Sparse Matrix Sparse Vector Multiplication - A Novel Approach

Monika Shah
Department of Computer Science & Engineering
Nirma University
monikag.shah@gmail.com

***Abstract*—The terabytes of information available on the internet creates a severe demand of scalable information retrieval systems. Sparse Matrix Vector Multiplication (SpMV) is a well-known kernel for such computing applications in science and engineering world. This raises need of designing an efficient SpMV. Researchers are putting their continuous effort to optimize SpMV that deal with wide class of sparse matrix patterns using various compressed storage formats, and algorithm for high performance computing devices like multi-core/many-core processor i.e. GPU. But, they have not focus on optimization of input vector, which is highly sparse for various applications.This paper presents a novel approach - Sparse Matrix Sparse Vector Multiplication (SpMSpV) to utilize sparse input vector efficiently. To demonstrate efficiency of the proposed algorithm, it has been applied to keyword based document search, where sparse matrix is used as index structure of text collection and sparse vector for query keywords. The proposed algorithm is also implemented over Graphical Processing Unit (GPU) to explore high parallelism. Implementation results over CPU and GPU both demonstrate that SpMSpV using Compressed Sparse Column (CSC) sparse format is more efficient for information retrieval applications that use highly sparse input vector.**

Keywords: Sparse Matrix, Sparse Vector, Information Retrieval, SpMSpV, Query Processing, SpMV

I. Introduction

In todays era, everyone is striving to run office paperless with help of computerized documents. Increasing digital awareness, population and complexity of computation direct researchers to work on most striking research topic - efficient Big data management. Bigdata is described as volumeneous and complex data set in both structured and unstructured format,which is highly sparse in nature. This motivates design and use of appropriate indexing, and search algorithms for efficient information retrieval (IR). Storage space and query processing time are two important factors to judge the performance of an information retrieval engine. For years, inverted index data structure is used as a central component in information retrieval systems. The support for fast query processing resulted in wide adaptation of this data structure. However, the complexity to update and in parallelization of inverted index for performance optimization poses a limitation on its usage. Goharian et al.[1] presents an alternative approach to inverted index by storing index of text collection in sparse matrix format and retrieve relvant information using sparse matrix-vector multiplication based query processing, where query keywords are stored in dense input vector. Query size is observed much small in compare to keyword collection used in the domain, and size of query vector is predefined to be equal to size of keyword collection for matrix-vector multiplication based query processing. For example, web search engine has query of 8-10 words generally, but keywords registered in search engine may be more than million. This makes query vector highly sparse.

In present days, Sparse Matrix Dense Vector Multiplication ($Ax = y$, where x and y are vector, and A is a sparse matrix) is a popular as core component of many other science, and engineering applications. Large class of SpMV applications have observed variety of sparse pattern for data matrix A. To deal with variety of sparse pattern and huge size data matrix, optimization of SpMV is still retained as research focus for academia and researchers. The key difference between past effort and this paper is that here focus is on sparse input vector x and previous effort was focusing in sparse matrix A.

This paper proposes sparse matrix-sparse vector(SpMSpV) multiplication as a novel approach to optimize performance of information retireval at next higher scale. It is proposed specially to overcome following important performance associated issues:

- Continous increase of keywords with time
- Input query vector is highly sparse in many IR applications
- Optimization attempt till now on SpMV have no emphasis on structure and sparse level of input vector
- Ever demanding requirement of reducing computational operation

Over last decade, we are witness of revolutionary change in computer architecture from CPU to multi-core to many-core device like GPU aiming performance optimization using data intensive parallel algorithms. The parallelization of SpMV plays a key role in improving performance of IR process. This has been demonstrated in articles [2], [3], [4], where they achieve substantial performance improvement over the sequential inverted index implementation. Hence, this novel approach SpMSpV is also implemented over GPU to perform query processing more efficiently.

1530-2016/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPPW.2015.18

Following factors plays an important role in improving the performance of the proposed implementation:

i) Super scalar parallelization using many-core device GPU
ii) Reduction in computation
iii) Reduction in storage space of query vector
iv) Agglomerative communication
v) Reduction in fetch operation to avoid drawback of low latency memory access in GPU

The rest of paper is structured as follows: The existing attempt to optimize SpMV on GPU have been discussed in section II. Section III-A brings forth attempt placed in this paper to optimize performance of Matrix Vector multiplication using a novel approach SpMSpV. Architecture of query processing using SpMSpV is presented in section III-B. Section IV exhibits the performance evaluation of proposed work and its speed-up comparison with well-known SpMV algorithms.

II. Related work

Many sparse formats and its associated SpMV algorithms are proposed as a result of past effort by academicians and researchers. Data compression is fundamental method to reduce overhead of transfer cost between CPU and GPU as well as to reduce transfer between disk memory and main memory. BELL et al. [5] have proposed many core sparse formats for sparse matrix claiming higher compression. Among this sparse storage formats, coordinate (COO), compressed sparse row (CSR) and compressed sparse column (CSC) are well-known highly compressed formats for sparse matrix. CSR reduces row index vector size from Number of Non-zero (NNZ) elements to (Number of Rows (NR) + 1), and CSC reduces column index vector size from NNZ to (Number of columns (NC) + 1) in compare to COO structure. There exists many sparse formats proposing compressed storage [6], [7], [8] for sparse matrix, load balanced workload distribution among threads of GPU devices[7], [9], [10], [11], reuse of input vector to reduce fetch operation using texture cache[9] or data reordering[7].

N. GOHARIAN et al.[1] have introduced an attractive approach for text based document search using SpMV kernel, which is adopted as fundamental architecture for information retrieval. To demonstrate SpMV based query processing, Table II represent database matrix of document collection given in Table I and Table III represent query vector in terms of term frequency. A comparative study of various SpMV kernels for text based query processing is described in the paper [12], [4], where CSR is observed highly efficient. Blocked structure of CSR (BCR) help to compress sparse matrix further. BCR is known highly compressed storage and also claim better query processing [12]. [3] describes various compression techniques on CSR based information retrieval. Statistics given for benchmark datasets in paper [13] shows that size of document collection or dataset (NR) is much smaller than size of keyword collection (NC). Hence, CSR compress better than CSC for such data sets. But, continuous increase in data set keeps increasing NR. Query processing using CSC does not perform well as processing cycles are wasted on zero values of input query vector. Some important observation of existing work related to SpMV optimization are as follow:

i) CSR provides better compression for matrix having small number of rows(NR), and CSC provides better compression for matrix having small number of columns(NC)
ii) Compression difference between CSR and CSC does not have more significance where NNZ much larger in compare to NR and NC.
iii) T(CSC) is $\lceil \frac{max\_col\_len}{max\_concurrent\_threads} \rceil$ x num_cols
iv) T(CSR) is $\lceil \frac{NR}{max\_concurrent\_threads} \rceil$ x max_row_len
v) Optimizations proposed on SpMV does not focus on sparse level and structure of input vector
vi) Each element from input vector is multiplied with only corresponding column of sparse matrix

These points drawn our attention to choose CSC sparse format for implementation of SpMSpV, where time complexity of CSC can be improved by reducing num_cols equals to NNZ of input vector.

III. Proposed Work

In this section, proposed SpMSpV kernel is described, which is suitable for highly sparse input vector. This section also include proposed framework of information retrieval for an application keyword based document search to demonstrate efficiency of SpMSpV.

TABLE I
KEYWORDS PRESENT IN DOCUMENT SET AND QUERY

Document ID	Keywords
D0	text text information retrieval
D1	information retrieval system
D2	text processing text system
D3	processing processing information
D4	text information text processing
D5	text search engine
Query	text processing

TABLE II
DATABASE MATRIX A (DOCUMENT X KEYWORD(TERM)) IN FORM OF TERM FREQUENCY *tf*

	information	text	processing	retrieval	system	search	engine
D0	1	2		1			
D1	1			1	1		
D2		2	1		1		
D3	1		2				
D4	1	2	1				
D5		1				1	1

TABLE III
QUERY VECTOR x IN FORM OF TERM FREQUENCY *tf*

infor-mation	text	proces-sing	retrieval	system	search	engine
	1	1				

A. SpMSpV kernel

CSC sparse format for sparse matrix and sparse format for input query vector are heart of SpMSpV kernel. Section II have described that query vector is highly sparse in query processing and similar applications. Query value Qv[], and Query Index Qi[] are two vectors that represent sparse format for sparse query vector as demonstrated in Fig.1. Qv lists non-zero values of the vector, and Qi lists index position for associated each value of Qv.

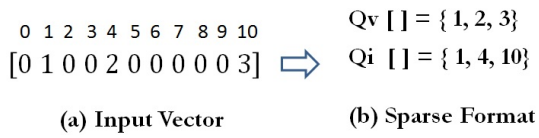

Fig. 1. Sparse format for input query vector

Fig.2 demonstrates basic methodology of SpMSpV algorithm for sparse matrix described in Table II with input query vector shown in Table VI. For each non zero element in queue vector, it is multiplied with non zero elements of respective column. For example, Non-zero element 0.0676 at position 2 is multipled with column 2 and another Non-zero element 0.36 at position 3 is multiplied with column 3 as scalar operations. Hence, T(CSC SpMSpV) is reduced to 2 in compare to T(CSC SpMV), which is 7 column multiplications. This result is also as accurate as dense matrix-vector multiplication.

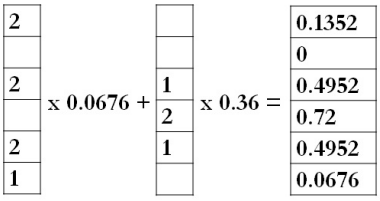

Fig. 2. SpMSpV computation

Factors listed in section I, which are responsible for higher performance on GPU device are considered here in design of SpMSpV algorithm. Detail description of these factors is presented here.

Reduction in Computation:
CSC, CSR, and COO SpMV algorithms are known for performing reduced arithmetic operations to compute $Ax = y$. For example, Table IV demonstrates SpMV for matrix given in Table II and vector given in Table VI. The key difference observed here is that SpMV performs $\sum NNZ[c]$ multiplication for all column c, while SpMSpV performs $\sum NNZ[i]$ multiplication(i correspond index position of non-zero element of input vector). In this example, any SpMV performs minimum 17 multiplication and 17 addition; while SpMSpV performs 7 multiplication and 7 addition. Thus, this algorithm reduces amount of computations at very high scale for multiplication of huge size matrix and highly sparse query.

TABLE IV
SPMV COMPUTATION FOR CSC,CSR,COO SPARSE FORMATS

Document ID	Result Vector
D0	= 1*0 + 2*0.0676 +1*0 = 0.1352
D1	= 1*0 + 1*0 + 1*0 = 0
D2	= 2*0.0676 + 1*0.36 + 1*0 = 0.4952
D3	= 1*0 + 2*0.36 = 0.72
D4	= 1*0 + 2*0.0676 + 1*0.36 = 0.4952
D5	= 1*0.0676 + 1*0 + 1*0 = 0.0676

Reduction in storage space of query vector
Data transfer between CPU and GPU is big overhead for GPU based application. Therefore, researchers prefer compressed storage formats for data to be transfer on GPU. Another observation is that query processing in information retrieval need to convert text query to vector format on host and transfer it on GPU device for every new query. Henceforth, this work also propose compression for input sparse query vector as shown in Fig.1. in addition to compression of sparse matrix

In this digital world, there exist many other query processing application like query processing for database, feature based object identification, content based image retrieval where content is described in form of features, auto-prediction of disease based on symptoms, bioscience research application to identify new biological sequence based on patterns, etc. All query processing applications have very small sized query i.e. search string for query is rarely 80,100,120 characters or 6-15 words long [13]. Thus, sparse format presented in Fig.1 reduce input vector size to 2x(15/87833)=0.03 percentage for query keyword vector size 87833 (As per statistics of database described in section IV-A). This demonstrate that sparse format for query vector reduce storage space requirement on device, which is important feature for embedded system.

TABLE V
DOCUMENT FREQUENCY (*df*) AND ENTROPY (*ep*) FOR EACH TERM

	infor-mation	text	proces-sing	retrieval	system	search	engine
df	4	4	3	2	2	1	1
ep	0.26	0.26	0.6	1.43	1.43	4.67	4.67

Agglomerative communication
Agglomeration is a process of bundling individual communication into a super communication. This will help in reducing communication cost by reducing iterations

TABLE VI
QUERY VECTOR IN FORM OF $tf * ep^2$

information	text	processing	retrieval	system	search	engine
0	0.0676	0.36	0	0	0	0

of communication. As per description above, input vector in our proposed algorithm is of much small size in compare to data bus bandwidth between CPU and GPU. The proposed SpMSpV on GPU applies this agglomoration by bundling Qi and Qv vectors into a single array and transferred in single iteration from host to device.

Reduction in fetch operation to avoid drawback of low latency memory access in GPU

The article [9] describes that reduction in fetch operation is essential to improve performance of implementation on GPU. It has been satisfied here using two important strategies: (i) Compression applied in sparse vector and sparse matrix, and (ii) use of texture cache to access element of queue value vector Qv[], and Queue index vector Qi[]. All non-zero elements of selected Qi[i] column is required to be multiplied with ith value of Qv vector. These non-zero elements of selected columns are assinged to individual threads and each thread use ith value from Qi and Qv vector. Hence, texture cache is suggested to to access Qi and Qv vector to reduce fetch operations.

SpMSpV algorithm for GPU is listed in Algorithm 1. All non-zero elements in query vector Qv is multiplied with corresponding column sequentially. For example described in Table VI and Fig.2, column 2 (Qi[0]) is multiplied with query keyword at index 2 (Qv[0]) followed by column 3 (Qi[1]) multiplication wtih query keyword at index 3 (Qv[1]). Non-zero elements of corresponding column Qi[i] are multiplied concurrently with Qv[i]. The synchronization free load distribution is important factor to achieve optimizing performance[9], which is achieved here as multiplication results of all elements of a column are independent of each other.

Algorithm 1 Sparse Matrix Sparse Vector Multiplication(SpMSpV)

Input : csc (CSC matrix),
Qiv (Index Vector and Value Vector of Input Vector),
Q_NNZ(number of non-zero values in input vector),
num_threads(parallel threads available)
num_blocks(parallel threads blocks)

Output: Y(Output Vector)

```
Qi ← Qiv
Qv ← Qiv + Q_NNZ
tid ← compute thread ID
for i = 0 → Q_NNZ do
    //Fetch Qi[i] using texture cache
    col ← fetch_I<UseCache>(i,Qi)
    //Fetch Qv[i] using texture cache
    x_val ← fetch_X<UseCache>(i,Qv)
    //Locate respective column
    col_start ← csc.Ap[col]
    col_end ← csc.Ap[col+1]
    for j = col_start + tid → col_end do
        row ← csc.Ai[j]
        y[row] ← y[row] + csc.Ax[j] * x_val
        j ← j + num_threads
    end for
    //Global barrier code to wait for all threads
    barrier_wait(num_blocks)
end for
return
```

B. Framework of Information Retrieval

To demonstrate efficiency of proposed SpMSpV algorithm, it has been applied to an application - keyword based document search. The proposed framework for keyword based document search engine is based on framework described in [1]. This framework proposes following approaches for optimization:

i) Index construction using Entropy based encoding
ii) Storage space compression for Query Vector
iii) Storage space compression using CSC for an index sparse matrix
iv) Reduce computation cost
v) Parallelization on GPU

Fig.3 shows proposed framework for Keyword based Document Search Engine. It comprises of main three modules: Document Parser, Indexer, and Query Processor. This application helps to search a relevant set of documents containing given text components from text documents like web documents, XML documents, and other normal text documents. These documents may contain tag structures along with information.

Parser:

The documents collections are processed using parser to extract keywords as shown in Table I. This can be done by eliminating unnecessary data like tags, stop words (i.e. a, an, the, is etc.), pronoun words (I, He, She etc.), punctuation symbols, white spaces, new line, tab from given input document. The removal of unnecessary words helps to reduce index size. Then, parser identifies unique keyword term, and its frequency *tf* in the document. The set of keyword, corresponding term frequency *tf*, and offset positions is passed to Indexer for updating document frequency *df* (count of document that contain the term), and entropy *ep* (computed as d/df * log(d/df)) for respective term.

Indexer:

This module constructs document directory and sparse index matrix using CSC. Document directory helps system to avoid

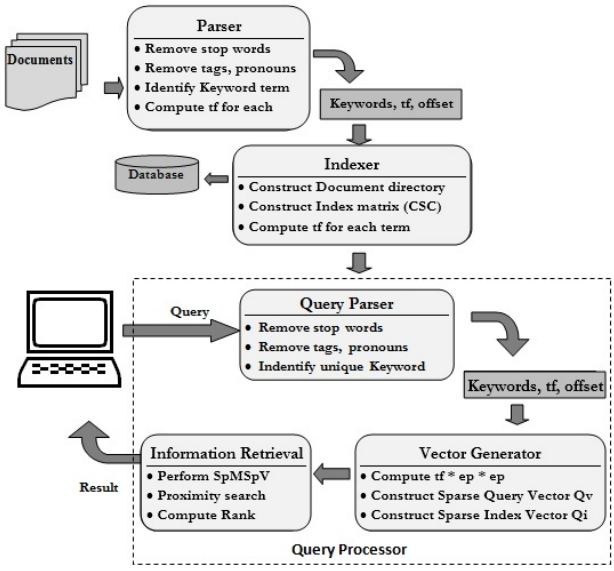

Fig. 3. Framework of Information Retrieval-Keyword based document search

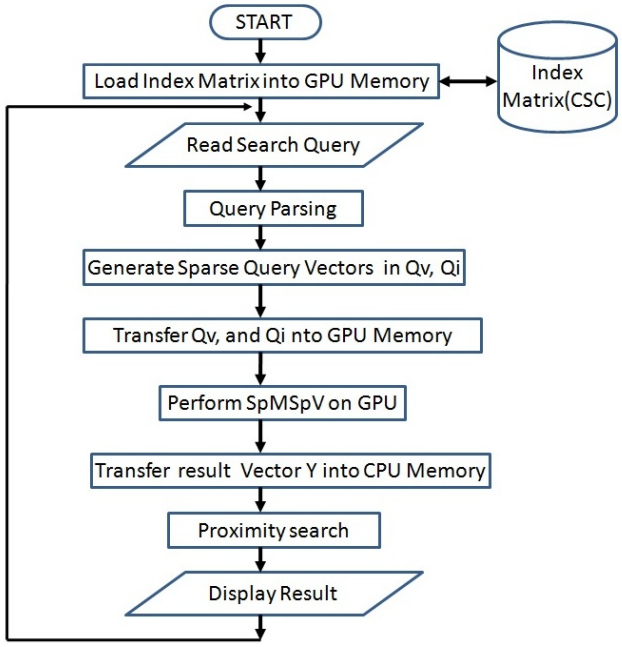

Fig. 4. Query processing

parsing for already parsed document. The index matrix for compressed storage will be stored using CSC as shown in Table II. CSC format comprises of three vector Ap, Ai, and Ax. Document Index of corresponding doucment is stored at Ai. To reduce storage space further, term frequency *tf* is stored at Ax instead of *tf * ep*.

Query Processor:

Query Processor is the heart of Information Retrieval. It uses heterogeneous environment consisting of CPU and GPU to perform query processing. Framework shown in Fig.3 and flow-chart shown in Fig.4 provides precise working of the Query processor. Recurrent use of SpMSpV for Query processing amortizes the overhead of one time communication cost of CSC database/index matrix from CPU to GPU. Query processor has three major components : Query Parser, Vector Generator, and Information Retrieval. Query parser identifies keywords from text query,and supplies term frequency, keywords, and offset corresponding to each term to Vector Generator. Query parsing, Vector Generation and Proximity search are executed on CPU. Vector Generator use previously computed entropy amd constructs vectors Qv and Qi for keywords in given query. Qv vector contain tf*entropy * entropy, where entropy is computed as d/df log(d/df). Information Retrieval component bundles Qv and Qi vector into single array and transfer it to GPU for parallelizing compute-intensive multiplication. SpMSpV is executed on GPU and its result is transferred to CPU.

IV. Experimental Result

For proper evaluation of this proposed information retrieval algorithm, keyword based document search have been successfully implemented on WebKB dataset and Industry dataset. The WebKB dataset [14] consist of sets of web pages from four computer science departments, with each page manually labeled into 7 categories: course, department, faculty, project, staff, student, or other. Industry dataset available from [15] covers business news stories from the web for specific duration. The WebKb dataset and Industry data set is used to prepare document collection. Main focus of this paper is on SpMSpV algorithm. To demonstrate efficiency of SpMSpV over SpMV algorithm, various SpMV algorithms like COO_flat, CSR, ELL, and HYB SpMV from NVIDIA cusp-library have been implemented on NVIDIA GPU. CPU version of SpMSpV and CSR have been also implemented to compare CPU - GPU performance.

A. Test Platform

These experiments have been executed on Intel(R) Core(TM) i3 CPU @ 3.20GHz with 4GB RAM, 2 x 256KB(L2 Cache) and 4 MB (L3 Cache), and NVIDIA C2070 GPU device using CUDA version 4.0 on Ubuntu 11.

Here, main focus is on sparse level of input vector. Hence, all tests have been applied over different sparse level of input query vector and same sparse matrix. This sparse matrix is generated from WebKb and Industry dataset with dimension 11582 NR x 87833 NC and 01298977 NNZ. Sparse level is computed here as $(Q\_NNZ/NC) * 100$. Lower sparse level represent highly sparse data. This paper has stated that SpMSpV performs well on highly sparse input vector. So, these tests have been applied to sparse level ranges from 0.001 to 1.

B. Result Analysis

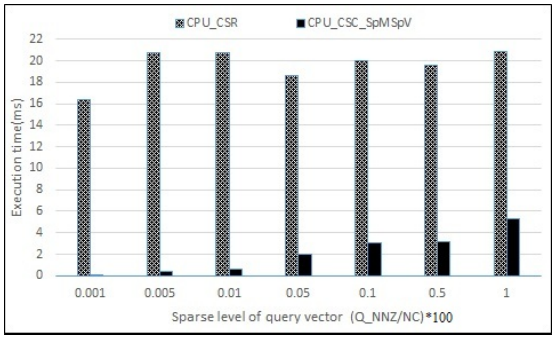

Fig. 5. Performance comparison between CSR_SpMV and CSC_SpMSpV on CPU

Based on the comparison given in [4], CSR SpMV gives better performance in comparison to other sparse matrix format. So, CSR is chosen to implement CPU based SpMV and its performance is compared with CPU based SpMSpV as shown in Fig.5.

Here, it can been observed that performance of SpMV remain consistent as index database matrix remain same for all execution and only query vector is changed. Performance of SpMSpV degrades with increased query vector size. SpMSpV performance is observed 4x-166x times better than CSR-SpMV with highly sparse input vector (small sparse level).

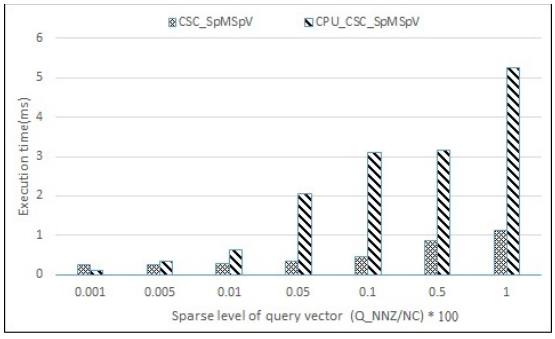

Fig. 6. Performance comparison of SpMSpV between CPU and GPU

GPU is well-known for large sized data intensive parallelism. On another side, SpMSpV also reduces computation at large scale. So, Performance of CPU based SpMSpV should be better than GPU for small number of operations. Performance comparison presented in Fig.6 satisfies this fact. Performance of GPU based SpMSpV implementation increases over CPU based implementation along with increasing count of keywords in query vector.

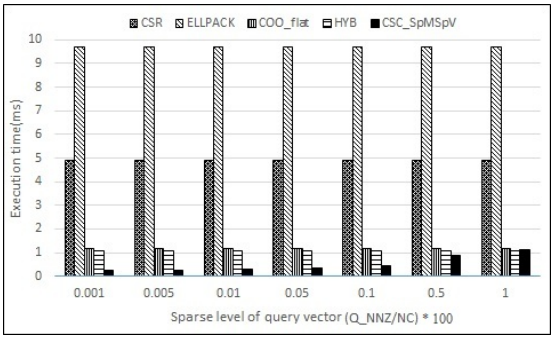

Fig. 7. Performance comparison between Various SpMV and SpMSpV on GPU

For proper evaluation of proposed SpMSpV algorithm, various benchmark sparse format and its associated SpMV algorithms were implemented on GPU and their execution time is compared as shown in Fig.7. Here, SpMSpV performance is found far better than CSR, and ELL algorithm. Inspite of COO_flat propose maximum concurrency with compromization in accuracy, SpMSpV still proposes 1x-4x speedup over COO_flat kernel on GPU with accuracy.

V. Conclusion and Future Work

In this paper, a novel but simple approach of matrix-vector multiplication is applied to gain much better performance for scenario when input vector is highly sparse. The proposed SpMSpV algorithm reduce computation at higher scale. Highly reduced computation suggest hybrid CPU-GPU approach to reduce memory transfer overhead between CPU and GPU. Performance bottleneck in this implementation is use of barrier between each two consecutive column computation to avoid synchronization conflict. But, it is negligible for small sized query and large set of document records. This approach can be more suitable for unstructured database used in BigData. In future, this approach may be applied on BigData based query processing.

References

[1] N. Goharian, D. Grossman, and T. El-Ghazawi, "Enterprise text processing: A sparse matrix approach," *Information Technology: Coding and Computing, International Conference on*, 2001.
[2] N. Goharian, A. Jain, and Q. Sun, "On the parallel implementation of sparse matrix information retrieval engine," 2002.
[3] S. S. Stein and N. Goharian, "On the mapping of index compression techniques on csr information retrieval," in *Proceedings of the International Conference on Information Technology: Computers and Communications*, IEEE Computer Society, 2003.
[4] N. Goharian, A. Jain, and Q. Sun, "Comparative analysis of sparse matrix algorithms for information retrieval," 2003.
[5] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *SC*, 2009.

[6] M. M. Baskaran and R. Bordawekar, "Optimizing sparse matrix-vector multiplication on gpus," *Engineering*, vol. 24704, no. RC24704, 2008.
[7] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on gpus: Implications for graph mining," *CoRR*, vol. abs/1103.2405, 2011.
[8] A. Pinar and M. T. Heath, "Improving performance of sparse matrix-vector multiplication," in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '99, 1999.
[9] M. Shah and V. Patel, "An efficient sparse matrix multiplication for skewed matrix on gpu," in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pp. 1301–1306, June 2012.
[10] A. Dziekonski, A. Lamecki, and M. Mrozowski, "A memory efficient and fast sparse matrix vector product on a gpu," *Progress In Electromagnetics Research*, vol. 116, pp. 49–63, 2011.
[11] W. Cao, L. Yao, Z. Li, Y. Wang, and Z. Wang, "Implementing sparse matrix-vector multiplication using cuda based on a hybrid sparse matrix format," in *International Conference on Computer Application and System Modeling*, 2010.
[12] B. Bani-Ismail and G. Kanaan, "Comparing different sparse matrix storage structures as index structure for arabic text collection," *Int. J. Inf. Retr. Res.*, vol. 2, 2012.
[13] N. Goharian, T. El-ghazawi, and D. Grossman, "On the enhancements of a sparse matrix information retrieval approach," in *Proceedings of the International Conference on Parallel and distributed Processing Techniques and Applications*, 2000.
[14] "http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/."
[15] "http://people.csail.mit.edu/jrennie/20newsgroups/."