News Search
- Mini IR project

by Isha Aggarwal

SECTIONS IN THE PRESENTATION

- Project idea
- Implementation overview
- Observations and Results

Project idea and detailed working

Brief Overview

- The project – “News Search” implements:
  - Clustering of news documents based on different zones
  - Exact top k weighted zonal ranked retrieval
  - Cluster based weighted zonal ranked retrieval
- Dataset: over 1 lakh news articles
- Implementation from scratch using C++, Openmp and CUDA

Dataset and document format

- Dataset:
  - "All the news" from Kaggle
  - News articles from around 15 American publications
  - Original format- .csv
  - Used a subset of it – 100093 docs

title:

publication:

author:

date:

year:

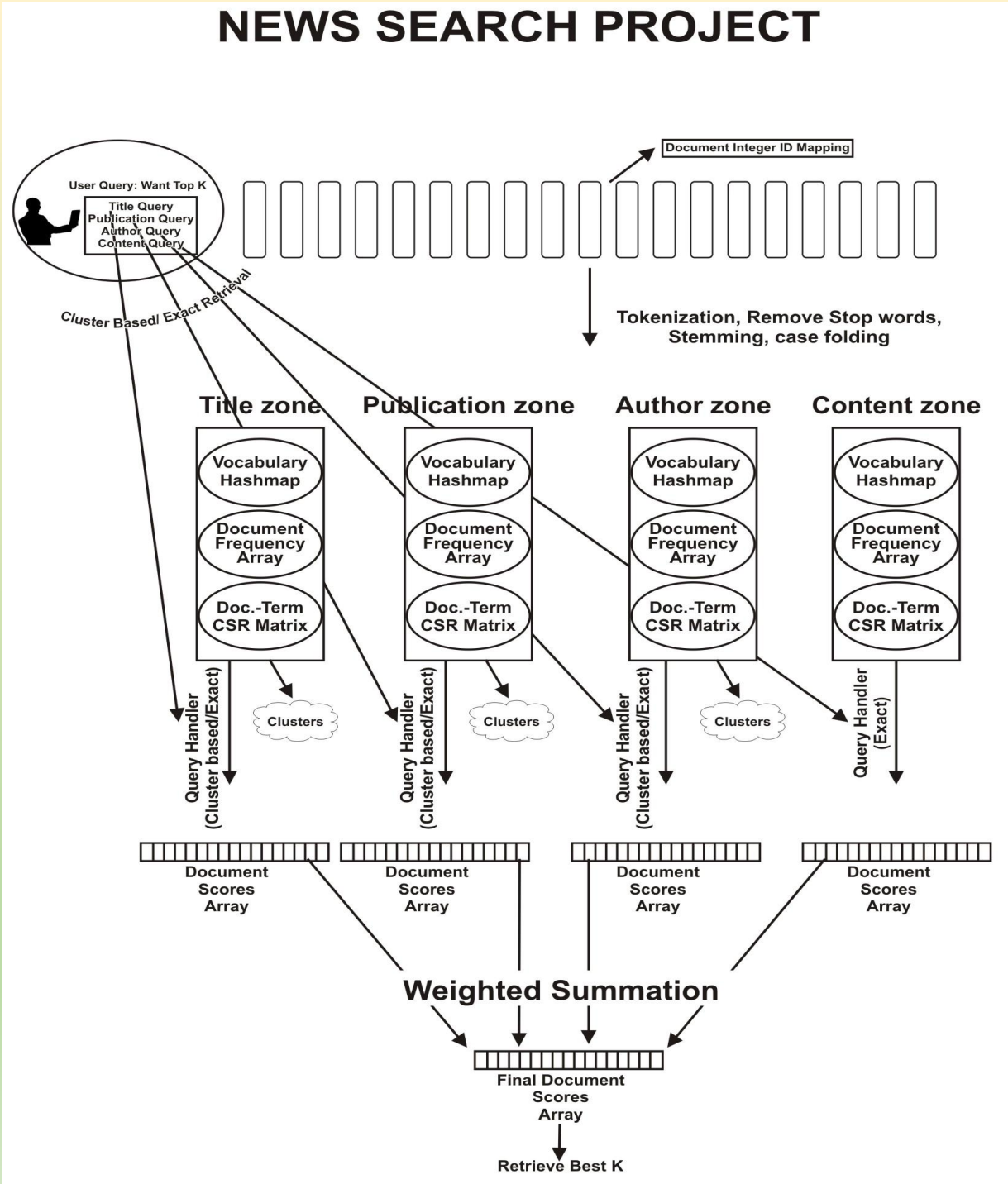
month:

url:

content:

- Document format used:
  - Separated .csv to different .txt files by writing a python converter
  - Zones taken into account for retrieval purpose: title, publication, author, content
  - Document names mapped to integer ids (0 to 100093) for ease of reference

Flow diagram



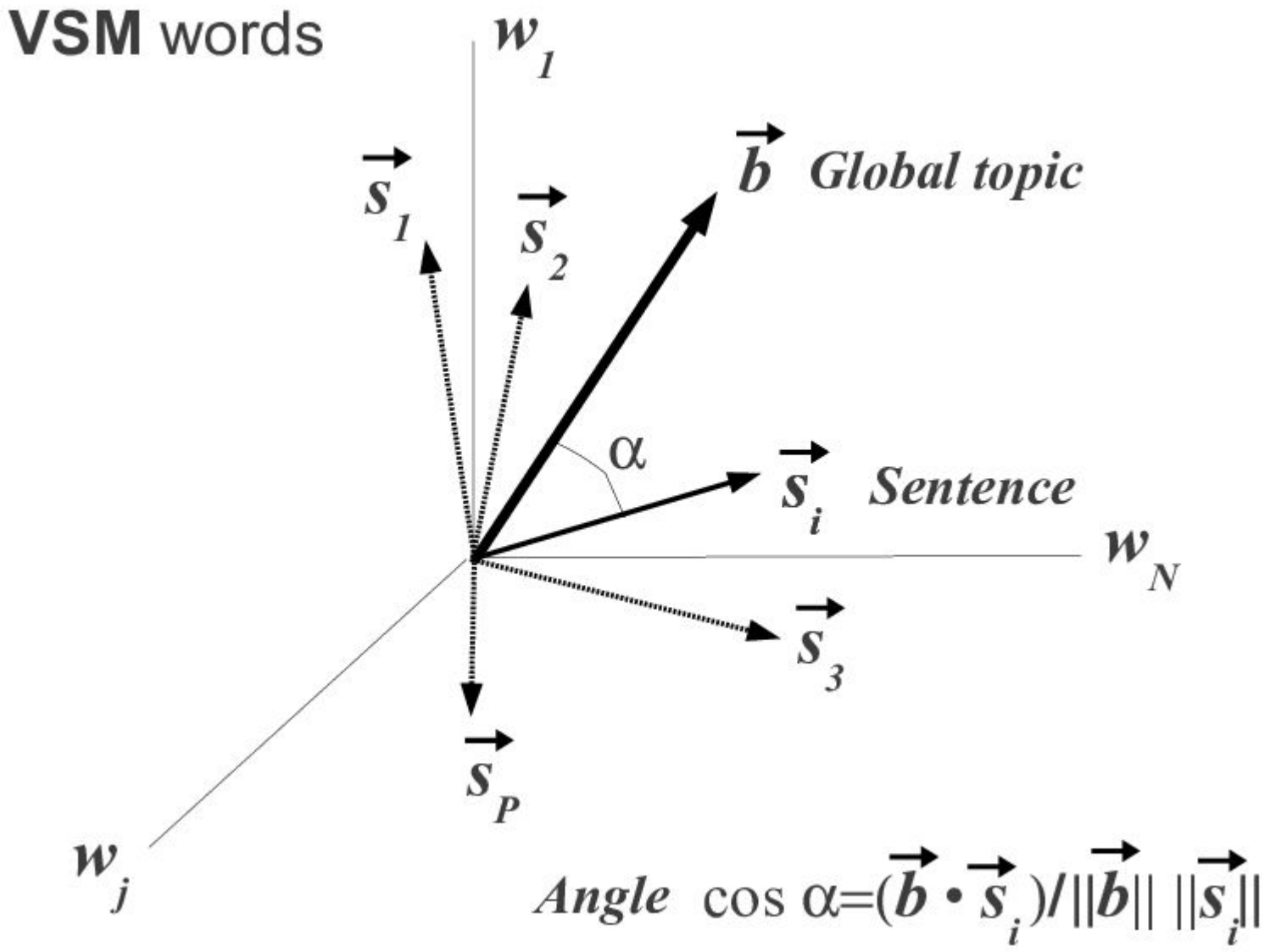
Tokenization, Case folding Stop words removal and Stemming

- Implemented Tokenizer using concept of regex
- Converted all word to lower case
- Removed stop words; used list by Gerard Salton from Cornell University
- Used Porter Stemmer for conversion to root form
- Stored information after this pre processing step for all 4 zones.
- Note: Avoided stemming and stop word removal for author and publication zone to retain exact names.

Zonal structures/Doc info storage

- Separate for each zone, stored everything in main memory
- Vocabulary
  - Hash map
  - Keys - unique terms
  - Values – term index - unique integer index (0 to vocab length)
- Doc-frequency array
  - Store doc-frequency for each word at array index = term index.
- Document – term matrix
  - i th row storing info about the doc having id = row index
  - j th column for term having term index/value = j in vocab hash map
  - (i,j) cell stores term frequency of term j in document i
  - Matrix stored in CSR format as it is a sparse matrix, and it can be fit into the main memory using CSR storage format.

Vector Space Model



Weighting scheme used:
log tf idf

weight for word W_i in doc-vector$_j$ = $(1+\log(tf_{i,j})).\log(N/doc\_freq_i)$

(In case tf = 0, weight = 0)

$tf_{i,j}$ = number of times the word W_i appears in doc j.

N = total number of docs in dataset

doc_freq_i = number of docs which contain W_i

Clustering

- Spherical k-means clustering algorithm
- Documents as document vectors following concept of vector space model; log tf-idf weighting scheme
- Clustered for zones- title, author and publication
- Also, implemented elbow method. Outputs dissimilarity score for num_clusters : 1 to 7, then user can choose a suitable number of clusters acc. to elbow point in the graph.

Exact top k retrieval

- Input free text queries from user for zone-title, author, publication and content
- Compute cosine similarity scores for all docs,zone_query pair for each of the zones
- Weighted summation of scores for corresponding documents
- Thus, obtained final set of scores for the user query
- Retrieve best k docs (in descending order of scores) – Ranked zonal retrieval

Cluster based retrieval

- Form of inexact top k retrieval
- Speeds up the retrieval process by reducing the number of docs to be matched with the zone query
- For each zone, found the cluster with the highest similarity to the zone query, and match with the docs only in that cluster. Allocate score 0 to all other docs for that zone query– this introduces inexactness
- For zone- content, simply compute scores for all doc,content_query pairs as did not cluster on basis of zone-content.
- Weighted summation of scores for corresponding documents
- Thus, obtained final set of scores for the user query
- Retrieve best k docs (in descending order of scores) – Ranked zonal retrieval

Implementation overview

- Openmp Implementation
- Cuda Implementation

Storage...

In both the cases,for each zone, the documents are stored in CSR matrix format.

Centroids and queries are taken as full vectors.

Openmp Implementation...

Parallelized parts of:

- cosine similarity score computation
- Document, centroid and query norms
- spherical k-means clustering (initializing seeds, assigning docs to clusters, recomputing centroids)
- query-handler(both cluster based and exact)
- elbow method (total within cluster dissimilarity)

Openmp Implementation...

Example: seeds initialization

```
18
19
20      void zone::initialize_centroids()
21      {
22          //get K seed points
23          //int array[num_clusters];
24          int* array = (int*)malloc(num_clusters*sizeof(int));
25          for(int i=0;i< num_clusters ;i++)
26          {
27                  array[i] = i;
28          }
29
30          //initialize centroids
31          #pragma omp parallel for //all local variables implicitly private
32          for(int i=0;i< num_clusters;i++)
33          {
34              int doc_index = array[i];
35              int start_ind_col_arr = matrix.row_ptrs[doc_index];
36              int end_ind_col_arr = matrix.row_ptrs[doc_index + 1];
37
38              #pragma omp parallel for
39              for(int j = start_ind_col_arr; j < end_ind_col_arr;j++)
40              {
41                  int word_ind = matrix.col_ind[j];
42                  centroids[i][word_ind] = 0;
43                  int tf = matrix.term_freq[j]; //log tf idf actually
44                  if(tf == 0)
45                      centroids[i][word_ind] = 0;
46                  else
47                  {
48                      centroids[i][word_ind] =  ( 1 + std::log10(double(tf)) )*std::log10(matrix.rows /doc_freq[word_ind]);
49
50                  }
51
52              }
53
54
55          }
56
57          free(array);
58
59      }
60
```

Cuda Implementation

CUDA kernels for parts of:

- k-means clustering
  - initializing seeds
  - assigning docs to clusters
  - recomputing centroids
- Document, centroid norms
- elbow method(total within cluster dissimilarity)
- query handler
  - cluster based
  - exact top k retrieval
- Scores summation

seeds initialization kernel

one warp per row of the csr matrix (or to initialize a seed)

All seeds are initialized in parallel.

```
370
371  __global__ void initialize_centroids_kernel( int num_clusters,int mat_rows, int mat_cols, int* mat_row_ptrs,
372   int* mat_col_ind, int* mat_term_freq, float* centroids, int* doc_freq)
373  {
374      int gid = blockDim.x * blockIdx.x + threadIdx.x;
375
376      int warp_index = (int)(gid / WARP_SIZE);  //(32 is warp size)
377
378      int centroid_index = warp_index;
379
380      if(centroid_index < num_clusters)
381      {
382          int doc_ind = centroid_index;
383
384          int start_ind_for_doc = mat_row_ptrs[doc_ind];
385
386          int end_ind_for_doc = mat_row_ptrs[doc_ind + 1];
387
388          int id_within_warp = gid % WARP_SIZE;
389
390          for(int k = start_ind_for_doc + id_within_warp; k < end_ind_for_doc; k = k + 32)
391          {
392              int tf = mat_term_freq[k];
393              int word_ind = mat_col_ind[k];
394
395              float weight = ( 1 + __log10f(tf))*(__log10f(mat_rows/doc_freq[word_ind]));
396
397              centroids[centroid_index*mat_cols + word_ind] = weight;
398          }
399      }
400  }
401
```

doc norms kernel

One warp per doc(row in csr matrix), then warp level reduction.

```
141
142 __global__ void compute_doc_norms_kernel(int mat_rows, int mat_cols, int* mat_row_ptrs, int* mat_col_ind,
143     int* mat_term_freq, int* doc_freq, float* doc_norms)
144 {
145     int gid = blockDim.x * blockIdx.x + threadIdx.x;
146
147     int warp_index = (int)(gid / WARP_SIZE);  //(32 is warp size)
148
149     int doc_ind = warp_index;
150
151     if(doc_ind < mat_rows)
152     {
153         int id_within_warp = gid%32;
154
155         int start_col_ind_for_doc = mat_row_ptrs[doc_ind];
156         int end_col_ind_for_doc = mat_row_ptrs[doc_ind + 1];
157
158
159         float temp = 0;
160
161         for(int a = start_col_ind_for_doc + id_within_warp; a < end_col_ind_for_doc; a = a + 32)
162         {
163             int tf = mat_term_freq[a];
164             int word_ind = mat_col_ind[a];
165             float weight = (1 + __log10f(tf))*(__log10f(mat_rows/doc_freq[word_ind]));
166             temp += weight*weight;
167         }
168
169         float val = temp;
170
171
172         for (int offset = 16; offset > 0; offset /= 2)
173         {
174             val += __shfl_down_sync(FULL_MASK, val, offset);
175         }
176
177         if(id_within_warp == 0)
178         {
179             doc_norms[doc_ind] = sqrt(val);
180         }
181
182     }
183 }
184
```

centroid norms kernel

Norms of all centroids are computed in parallel.

-- Used 2 D grid.

-- For centroid i , all threads with blockidx.y = i are used.

-- for each individual centroid, partial results are written into shared memory and then block reduce is called.

```
489
490  void zone::Compute_centroid_norms()
491  {
492
493      dim3 block(THREADS_PER_BLOCK);
494      int work_per_thread = 4;
495      int gridsize_x = ceil((double)matrix.cols / (double)(THREADS_PER_BLOCK * work_per_thread));
496      dim3 grid(gridsize_x,num_clusters);
497
498      float* gpu_buffer;
499      int buffer_size = gridsize_x*num_clusters;
500      cudaMalloc((void**)&gpu_buffer, buffer_size * sizeof(float));
501
502      dim3 grid2(1,num_clusters);
503
504      centroid_norms_kernel1 <<< grid , block >>> (matrix.cols, num_clusters , centroids_gpu, gpu_buffer);
505      centroid_norms_kernel2 <<< grid2 , block >>>(gridsize_x, num_clusters, gpu_buffer, centroid_norms_gpu);
506
507      cudaFree(gpu_buffer);
508
509  }
510
```

```
434
435
436  __global__ void centroid_norms_kernel1( int mat_cols,int  num_clusters ,float* centroids,float* buffer )
437  {
438      int dimgrid_x = gridDim.x;
439      int dimblock = blockDim.x;
440
441      int lid_x = threadIdx.x;
442      int gid_x = blockDim.x * blockIdx.x + threadIdx.x;
443
444      float tmp = 0;
445
446      int block_id_y = blockIdx.y;
447
448      for (int i = gid_x; i < mat_cols; i += dimgrid_x * dimblock)
449      {
450          tmp += centroids[block_id_y*mat_cols + i]*centroids[block_id_y*mat_cols + i];
451      }
452
453      __shared__ float tmp_work[THREADS_PER_BLOCK];
454      tmp_work[lid_x] = tmp;
455
456      __syncthreads();
457      block_reduce(tmp_work);
458
459      if (lid_x == 0)
460          buffer[ dimgrid_x*block_id_y +  blockIdx.x] = tmp_work[0];
461  }
462
463
```

centroid norms

kernel continued..

```
__global__ void centroid_norms_kernel2(int size_x ,int num_clusters, float* buffer,float* centroid_norms)
{
    int nt = blockDim.x;
    int tid = threadIdx.x;

    int block_id_y = blockIdx.y;

    float temp = 0;
    for(int i = tid; i < size_x; i += nt)
    {
        temp += buffer[block_id_y*size_x + i ];
    }

    __shared__ float tmp_work[THREADS_PER_BLOCK];
    tmp_work[tid] = temp;

    __syncthreads();
    block_reduce(tmp_work);

    if(tid == 0)
        centroid_norms[block_id_y] = sqrt(tmp_work[0]);
}
```

```
__device__ void block_reduce(float* data)
{
    int nt = blockDim.x;
    int tid = threadIdx.x;

    for (int k = nt / 2; k > 0; k = k / 2)
    {
        __syncthreads();
        if (tid < k)
        {
            data[tid] += data[tid + k];
        }
    }
}
```

RESULTS Folder:

As the code is executed,num_clusters and query is taken as input from the user and several files are produced.

Doc-names-index-MAP.txt

doc_term_csr_matrix_zone-TITLE.txt
doc_term_csr_matrix_zone-PUBLICATION.txt
doc_term_csr_matrix_zone-AUTHOR.txt
doc_term_csr_matrix_zone-CONTENT.txt

Elbow_Method_zone-TITLE.txt
Elbow_Method_zone-PUBLICATION.txt
Elbow_Method_zone-AUTHOR.txt

clustering_results_zone-TITLE.txt
clustering_results_zone-PUBLICATION.txt
clustering_results_zone-AUTHOR.txt

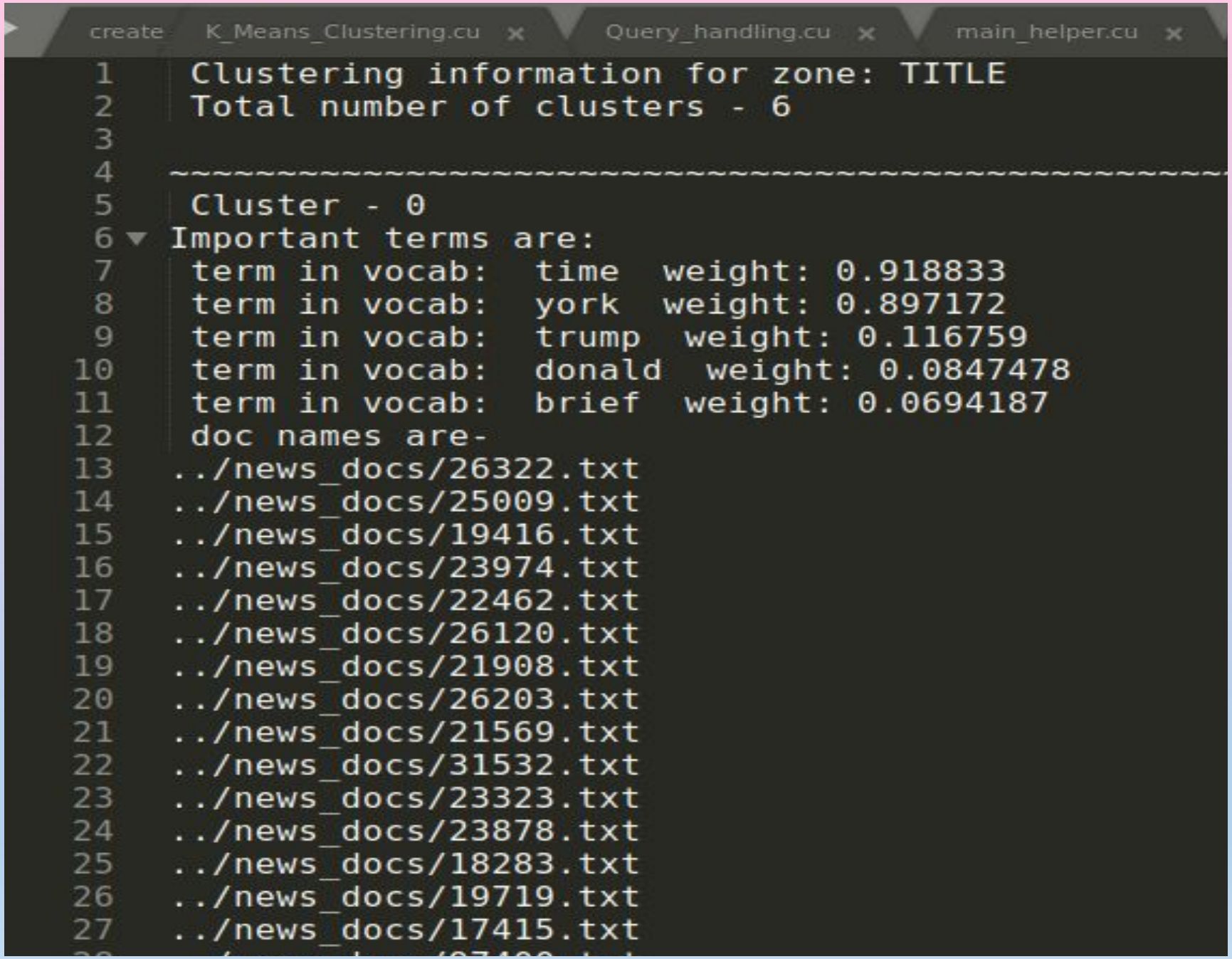
RETRIEVAL_RESULTS_round-0.txt
RETRIEVAL_RESULTS_round-1.txt
RETRIEVAL_RESULTS_round-2.txt
.
.
.

Elbow method example:

K_Means_Clustering | Query_handling.cu | main_helper.cu | myheader.h | *Elbow_Method_zone-TITLE.txt* | main.cp

```
elbow method for zone: TITLE

--------------------------------------------------------------------------------
K value = 1
 average dissimilarity is: 94415.9

--------------------------------------------------------------------------------
K value = 2
 average dissimilarity is: 93361.9

--------------------------------------------------------------------------------
K value = 3
 average dissimilarity is: 93054

--------------------------------------------------------------------------------
K value = 4
 average dissimilarity is: 92070.8

--------------------------------------------------------------------------------
K value = 5
 average dissimilarity is: 91449.1

--------------------------------------------------------------------------------
K value = 6
 average dissimilarity is: 90942.5

--------------------------------------------------------------------------------
K value = 7
 average dissimilarity is: 90635.7
```

Clustering results example:



Retrieval results example:

Cluster based

K_Means_Clustering. Query_handling.cu × main_helper.cu × myheader.h × RETRIEVAL_RESULT

```
1  Cluster based retrieval
2
3   zone : TITLE, free text query is: The First lady
4   zone : PUBLICATION, free text query is: New York TIMES
5   zone : AUTHOR, free text query is: David
6   zone : CONTENT, free text query is: Melania Trump
7  Top 7 docs are :
8  doc  name: ../news_docs/25778.txt  score: 1.61145 out of 4.00
9  doc  name: ../news_docs/18044.txt  score: 1.42212 out of 4.00
10 doc  name: ../news_docs/25059.txt  score: 1.42192 out of 4.00
11 doc  name: ../news_docs/20467.txt  score: 1.41313 out of 4.00
12 doc  name: ../news_docs/21451.txt  score: 1.41313 out of 4.00
13 doc  name: ../news_docs/22815.txt  score: 1.41313 out of 4.00
14 doc  name: ../news_docs/24268.txt  score: 1.41313 out of 4.00
15  The retrieval time is :129 milliseconds
16
```

Exact Retrieval

K_Means_Clustering. Query_handling.cu × main_helper.cu × myheader.h × RETRIEVAL_RESULTS-

```
1  Exact top k Retrieval
2
3   zone : TITLE, free text query is: Trump visits India
4   zone : PUBLICATION, free text query is: CNN
5   zone : AUTHOR, free text query is:
6   zone : CONTENT, free text query is: India Narendra Modi National
7  Top 4 docs are :
8  doc  name: ../news_docs/61213.txt  score: 1.58874 out of 4.00
9  doc  name: ../news_docs/58637.txt  score: 1.56299 out of 4.00
10 doc  name: ../news_docs/60771.txt  score: 1.55294 out of 4.00
11 doc  name: ../news_docs/60059.txt  score: 1.53572 out of 4.00
12  The retrieval time is :130 milliseconds
13
```

Observations and Results

Correctness check...

- Compared the results from plain sequential, openmp and cuda implementation.
- Manually opened the retrieved documents etc...

Vocabulary size - Full dataset

- Dataset: 100093 docs (414 MB)

- Vocabulary size:
  - Title zone: 38766
  - Publication zone: 35
  - Author zone: 6459
  - Content zone: 191184

Machine specifications

- cpu: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s)

- gpu: Nvidia GeForce MX130

Zonal structures creation time

- Document parsing and creation of zonal structures-vocabulary, doc-frequency array, csr matrix structure: for all zones jointly takes around 12-13 minutes - for full dataset.

Runtime Comparison

DATASET (Number of docs)	Implementation:	K Clustering- TITLE		K Clustering- PUBLICATION		K Clustering- AUTHOR		Cluster based retrieval	Exact top k retrieval
1000	Sequential	6	1241 millisec	6	12 millisec	7	213 millisec	175 millisec	179 millisec
	OpenMp		684 millisec		17 millisec		80 millisec	55 millisec	71 millisec
	CUDA		9 millisec		6 millisec		4 millisec	7 millisec	3 millisec
5000	Sequential	7	80.7 sec	6	44 millisec	2	237 millisec	1454 millisec	2000 millisec
	OpenMP		25.6 sec		76 millisec		86 millisec	371 millisec	415 millisec
	CUDA		53 millisec		3 miilisec		7 millisec	14 millisec	14 millisec

Runtime Comparison

DATASET (Number of docs)	Implementation:	K Clustering- TITLE		K Clustering- PUBLICATION		K Clustering- AUTHOR		Cluster based retrieval	Exact top k retrieval
10000	Sequential	6	4.57 minutes	6	73 millisec	2	615 millisec	4122 millisec	6065 millisec
	OpenMp		80.8 sec		115 millisec		245 millisec	1406 millisec	1175 millisec
	CUDA		91 millisec		6 millisec		2 millisec	40 millisec	20 millisec
25000	Sequential	6	impractical	6	---	2	---	---	21.634 sec
	OpenMP		very very slow		---		---	---	6.145 sec
	CUDA		348 millisec		6 millisec		3 millisec	45 millisec	51 millisec

Runtime Comparison

DATASET (Number of docs)	Implementation:	K Clustering- TITLE		K Clustering- PUBLICATION		K Clustering- AUTHOR		Cluster based retrieval	Exact top k retrieval
50000	Sequential	6	impractical	6	----	2	----	---	1.341 minutes
	OpenMp		impractical		----		----	---	16.390 sec
	CUDA		597 millisec		20 millisec		5 millisec	85 millisec	82 millisec
100093 docs(FULL DATASET)	**Sequential**	6	**impractical**	6	---	2	---	---	**2.66 min**
	OpenMP		**impractical**		---		---	---	**48 sec**
	CUDA		**1159 millisec**		**30 millisec**		**14 millisec**	**129 millisec**	**130 millisec**

Limitations...

For larger datasets, we will run out of memory.

References…

- An Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze –Chapter-16 : Flat clustering, Chapter-6: Scoring, term weighting and the vector space model

- Salton, Gerard. 1971a. Cluster search strategies and the optimization of retrieval effectiveness.


Thank you for your attention...
Los Angeles Times
BUSINESS
Jobless rate is expected
Fresh Scrutiny of BofA
INVESTOR'S BUSINESS DAILY
Judge Tosses Oracle Award
Banks Hard Sell
Money
Profit And Lea
Investors.com