ABUMPIMP 2024

PIM Lucene

Copyright UPMEM® 2024

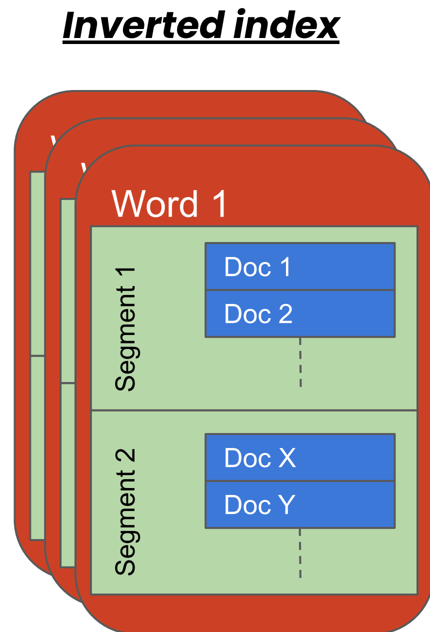
Lucene Primer

Copyright UPMEM® 2024

What is Index Search ?

- An **index search engine** identifies items in a text database that correspond to keywords specified by the user (web pages, text documents, e-commerce product …)
- An **inverted index** is built and maintained in order to answer queries with low latency / high throughput
- **Apache Lucene** is a powerful and popular open-source search library written in Java
- **UPIS**: Engine for exact phrase match: **> 600 queries/sec** with full PIM server on wikipedia

Inverted index



Copyright UPMEM® 2024

up mem

Lucene for DPU

Copyright UPMEM® 2024

What is PIM Lucene ?

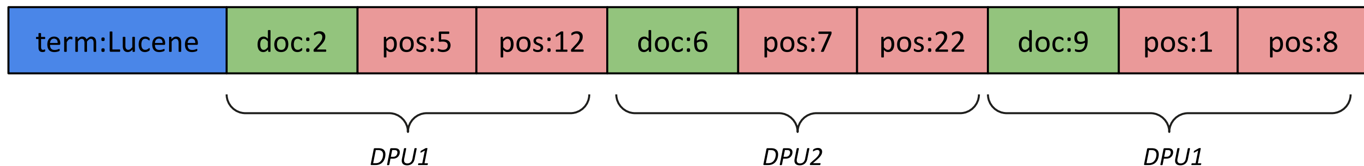
- PIM Lucene is a project to create an extension of Lucene to offload specific queries to UPMEM PIM
- Public on github: https://github.com/upmem/pim-lucene
- The objectives are :
  - Create a **non-intrusive** extension of the Lucene code base
  - Provide an option to use PIM for specific query types or part of the query execution
  - Provide better query throughput / lower energy consumption
  - Do not impact regular Lucene's functionality or performance (it is an extension)
  - The first query being implemented is the **PhraseQuery**

Copyright UPMEM® 2024

up
mem

PIM Lucene Index Architecture (1)

- Our design choice is to **keep the original Lucene index untouched**
- A new index specific to PIM is stored in the PIM memory
- A ***PIMIndexWriter*** is the interface for writing a Lucene index that also contains the PIM index
  - First create the normal Lucene index
  - Then read the Lucene index and create the PIM index
  - The postings of each term are spreaded over the DPUs based on the document ID
  - Each DPU contains a subset of documents

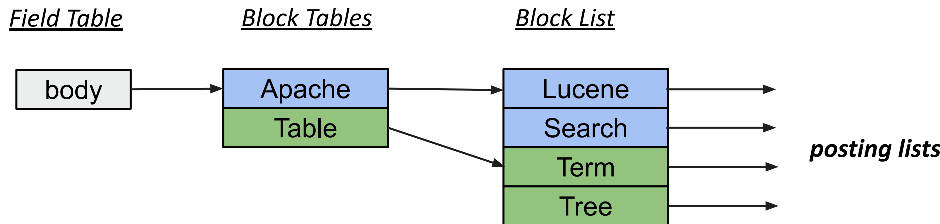


Copyright UPMEM® 2024

upmem

PIM Lucene Index Architecture (2)

- The PIM index stores the **term bytes sorted** (difference with UPIS project)
  - This will enable to handle prefix or fuzzy queries
- The PIM index for one DPU consists in four parts:
  - A field table that associates to each field the address where to find the field's term block table
  - A list of term block table for each field, used to find the block where a particular term should be searched
  - A block list where each block is a list of terms of a small and configurable size meant to be scanned linearly
  - The postings lists (delta-encoded)
- Ex with one field “body” containing 6 terms “Apache”, “Lucene”, “Search”, “Table”, “Term”, “Tree”, block size=3

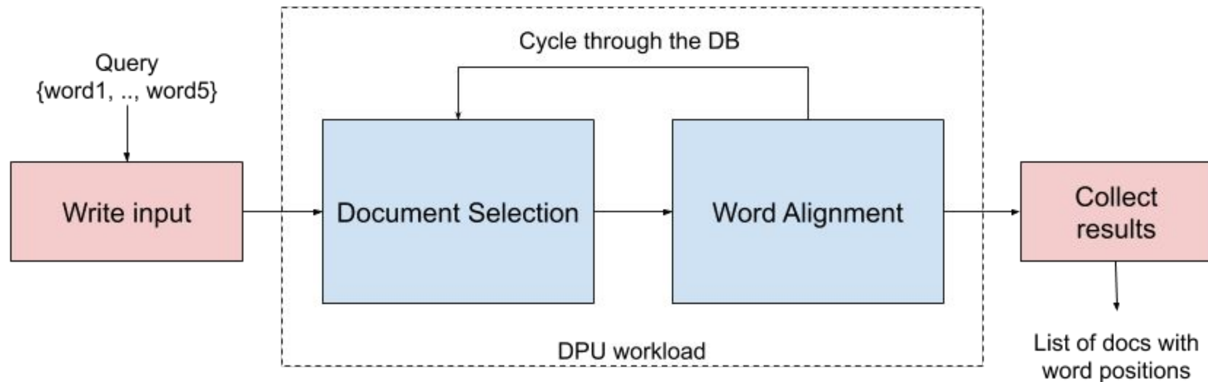


Copyright UPMEM® 2024

up mem

Searching Consecutive Words

1. Document selection: Find next DID where all words from the query are present
2. Word alignment: Check if words are found consecutively in this DID
3. Repeat the process until all documents have been looked into

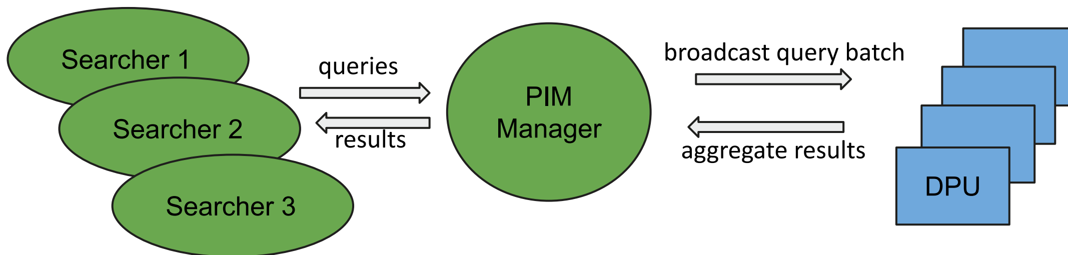


Copyright UPMEM® 2024

upmem

PIM Lucene Query Architecture

- Create a specific query class in Lucene for each query to be executed on PIM
  - For instance **PimPhraseQuery** object
- The system may decide to run the query on PIM or with standard Lucene index
- A global PIM manager collects the queries coming from different searcher threads
- Queries are sent as a batch to the DPUs and results are collected => [docId, freq]
- The CPU uses the freq and the norm (stored on disk) to compute the score



Copyright UPMEM® 2024

upmem

Code Example : Indexing

Create a PimIndexWriter, the PimConfig specifies PIM system parameters

```
Analyzer analyzer = new StandardAnalyzer();
Directory indexDirectory = new MMapDirectory(Paths.get(index));
Directory pimIndexDirectory = new MMapDirectory(Paths.get(index + "/dpu"));
IndexWriterConfig iwc = new IndexWriterConfig(analyzer);
// provide a directory for pim index and a pim config to the PimIndexWriter constructor
IndexWriter writer = new PimIndexWriter(indexDirectory, pimIndexDirectory, iwc, new PimConfig(nbDpus, 16));
```

Closing the writer will commit the standard Lucene index and create the PIM index

```
writer.close();
```


Copyright UPMEM® 2024

up mem

Code Example : Search

Create index reader, open and load the PIM index

```
IndexReader reader = DirectoryReader.open(MMapDirectory.open(Paths.get(index)));
IndexSearcher searcher = new IndexSearcher(reader);
// load PIM index from PIM directory
PimSystemManager.get().loadPimIndex(MMapDirectory.open(Paths.get(index + "/dpu")));
```

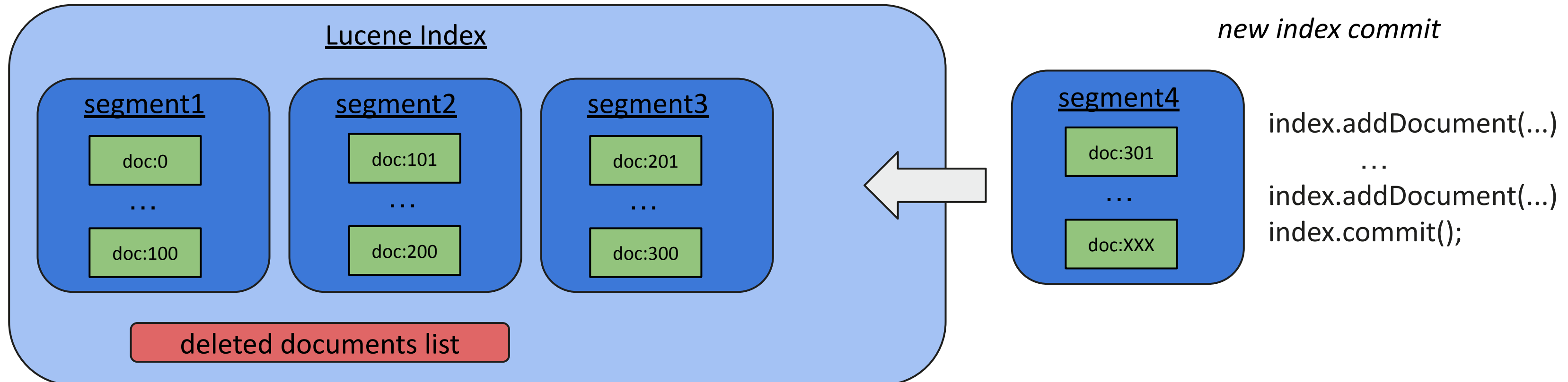
Build the PimPhraseQuery and search the index

```
PimPhraseQuery.Builder builder = new PimPhraseQuery.Builder();
/* add terms */
builder.add(new Term("body", "Apache"), 0);
builder.add(new Term("body", "Lucene"), 1);
PimPhraseQuery query = builder.build();
/* search */
TopDocs results = searcher.search(query, 100);
```


Copyright UPMEM® 2024

Lucene Segments

- Lucene uses the concept of index segments which allows for dynamic updates of the index
- Each segment is the index for a subset of all documents (postings, norms etc.)
- An index commit creates a new segment which contains the documents added for the commit
- A segment is immutable, deleted documents are marked as such but the data stays in the index
- Periodically, segments are merged and documents actually deleted

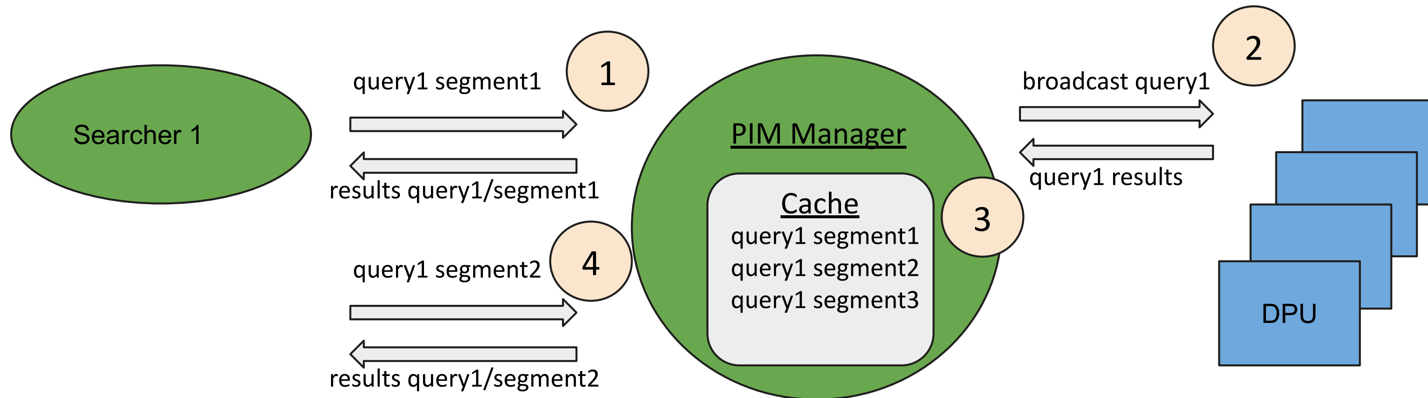


Copyright UPMEM® 2024

up mem

Lucene Segments Support for DPU

- Our design choice is **not to support incremental updates** of the PIM index for now
  - After each commit the complete PIM index is re-generated
  - This is sufficient for scenarios where the index updates are infrequent
- Still, high-level Lucene APIs expects the results to be provided on a per segment basis (in order)
  - Modifying this would mean significantly changing Lucene's architecture
- Solution: Results of all segments are **cached** and returned on demand on a per-segment basis

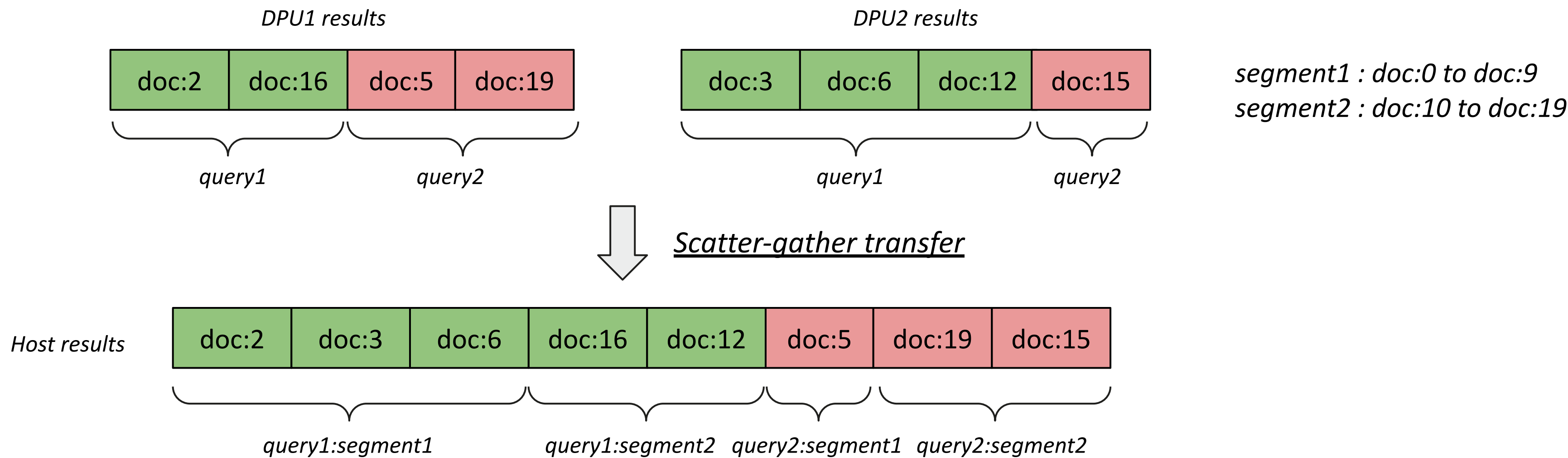


Copyright UPMEM® 2024

up mem

Scatter-Gather Transfer of Results

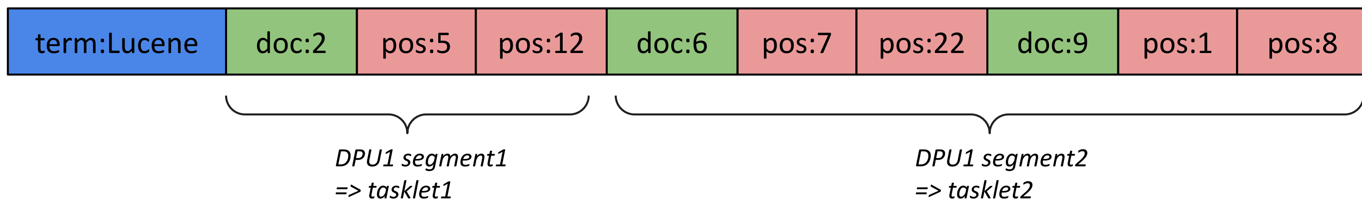
- Results on the host are **scattered**: each DPU provide results for different queries and segments
- We need to return to each search thread a reader for results of a particular query/segment
- It is possible to have all readers make “scattered” reads from the same buffer but better is to use scatter-gather transfers



Copyright UPMEM® 2024
up mem

Tasklets Load Balancing

- The work on a DPU needs to be parallelized between different tasklets
- A straightforward way is to **parallelize over queries**: each tasklet handles a different query
  - Does not work well for small batches of size less than the number of tasklets
  - The amount of work can greatly varies between queries => bad load balancing
- A better solution is to create "**PIM-index segments**" which are defined by a document id range
- Each tasklet then searches a segment of each query
- Small overhead in PIM-index to store DPU index skip information (delta-encoded positions)



10 documents in index
DPU segment1 range=1..5
DPU segment2 range=6..10

Copyright UPMEM® 2024

up mem

Benchmarks

Copyright UPMEM® 2024

Benchmarking Setup

UPMEM Benchmark Server (PIM+CPU vs. CPU compute)		
Component	**Details**	**QTY (PIM)**
CPU	Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, 16 cores, 22MB cache, 100W TDP	2
RAM	Samsung 32GiB 2400 MHz M393A4K40CB2-CTD DIMM	8
PIM Memory	DDR4-2400 PIM Module (16 x 512MB/8DPU) @ 350MHz	16

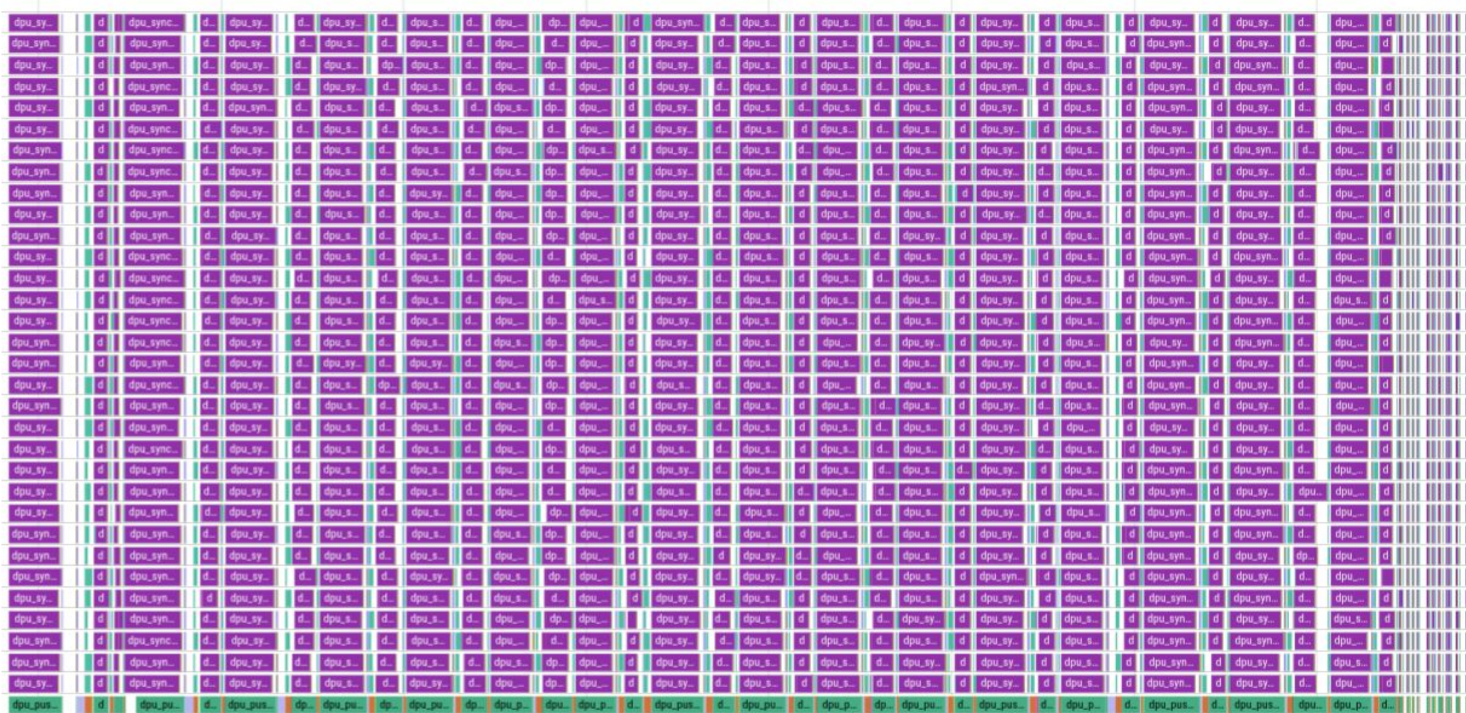
English wikipedia dataset - 20M+ files - over 240GB of text

1036 queries extracted from Lucene nightly benchmarks

Copyright UPMEM® 2024

up
mem

Profiling View (64 Search threads)



DPU execution is in purple => it takes the largest fraction of the time

Load balancing OK

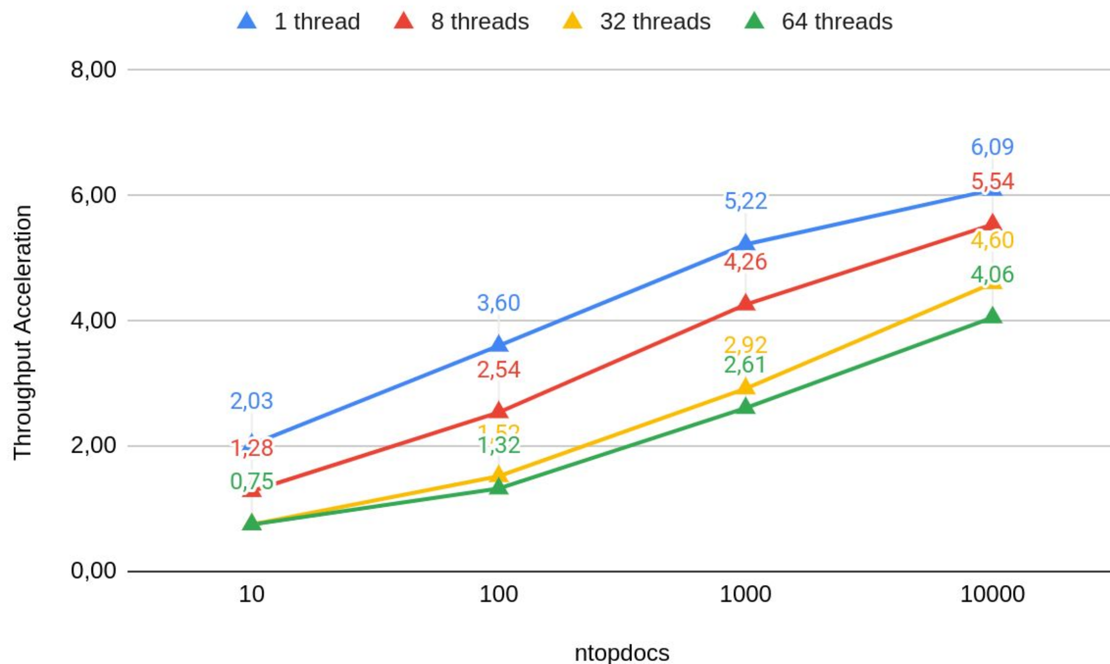
The time to read and score the results is shadowed by DPU time

Batch size still varies

Copyright UPMEM® 2024

upmem

PIM-Lucene PhraseQuery Speedup

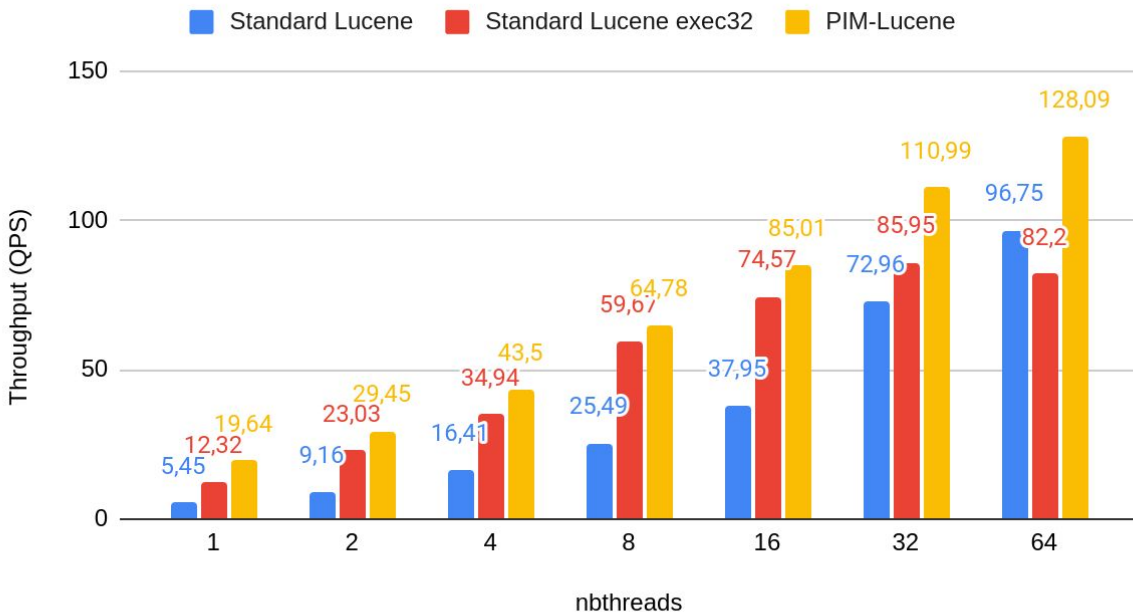


Two main observations

- Speedup increases with number of top docs
- Speedup decreases with number of searcher threads

Copyright UPMEM® 2024

Parallelism with Searcher threads

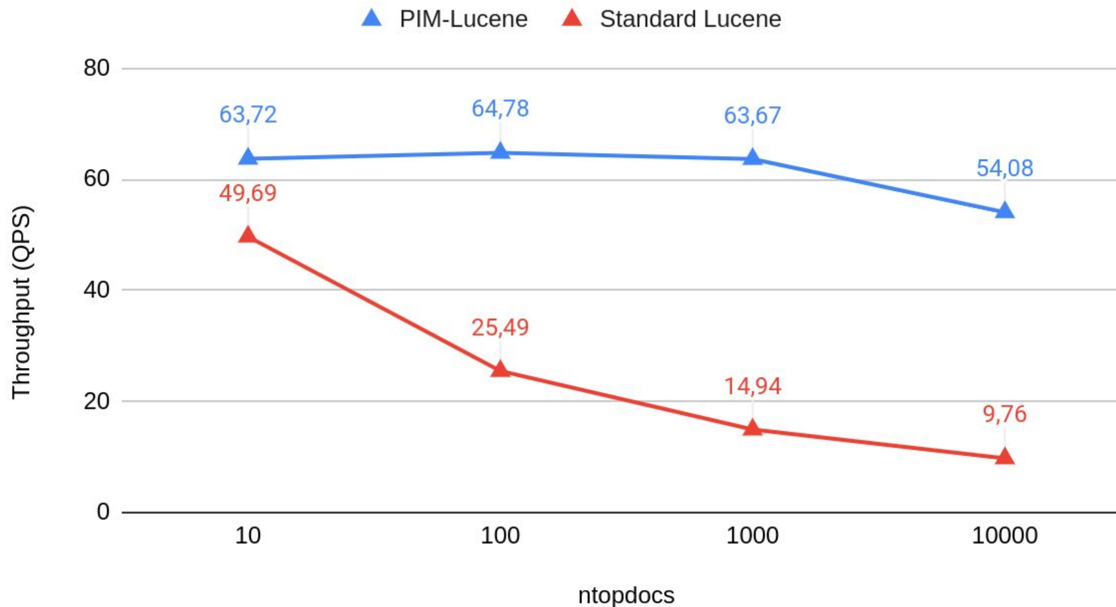


- When the number of threads is low, parallelism on one query makes a difference
- When using query-level parallelism in Lucene (red bars), the throughput is better for low number of threads

Copyright UPMEM® 2024

up mem

Performance vs Number of Topdocs

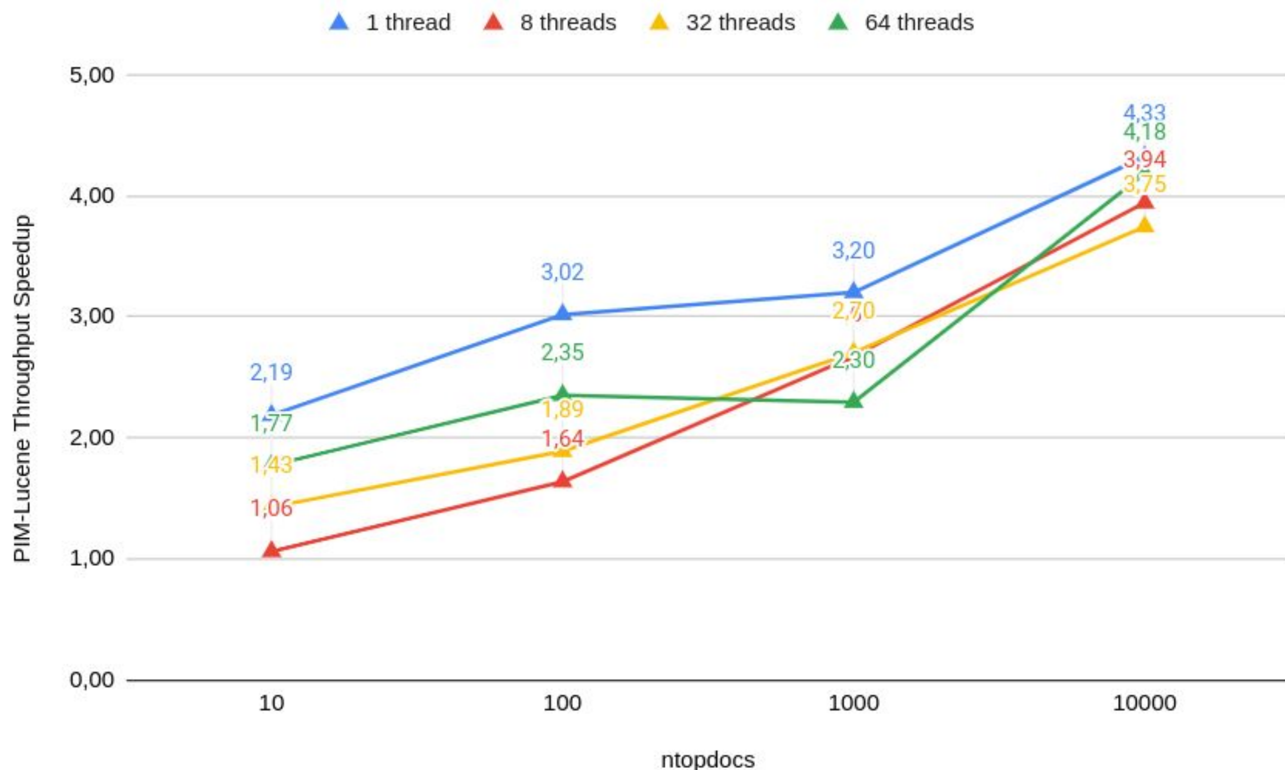


- Lucene shows a good acceleration when the number of top docs decreases
- This is because it uses a lower bound to avoid checking all documents
- **<u>Next step</u>: implementing the lower bound on DPU**

Copyright UPMEM® 2024

up
mem

Lower Bound Synchronization



- This makes use of the Parallel WRAM access feature
- Host maintains a heap of the best results
- Acceleration with low number of topdocs has been smoothened

Copyright UPMEM® 2024

up mem