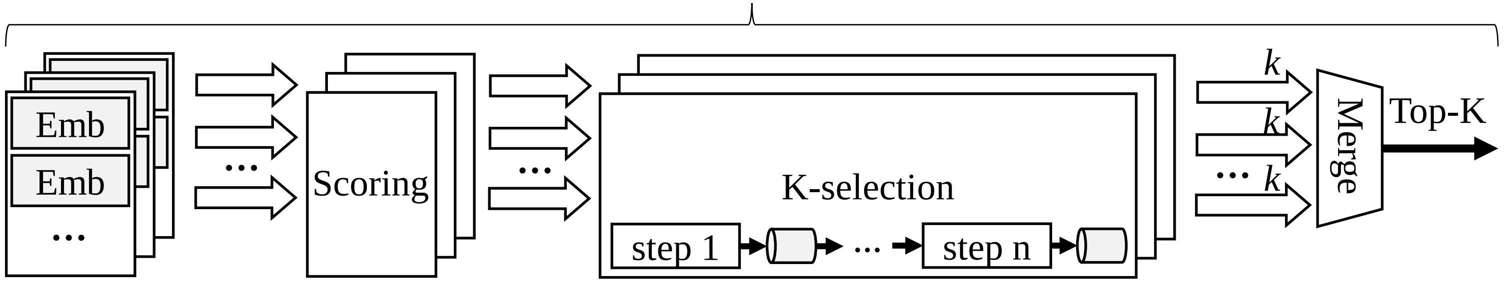


Practically Ideal EBR Architecture

Entire EBR data flow

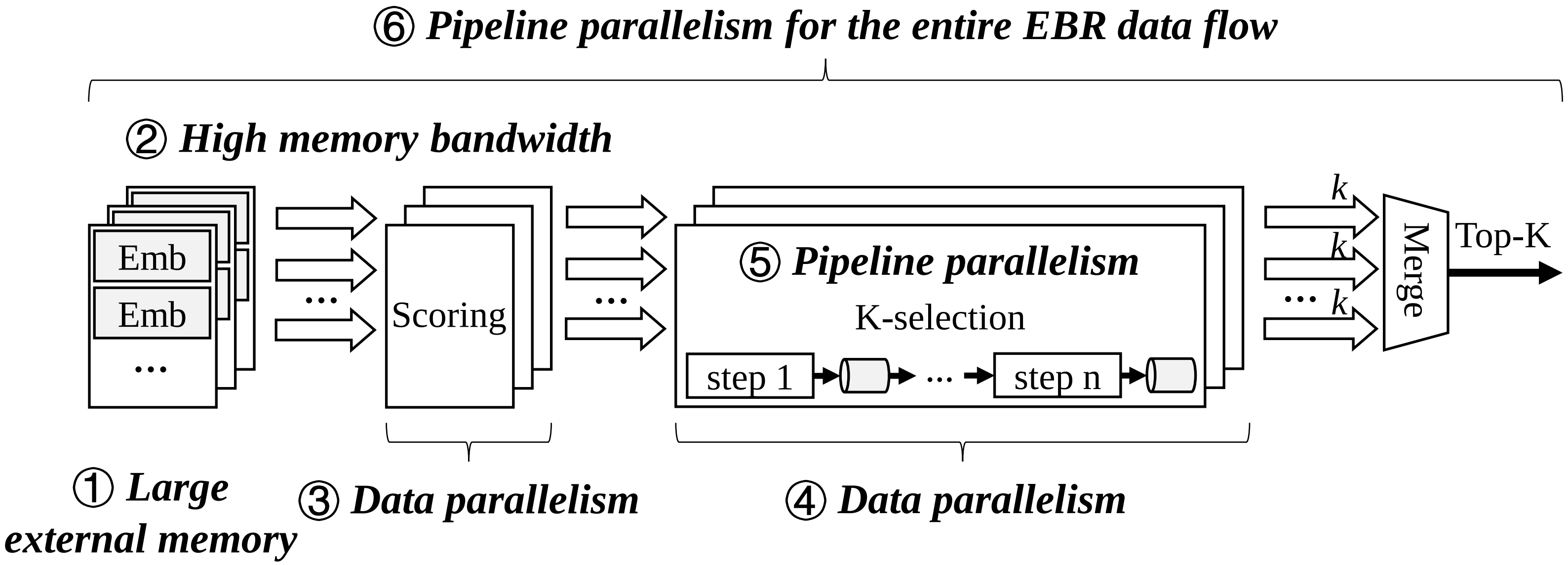
```
# Scoring
for i in corpus_size:
    item_emb = corpus[i] # corpus scanning
    scores[i] = sim_calc(user_emb, item_emb) # similarity calc
# K-selection
ret_items = topk(scores) # returns the sorted top_k items
```

⑥ ***Pipeline parallelism for the entire EBR data flow***



Practically Ideal EBR Architecture

Batch size = 1: achieve minimal latency



$$latency = \frac{S}{B} + C$$

(Theoretical lower bound: $\frac{S}{B}$)

S : corpus size

B : memory bandwidth

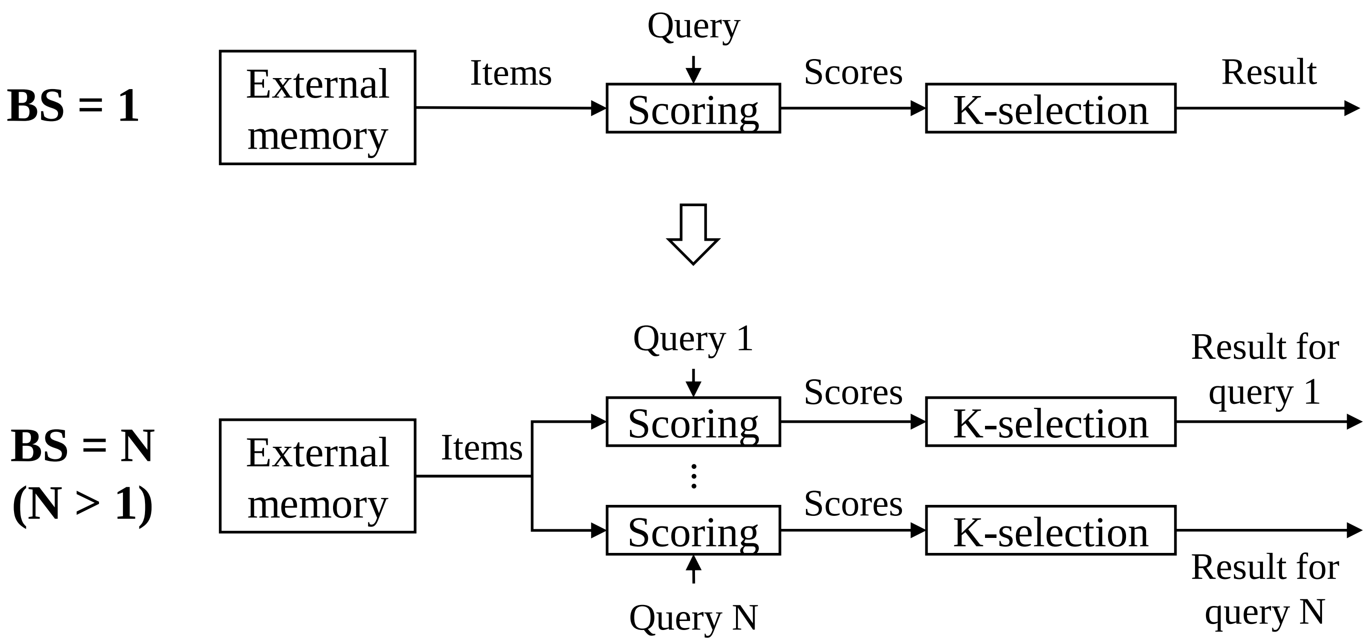
C : pipeline latency

(typically small, so that $\frac{S}{B} + C \approx \frac{S}{B}$)

Fully-pipelined and non-congested data flow with a single pass of external memory

Practically Ideal EBR Architecture

Batch size = N: achieve maximal latency-bounded throughput

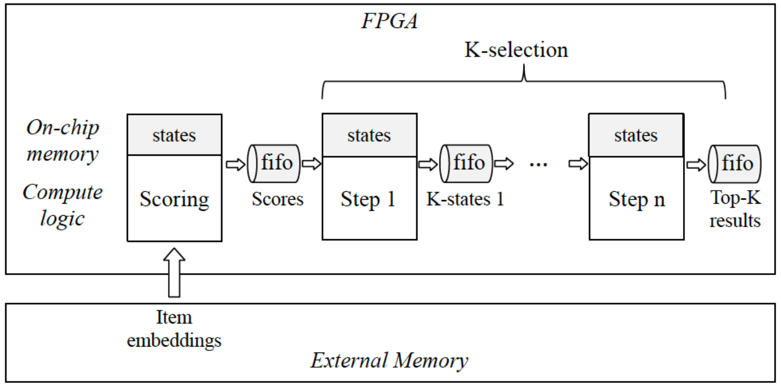


Increase latency-bounded throughput linearly by increasing batch size while preserving minimal latency

FPGA Opportunities for the Ideal Architecture

- FPGA is a programmable chip
  - HBM: 8-32GB, 460GB/s
  - Massive on-chip memories (10s of MB)
  - Massive programmable logic elements
  - Programmable interconnects

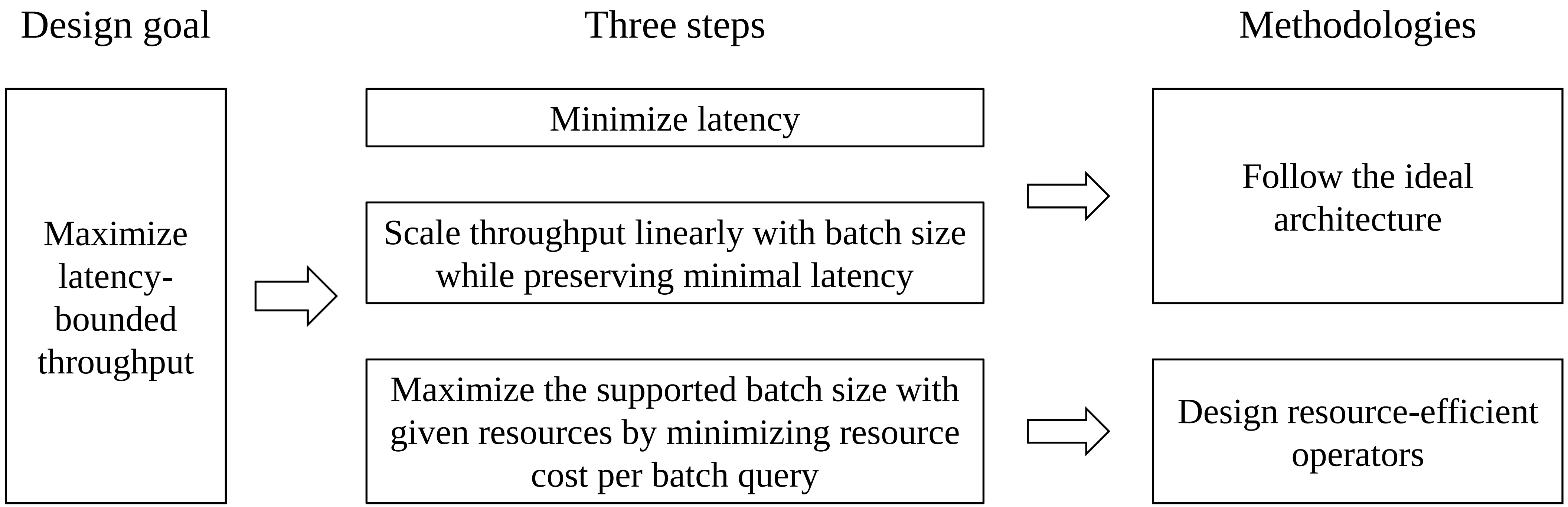
- Meet properties of the ideal EBR architecture
  - Moderate corpus store & fast corpus scanning
  - Data parallelism for similarity calculation
  - Data/pipeline parallelism for K-selection
  - Fully-pipelined data flow



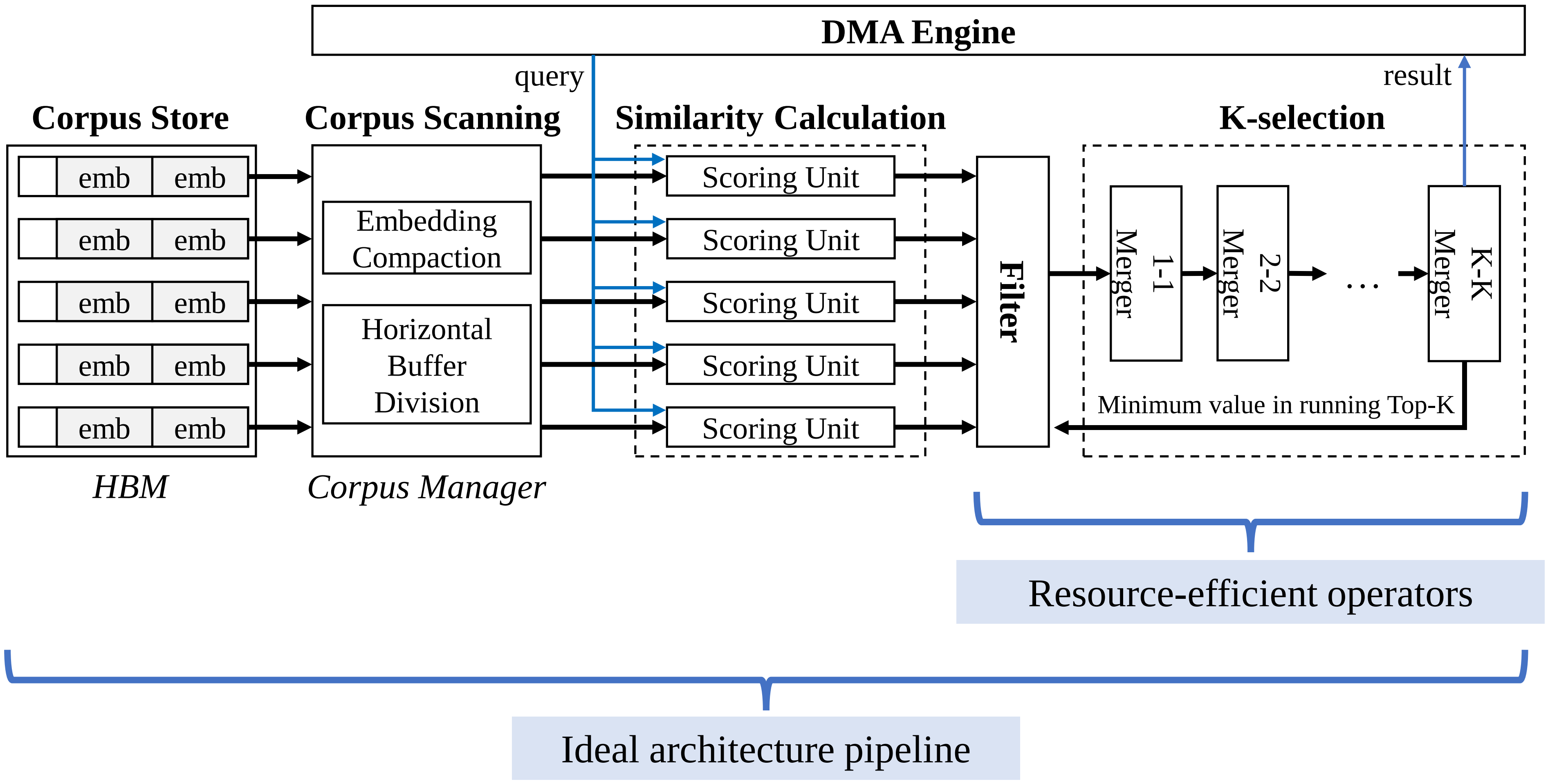
Motivate our design: **FAERY**

FPGA-Accelerated Embedding-based Retrieval sYstem)

FAERY Design Goal

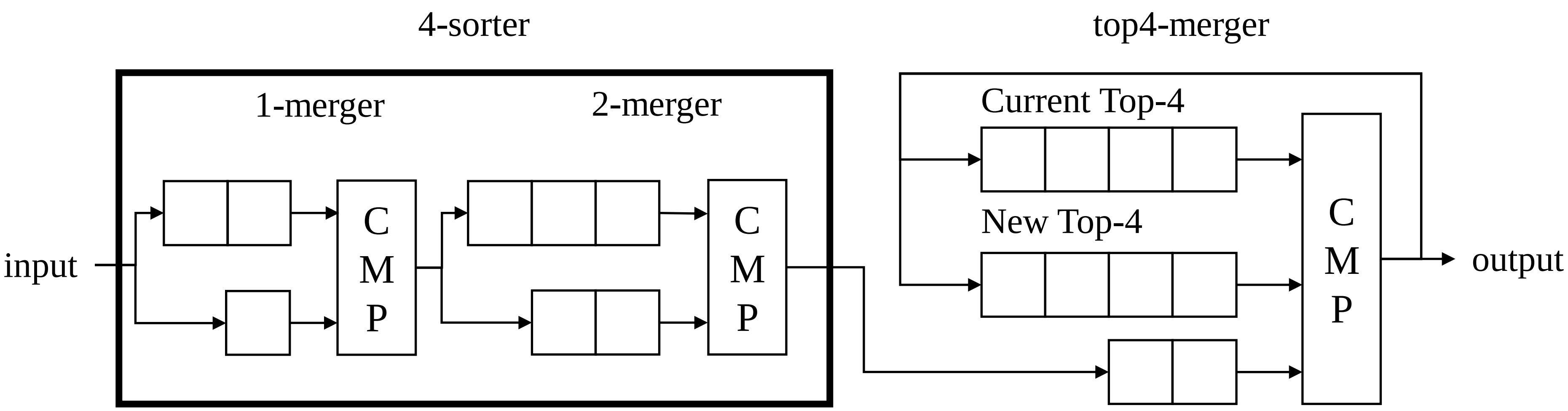


FAERY Accelerator Architecture



FAERY Accelerator - K-selection

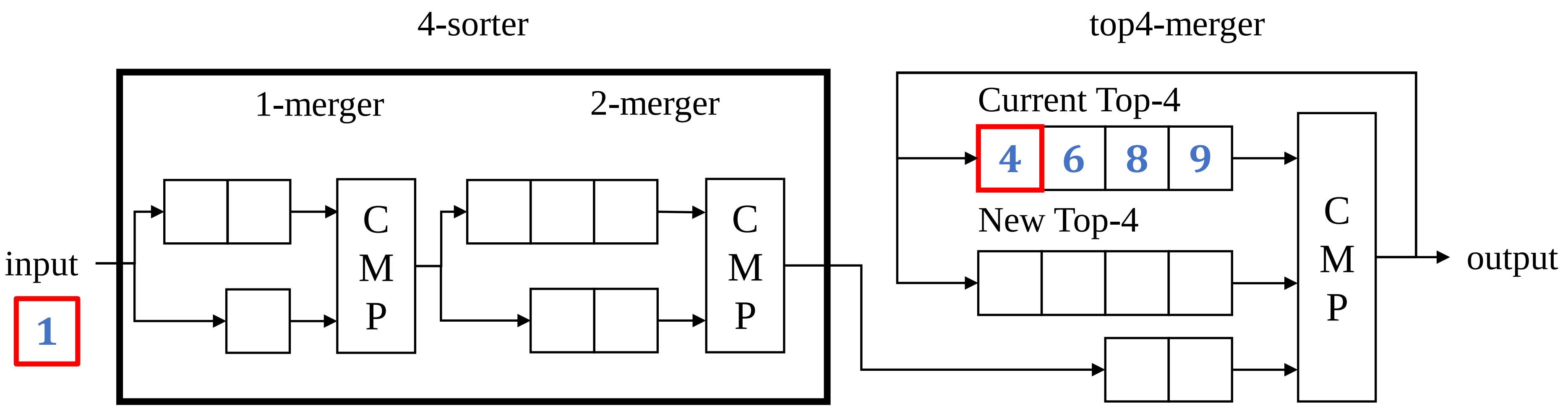
An example of 4-selection pipeline based on bottom-up merge sort



1. Bottom-up merge sort allows processing input scores in a streaming manner.
2. Pipeline parallelism within K-selection is compute-efficient and scalable, e.g., the above architecture requires only $O(logk)$ comparators.

N. Matsumoto, etc. "Optimal parallel hardware k-sorter and top k-sorter, with FPGA implementations." *2015 14th International Symposium on Parallel and Distributed Computing*.

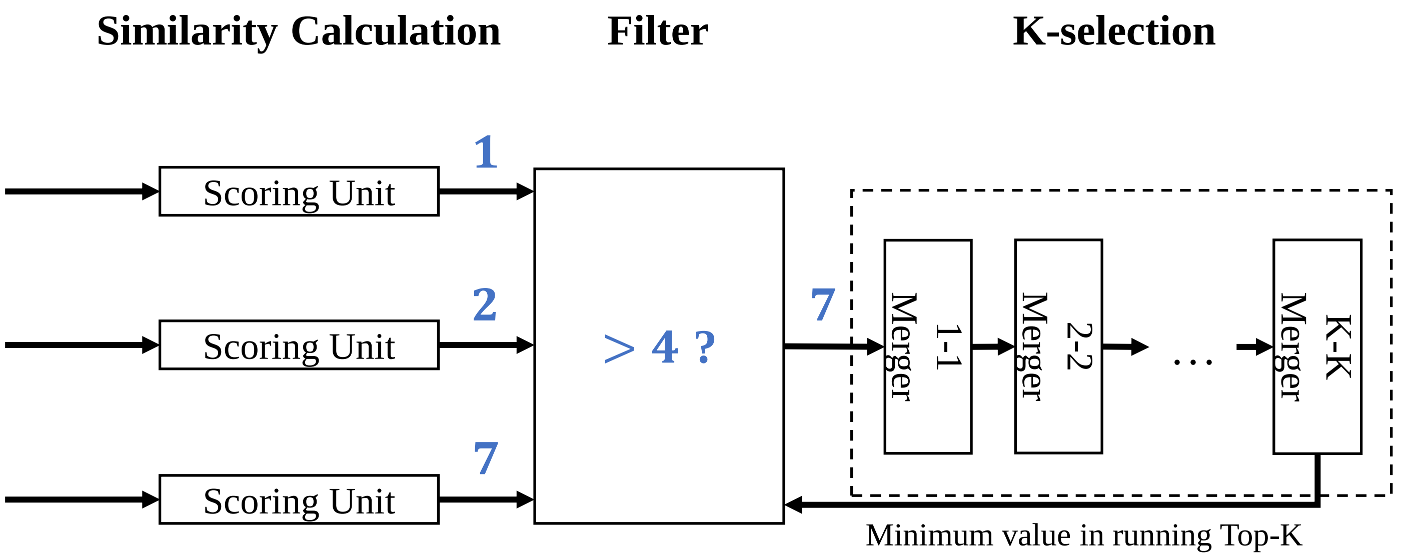
Observation



If the input score is **not greater** than the minimum score of the current Top-K result, this input will not be in the new Top-K result

These non-Top-K items can be early dropped to significantly reduce traffic to K-selection

FAERY Accelerator - Filter



Saving resource by using filter and a small number of K-selection pipelines to match the bandwidth of multiple scoring units

Prototype Implementation

Prototype setting

- Xilinx VU35P FPGA with a clock frequency of 400 MHz
- One embedding contains 128 elements of 2 bytes each
- *k* is 1024

Per-query pipeline implementation

HBM	Corpus manager	Similarity calculation	Filter	K-selection
8 GB & 460 GB/s Support 16M items	400 MHz matches the HBM bandwidth	Inner product latency = 6 clock cycles	Save 32% on-chip memories and 27% compute resources	Bottom-up merge sort latency = 1034 clock cycles

Resource utilization & batch implementation (batch size = 3)

	Per-query resources	Common resources
LUT	7.31%	11.05%
FF	6.98%	14.78%
BRAM	13.05%	10.66%
DSP	8.6%	0.07%

Evaluation Setup

Baseline:

- Faiss, an open-source similarity search library, which supports both CPU and GPU
- Faiss GPU implementation utilizes *WarpSelect*, denoted as GPU-o
- Another GPU baseline replaces *WarpSelect* with *RadixSelect*, denoted as GPU-e
- Ideal latency ($\frac{S}{B} + C$) of the ideal architecture

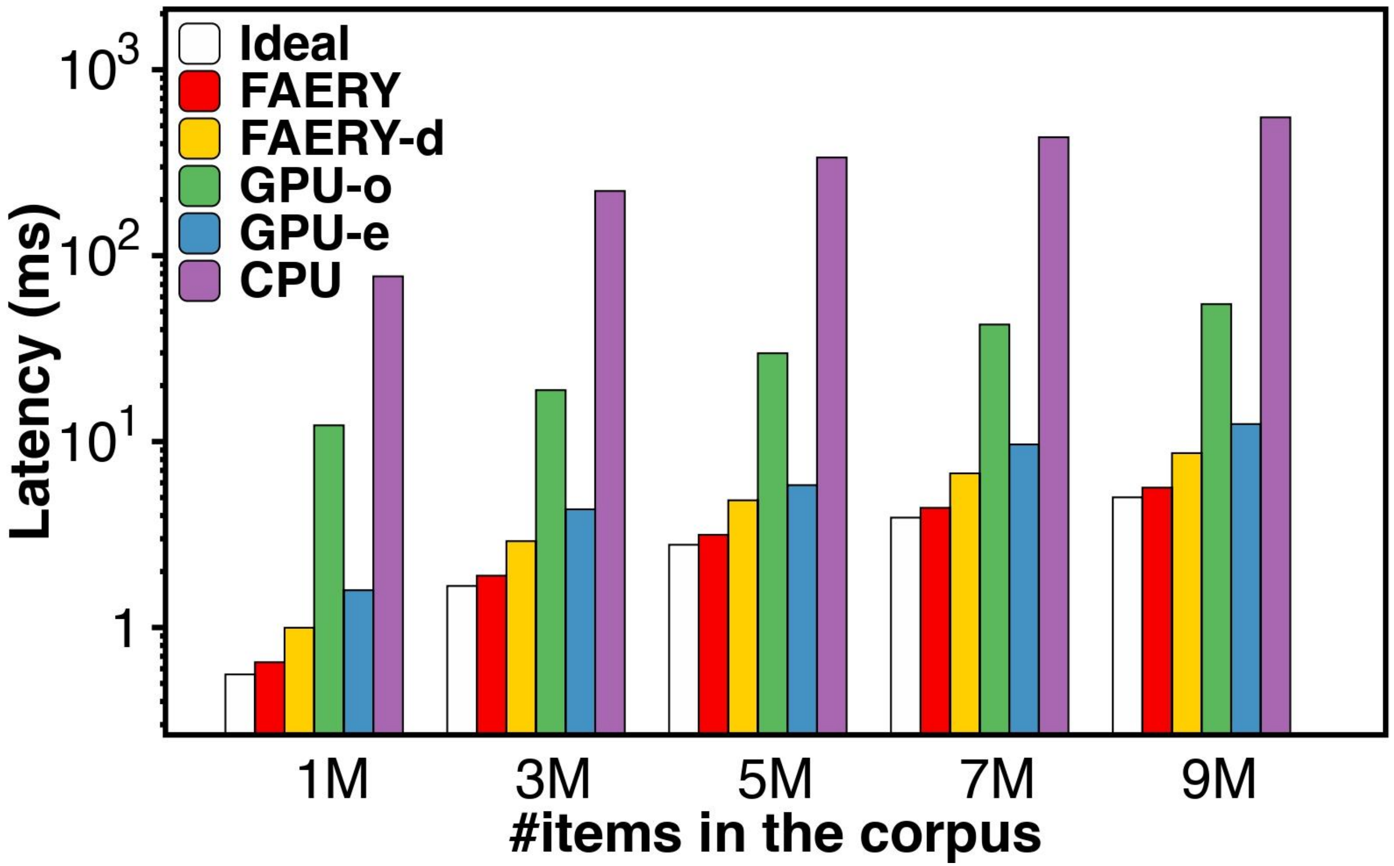
Platforms:

- CPU-based EBR: two 16-core Intel Xeon Gold 5218 CPUs
- GPU-based EBR: Nvidia T4 GPU with 300 GB/s GDDR6
- FAERY-d: degraded FAERY with the same memory bandwidth (300 GB/s) as T4

Corpora:

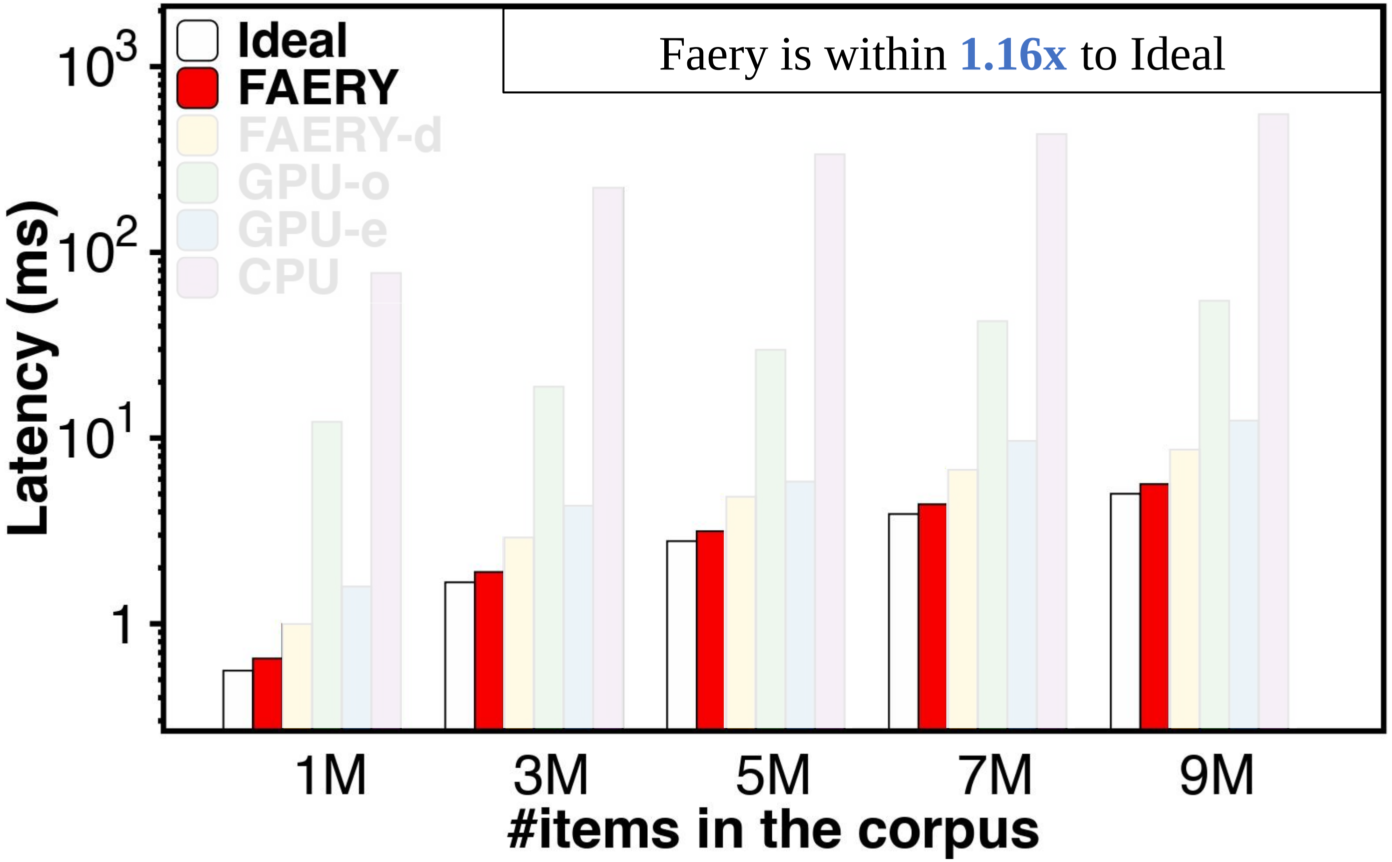
- Synthetic random corpora with different corpus size (1M-15M items)

Latency



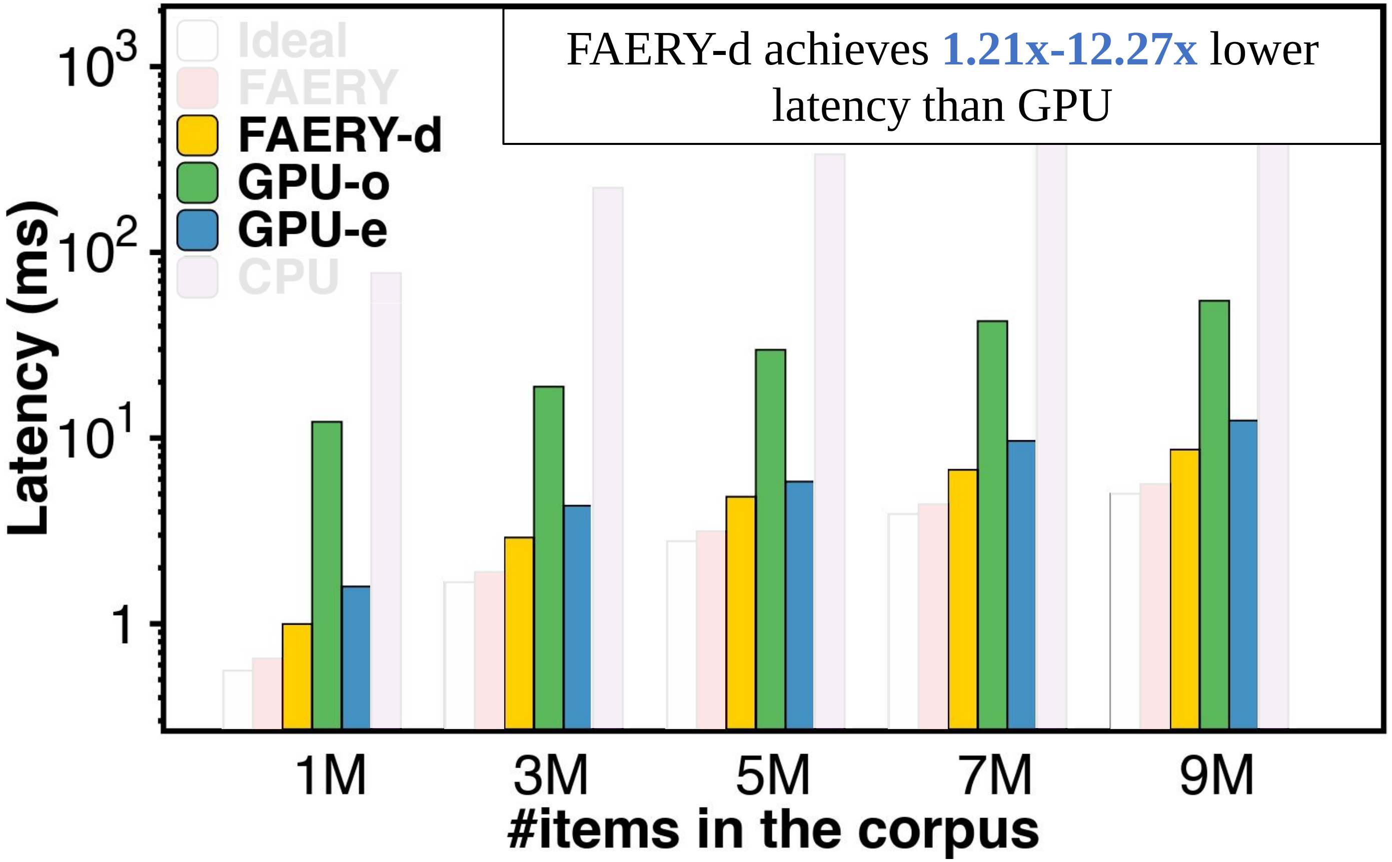
Ideal
FAERY
FAERY-d
GPU-o
GPU-e
CPU
Latency (ms)
10^3
10^2
10^1
1
1M
3M
5M
7M
9M
#items in the corpus

Latency



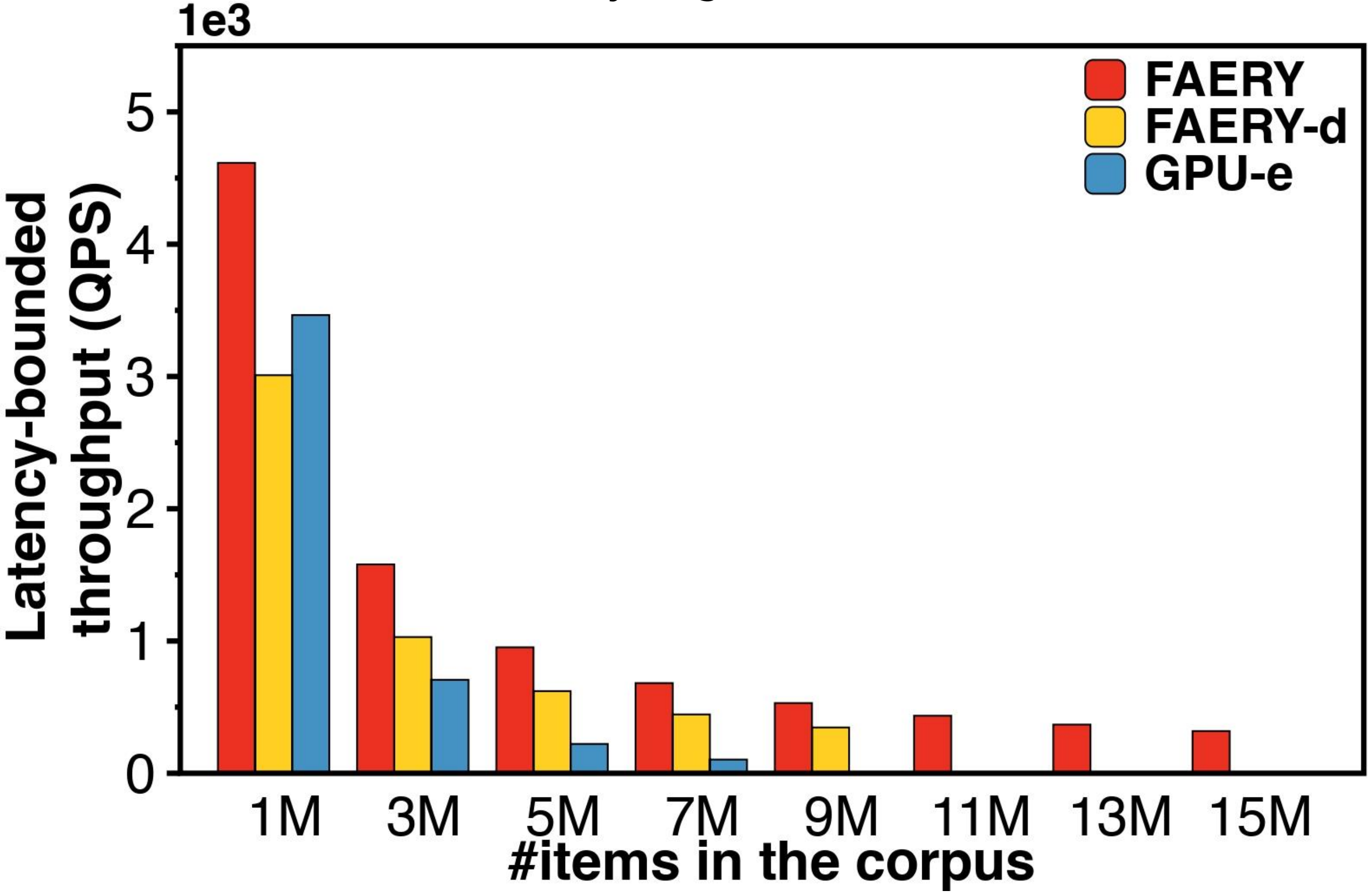
Faery is within **1.16x** to Ideal
Ideal
FAERY
FAERY-d
GPU-o
GPU-e
CPU
10^3
10^2
10^1
1
Latency (ms)
1M
3M
5M
7M
9M
#items in the corpus

Latency


Ideal
FAERY
FAERY-d
GPU-o
GPU-e
CPU
FAERY-d achieves 1.21x-12.27x lower latency than GPU
10^3
10^2
10^1
1
Latency (ms)
1M
3M
5M
7M
9M
#items in the corpus

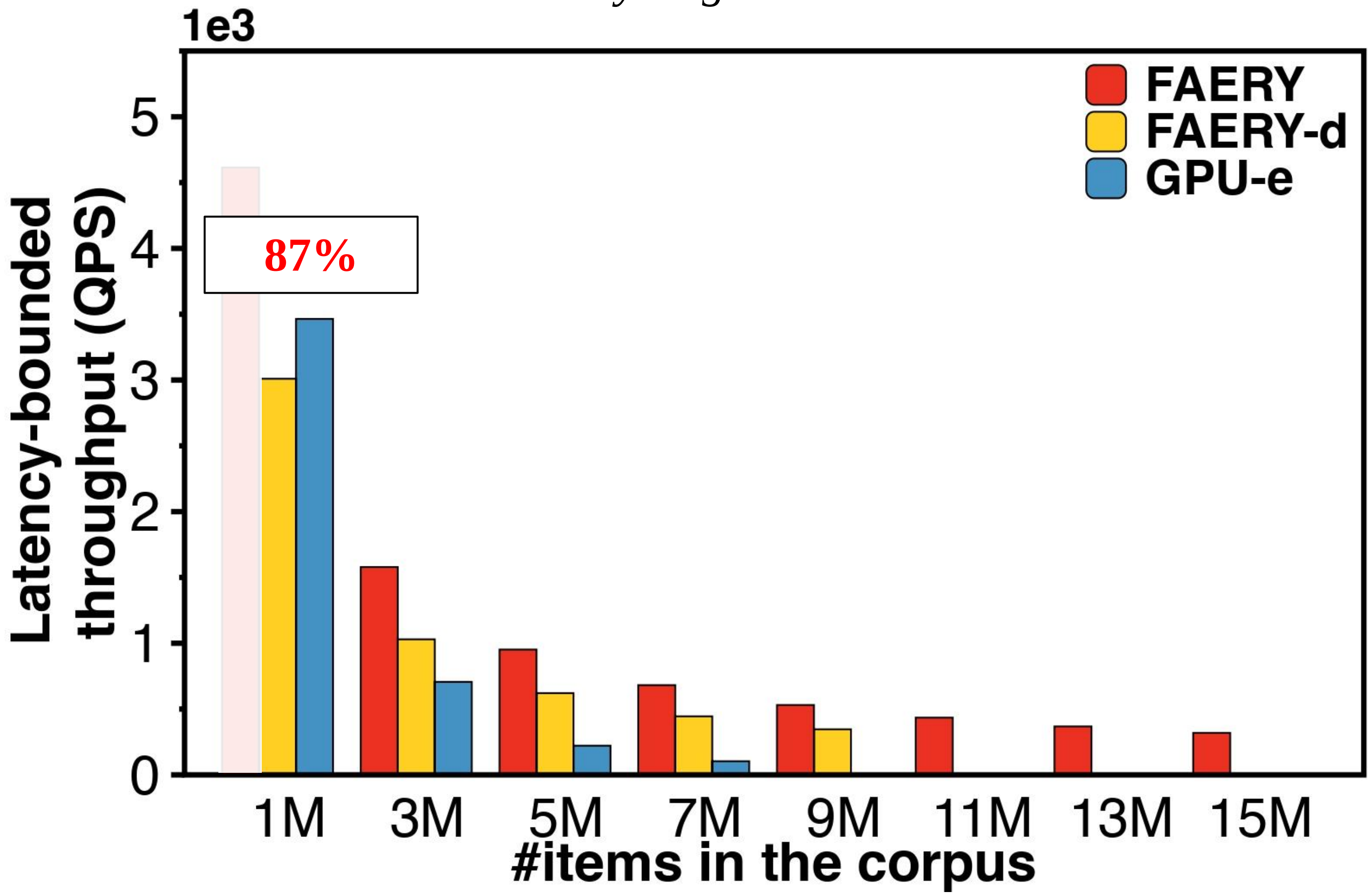
Latency-bounded Throughput

Latency target: 10 ms



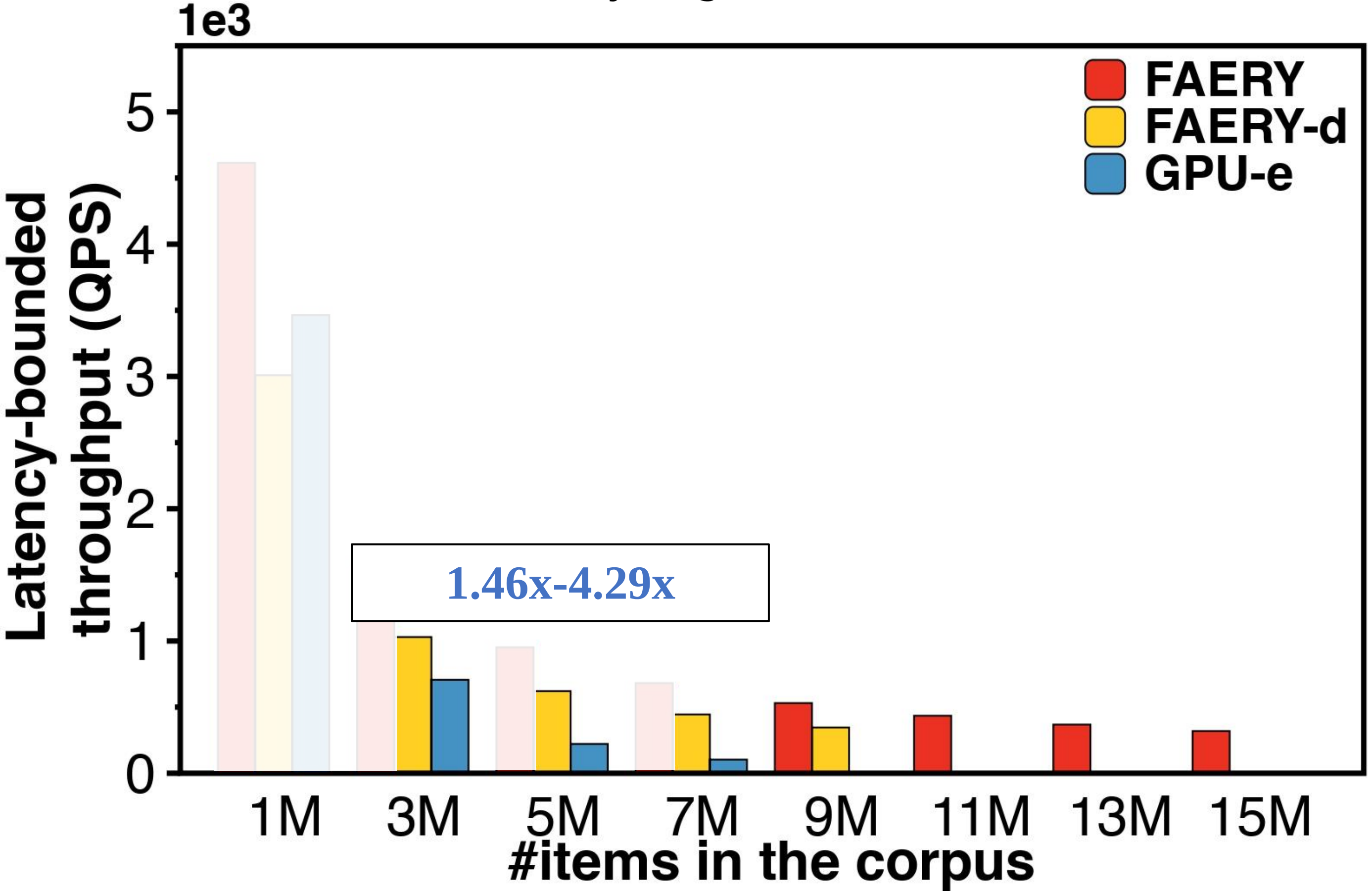
Latency-bounded Throughput

Latency target: 10 ms

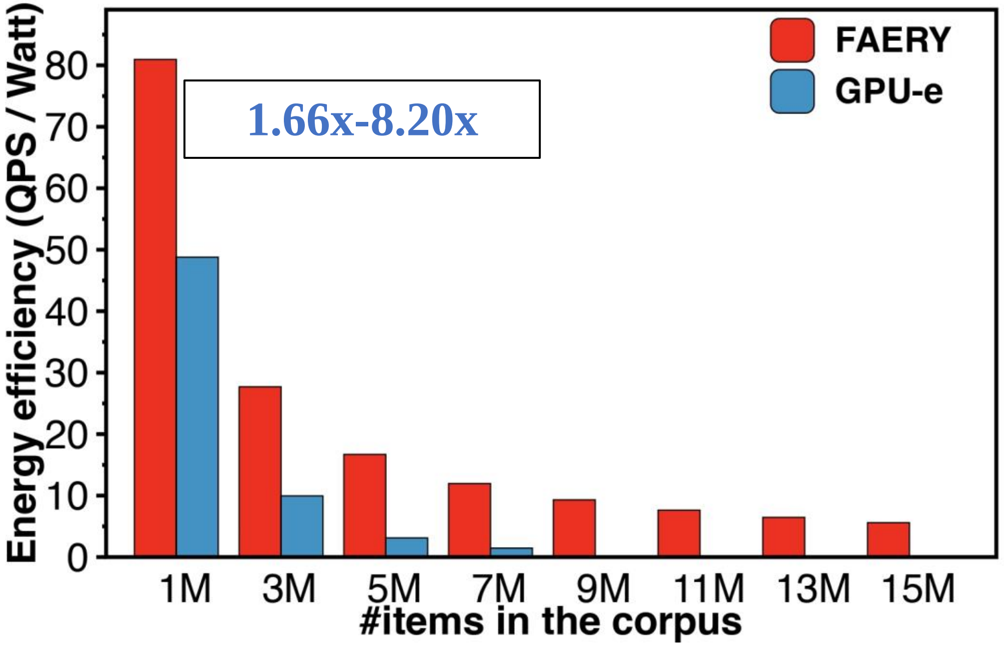


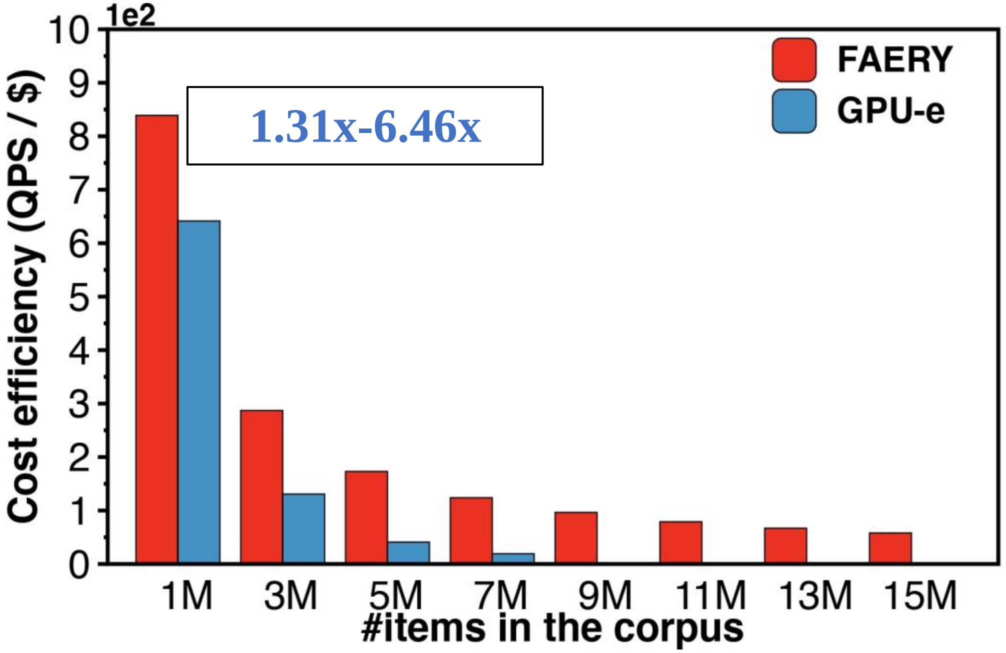
Latency-bounded Throughput

Latency target: 10 ms



Energy & Cost Efficiency





Summary of Evaluation

Architecture	Properties
CPU-based EBR	Support extremely large corpus (> 100 GB) with poor performance
GPU-based EBR	Provide high raw throughput (up to 1.44x compared to FAERY) with poor latency
FAERY	Provide low latency (within 1.16x to ideal) and high latency-bounded throughput (up to 4.29x compared to GPU) with programmability/maintenance overhead

Conclusion

- We study the EBR algorithm from the first principles and derive a practically ideal EBR architecture
- We design FAERY, a domain specific accelerator for EBR, which is an embodiment of the ideal EBR architecture with filtering optimization
- FAERY can be extended to accelerate a generic vector search in future

Thank you!

Contact email: czengaf@connect.ust.hk