

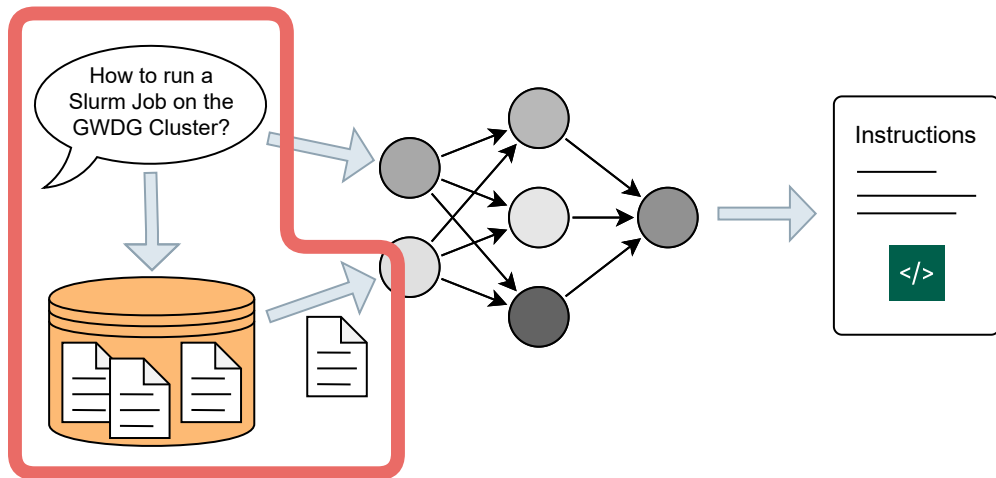


Darius Heere

MPI-Based Exact Dense Vector Search for RAG Pipelines

Supervisor: Claas Kochanke

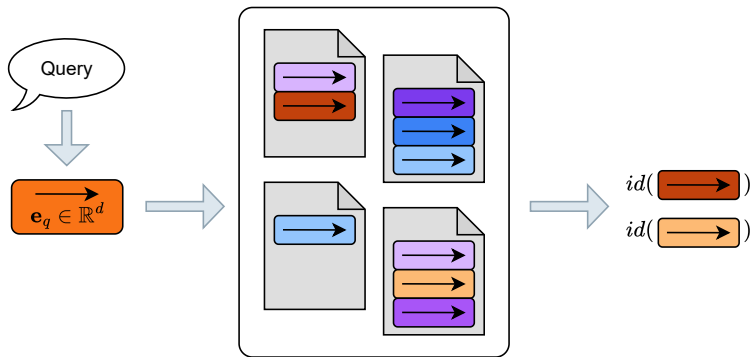
Project focus: Retrieval in RAG



Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution
- 3 Parallelized Algorithm
- 4 Benchmark Results
- 5 Conclusion
- 6 Appendix

Idea: Exact top-k search in a vector index



Retrieve the *ids* of the top-*k* indexed vectors that best match the query embedding.

Similarity score via dot product

$$s_i = \mathbf{e}_q^\top \mathbf{e}_i = \sum_{j=1}^d e_{q,j} e_{i,j}$$

$$\mathcal{I} = \{\mathbf{e}_1, \dots, \mathbf{e}_N\}$$

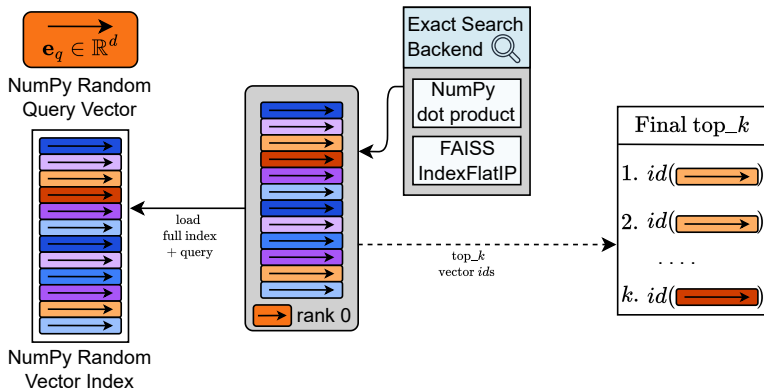
Top-*k* retrieval

Compute s_i for every indexed vector $\mathbf{e}_i \in \mathcal{I}$, then return the *ids* of the *k* highest-scoring vectors.

Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution**
- 3 Parallelized Algorithm
- 4 Benchmark Results
- 5 Conclusion
- 6 Appendix

Exact top-k search: Sequential solution

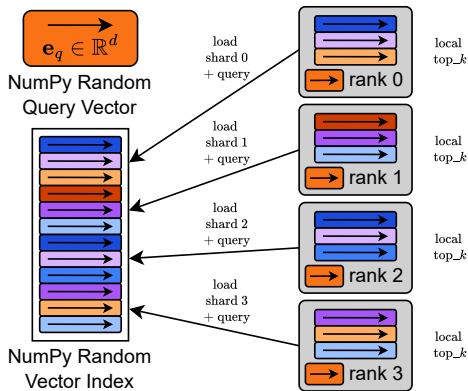


Limitation: Memory and compute are limited to a single process.

Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution
- 3 Parallelized Algorithm**
- 4 Benchmark Results
- 5 Conclusion
- 6 Appendix

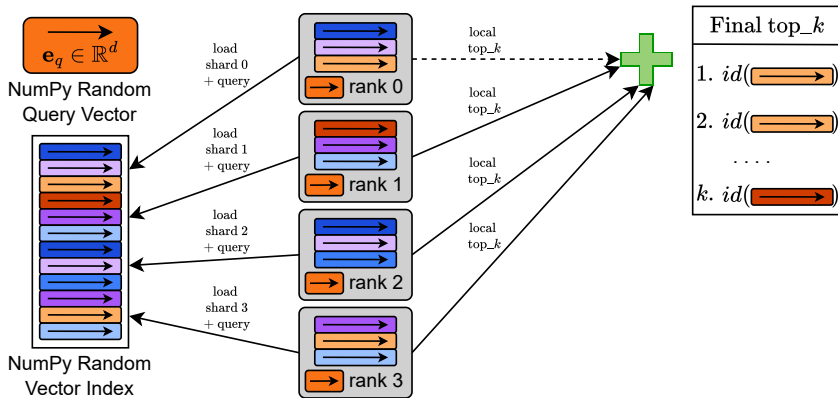
Exact top-k search: Parallel solution



Idea: Split the vector index into shards

⇒ Each MPI rank loads and searches only a local subset, the query vector is replicated.

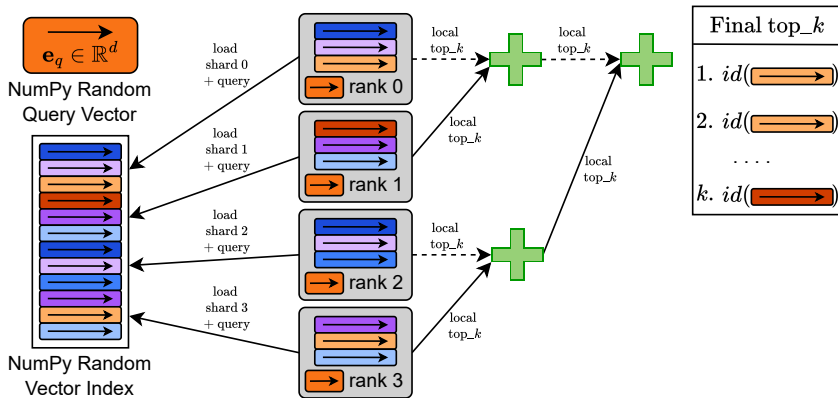
Parallel exact top-k search: Centralized merging



Communication: Local top- k scores and global vector IDs are gathered on rank 0.

⇒ Correctness is checked against an independent sequential full-index search.

Parallel exact top-k search: Tree-based merging



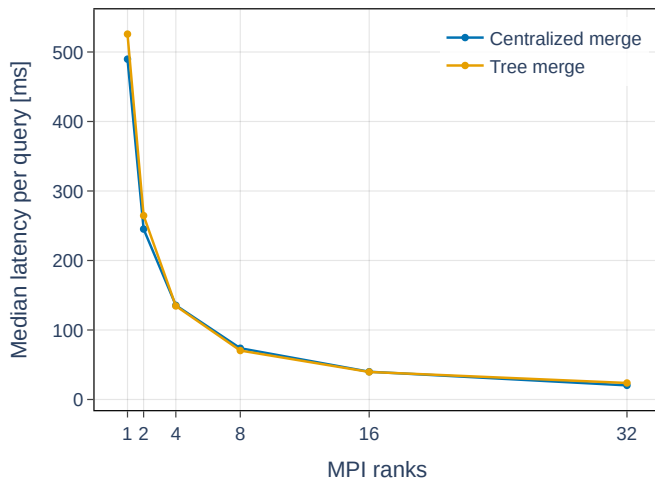
Communication: Partner ranks exchange local top-k scores and global vector IDs.

⇒ Merge work is distributed instead of concentrating all candidates on rank 0.

Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution
- 3 Parallelized Algorithm
- 4 Benchmark Results**
- 5 Conclusion
- 6 Appendix

Hot latency strong scaling: Retrieval latency vs. MPI ranks



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

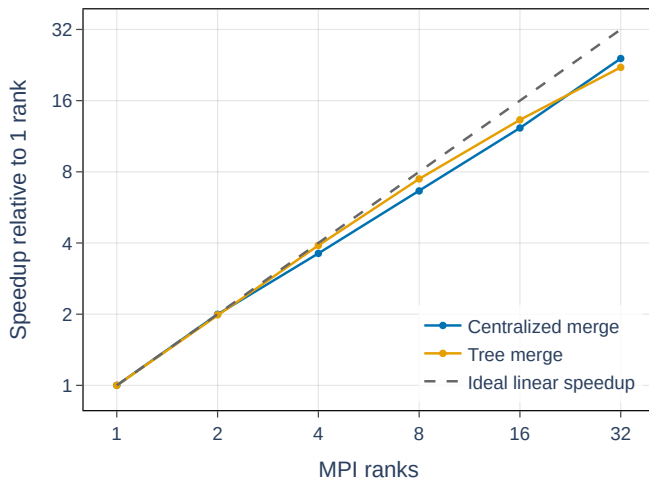
Cluster Setup

Nodes	1
Ranks/node	1-32
CPUs/rank	1
FAISS thr./rank	1
RAM/node	32 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	6.0%–38.1%

Hot latency strong scaling: Speedup vs. MPI ranks



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

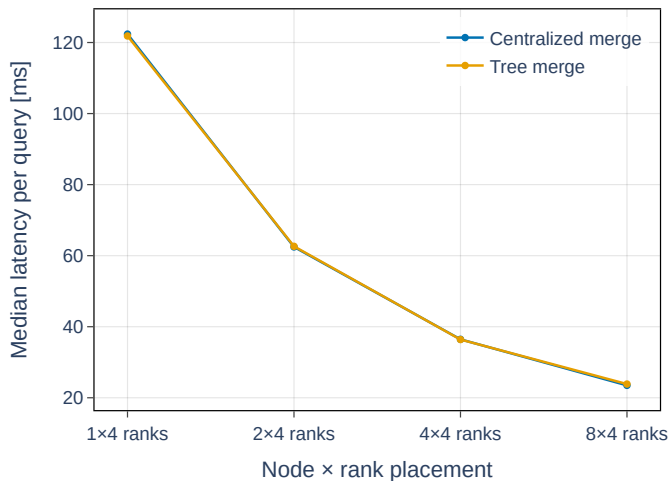
Cluster Setup

Nodes	1
Ranks/node	1–32
CPUs/rank	1
FAISS thr./rank	1
RAM/node	32 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	6.0%–38.1%

Hot latency multi-node scaling: Latency vs. nodes



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

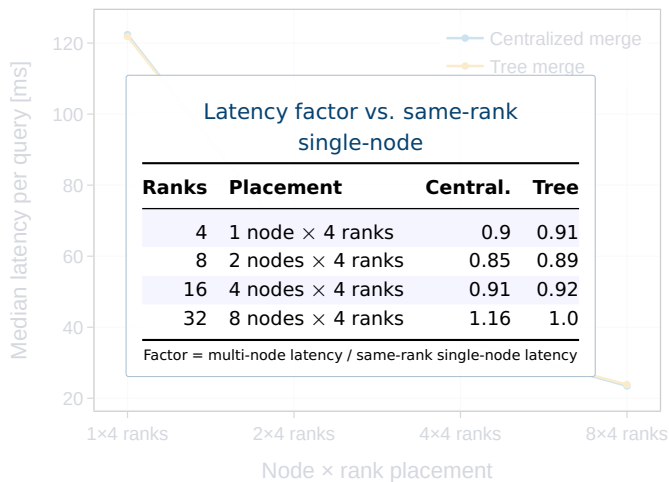
Cluster Setup

Nodes	1–8
Ranks/node	4
CPUs/rank	1
FAISS thr./rank	1
RAM/node	16 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	0.3%–43.1%

Hot latency multi-node scaling: Same-rank latency factor



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

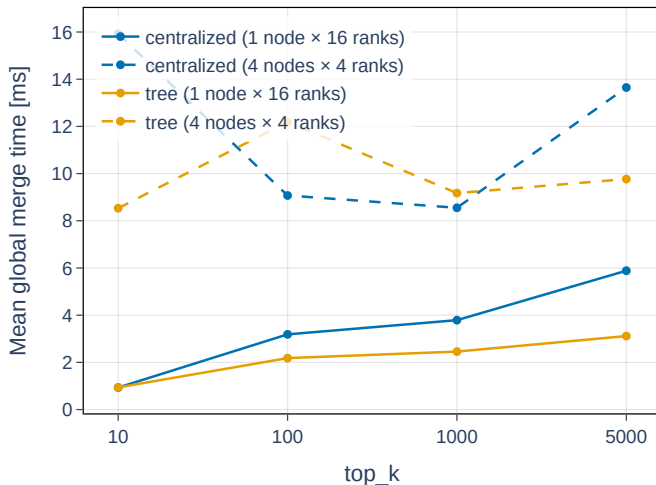
Cluster Setup

Nodes	1–8
Ranks/node	4
CPUs/rank	1
FAISS thr./rank	1
RAM/node	16 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	0.3%–43.1%

Merge stress with increasing top-k: Merge timings



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10 – 5000
Backend	FAISS IndexFlatIP

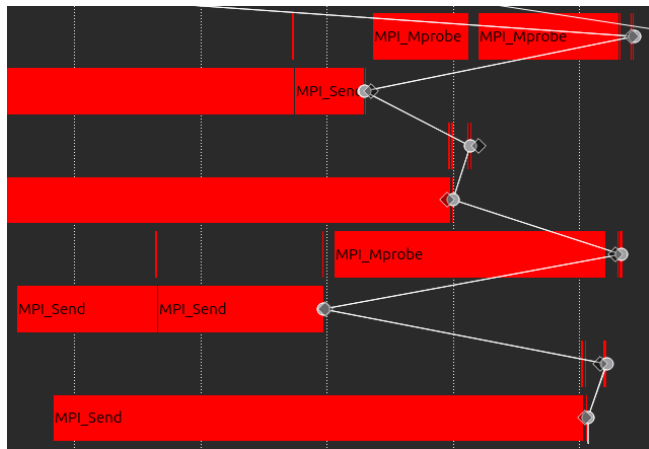
Cluster Setup

Nodes	1,4
Ranks/node	16, 4
CPUs/rank	1
FAISS thr./rank	1
RAM/node	16 GB (4×4), 32 GB (1×16)

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	mean
Centralized	gather + rank 0 merge
Tree	full tree reduction

Vampir warnings and merge trace details



Zoomed in Vampir trace view of the merge phase.

Warnings		
Trace Info		
Type	Time	Info
▼ Pending messages		30 in total
Pending send message	1.518009 s	
Pending send message	1.518082 s	
Pending send message	1.518745 s	
Pending send message	1.518813 s	
Pending send message	1.519736 s	
Pending send message	1.519857 s	
Pending send message	1.520251 s	
Pending send message	1.520505 s	
Pending send message	1.521747 s	
Pending send message	1.521785 s	

Vampir warnings window.

Run setup

Merge mode	mpi_tree
Backend	NumPy
Warmups	0
Measured runs	1
top-k	5000
Nodes	4
Ranks/node	4

Tree merge update: Object communication to explicit buffers

Original mpi_tree

Sender branch

```
comm.send(scores, dest=partner, tag=0)  
comm.send(indices, dest=partner, tag=1)
```

Receiver branch

```
scores = comm.recv(source=partner, tag=0)  
indices = comm.recv(source=partner, tag=1)
```

⇒ Receive side appears as MPI_Mprobe / object-message handling.

Updated mpi_tree_explicit

Sender branch

```
scores = np.ascontiguousarray(current_scores)  
indices = np.ascontiguousarray(current_indices)
```

```
comm.Send(scores, dest=partner, tag=0)  
comm.Send(indices, dest=partner, tag=1)
```

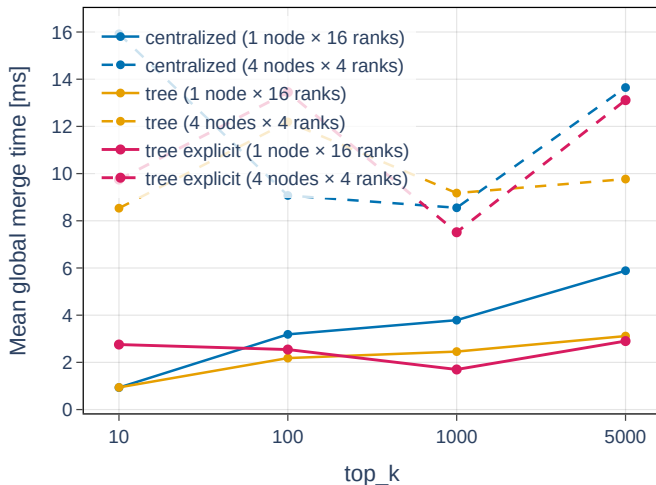
Receiver branch

```
recv_scores = np.empty_like(scores)  
recv_indices = np.empty_like(indices)
```

```
comm.Recv(recv_scores, source=partner, tag=0)  
comm.Recv(recv_indices, source=partner, tag=1)
```

⇒ Trace shows direct MPI_Send / MPI_Recv pairs.

Merge stress with increasing top-k: Merge timings after update



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10 – 5000
Backend	FAISS IndexFlatIP

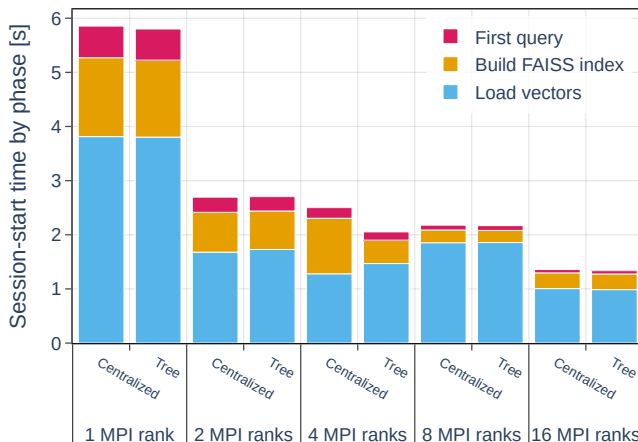
Cluster Setup

Nodes	1, 4
Ranks/node	16, 4
CPUs/rank	1
FAISS thr./rank	1
RAM/node	16 GB (4×4), 32 GB (1×16)

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	mean
Centralized	gather + rank 0 merge
(Explicit) Tree	full tree reduction

Session-start / cold-ish retrieval benchmark



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

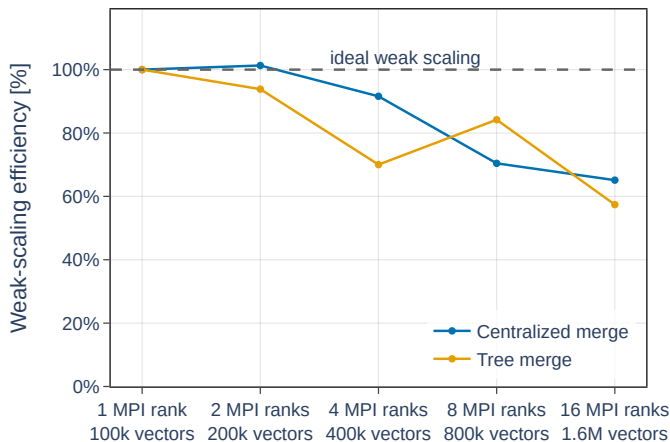
Cluster Setup

Nodes	1
Ranks/node	1-16
CPUs/rank	1
FAISS thr./rank	1
RAM/node	32 GB

Measurement

Timed phase	cold retrieval
Warmup runs	0
Measured runs	5 jobs per setting
Reported	median per phase
Total jobs	50 independent Slurm jobs

Weak scaling: Efficiency vs. MPI ranks



Weak-scaling configuration: fixed 100k vectors per rank

Configuration

Dataset	100k – 1.6M vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

Cluster Setup

Nodes	1
Ranks	1–16
CPUs/rank	1
FAISS thr./rank	1
RAM/node	8 GB – 64 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	7.7%–18.1%

Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution
- 3 Parallelized Algorithm
- 4 Benchmark Results
- 5 Conclusion**
- 6 Appendix

Possible improvements and future work

- **Startup overhead:** Vector-data loading dominates time-to-first-query.
 - ⇒ Store pre-sharded vector files, so each rank can directly load its assigned shard.
- **Algorithmic limit:** Exact flat search still compares the query with every vector, even when the scan is distributed.
 - ⇒ Test FAISS IVF or HNSW indexes to avoid full scans, then evaluate the speedup/quality trade-off with latency and recall@ k .

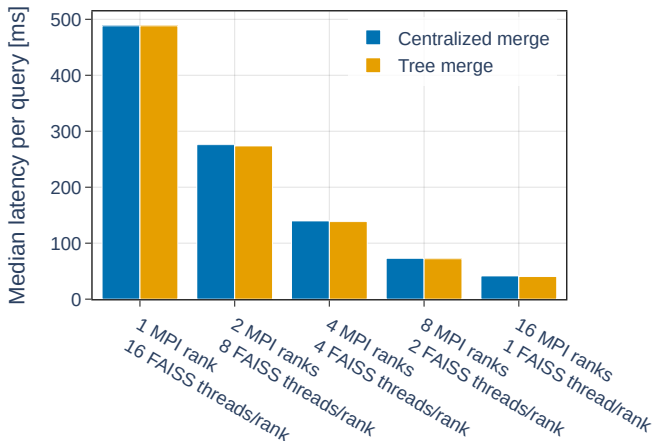
Conclusion: Lessons for RAG-style retrieval

- For realistic top- $k = 10$, local search dominates and MPI communication remains small.
- Time-to-first-query is affected mainly by vector loading and FAISS index construction.
- MPI sharding scales well for exact retrieval: Latency drops strongly as more ranks search smaller index shards.
- MPI makes the same retrieval framework usable on one node or across multiple nodes.

Outline

- 1 RAG Retrieval as Exact Top-k Search
- 2 Sequential Solution
- 3 Parallelized Algorithm
- 4 Benchmark Results
- 5 Conclusion
- 6 Appendix**

MPI ranks vs. FAISS threads trade-off: Latency vs. configuration



Allocation of fixed 16 Slurm-CPU budget

Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

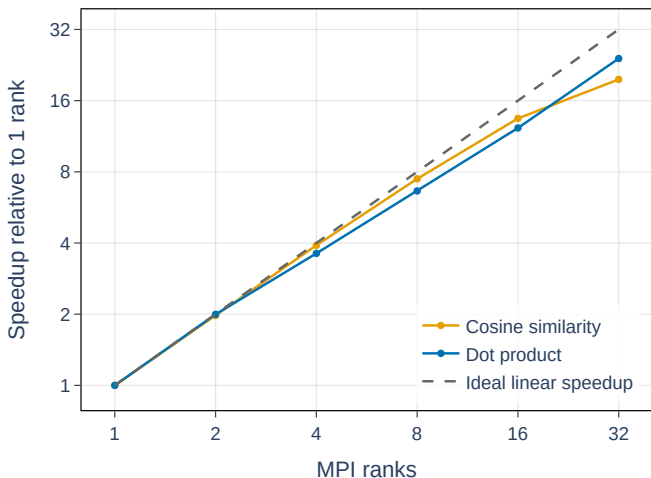
Cluster Setup

Nodes	1
Total CPU budget	16
Ranks/node	1–16
CPUs/rank	16–1
FAISS thr./rank	16–1
RAM/node	32 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Run-to-run CV	6.5%–19.8%

Hot latency strong scaling: Dot product vs. cosine similarity



Configuration

Dataset	500k vectors (ℓ_2 -normalized)
Dimension	2560
Queries/run	1
top-k	10
Backend	FAISS IndexFlatIP

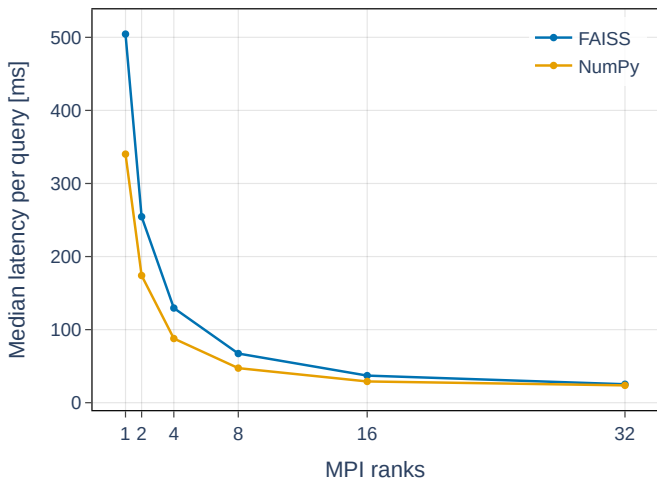
Cluster Setup

Nodes	1
Ranks	1–32
CPUs/rank	1
FAISS thr./rank	1
RAM/node	32 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Note	centralized merge was used

Backend comparison: FAISS IndexFlatIP vs. NumPy dot product



Configuration

Dataset	500k vectors
Dimension	2560
Queries/run	1
top-k	10
Backends	FAISS, NumPy

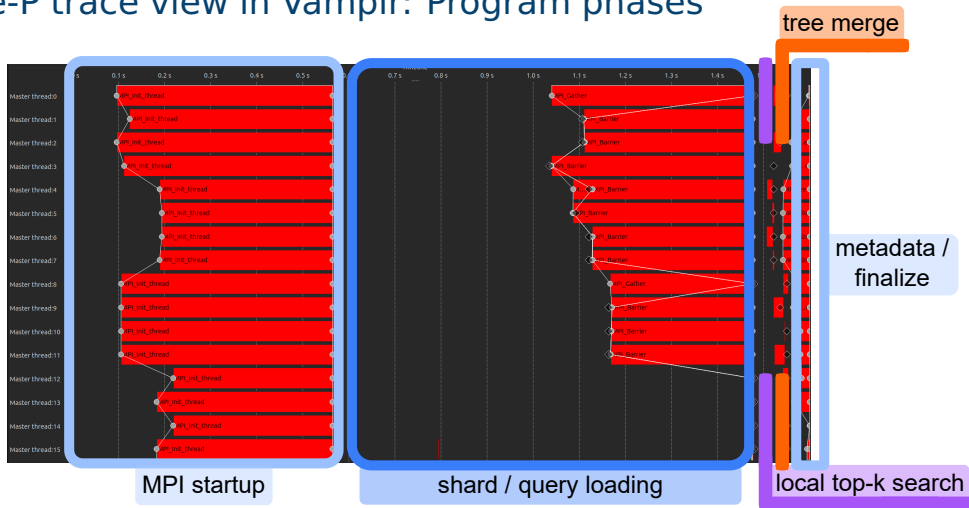
Cluster Setup

Nodes	1
Ranks	1–32
CPUs/rank	1
FAISS thr./rank	1
OpenBLAS thr./rank	1
RAM/node	32 GB

Measurement

Timed phase	hot retrieval
Warmup runs	5
Measured runs	50
Reported	median
Note	centralized merge was used

Score-P trace view in Vampir: Program phases



Trace of MPI processes for one query retrieval (tree merge, top-k=5000, NumPy backend, 4 nodes × 4 ranks).