k-NN Text Classification using an FPGA-Based Sparse Matrix Vector Multiplication Accelerator

Kevin R. Townsend, Song Sun, Tyler Johnson, Osama G. Attia, Phillip H. Jones, Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA, USA
{ktown, sunsong, tyler07, ogamal, phjones, zambreno}@iastate.edu

***Abstract*—Text classification is an important enabling technology for a wide range of applications such as Internet search, email filtering, network intrusion detection, and data mining electronic documents in general. The k Nearest Neighbors (k-NN) text classification algorithm is among the most accurate classification approaches, but is also among the most computationally expensive. In this paper, we propose accelerating k-NN using a novel reconfigurable hardware based architecture. More specifically, we accelerate a k-NN application's core with an FPGA-based sparse matrix vector multiplication coprocessor. On average our implementation shows a speed up factor of 15 over a naïve single threaded CPU implementation of k-NN text classification for our datasets, and a speed up factor of 1.5 over a 32-threaded parallelized CPU implementation.**

***Index Terms*—Text Classification, Reconfigurable Computing, k Nearest Neighbor, FPGA**

I. Introduction

As electronic medium continues to grow, so does the need for efficient mechanisms to organize this vast amount of information. For example, web browsers have billions [1], [2] of sites to search for a given query, and servers filter over 100 billion emails per day for spam world wide [3]. Machine learning techniques have been developed to enable such tasks. Some of these techniques focus on classifying electronic documents based on text [4], some based on images [5], [6], and hybrid techniques that use both. In the area of text classification, the primary approaches used are: Naïve Bayes, Rule Based, Decision Tree, k Nearest Neighbors (k-NN), Support-vector Machines (SVM), and Neural Network [4].

In this paper, we focus on accelerating k-NN text classification. k-NN has been shown to give good classification accuracy in many cases [7], [8], [9], however it is also known to be computationally expensive [10]. As discussed in Section III, multiplication of large sparse matrices with vectors is a substantial part of the k-NN computation. We propose leveraging our reconfigurable coprocessor architecture [11], called R^3, to accelerate this aspect of k-NN based text classification.

Our approach is evaluated against a naïve single CPU core k-NN implementation and against a 32 hardware-threaded CPU implementation. On average our implementation shows a speed up factor of 15 over the naïve single threaded CPU implementation, and a speed up factor of 1.5 over the 32-threaded parallelized CPU implementation, in terms of the number of nonzero elements processed in the sparse matrix per second (term-document elements per second). Our FPGA-based approach processes 2.7 billion term document elements per second, while the multiple CPU thread approach processes 1.8 billion term-document elements per second.

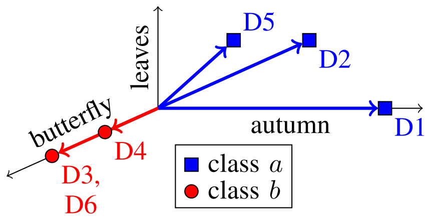

Fig. 1: Term-document vectors in a 3-D space where each dimension represents a word.

The remainder of this paper is organized as follows. Section II discusses related work. Section III then provides a brief overview of k-NN based text classification. In Section IV, implementation details of our FPGA-based design are provided. Publicly available workloads are utilized in Section V as benchmarks to evaluate our FPGA-based implementation, and compare against a single-threaded and multi-threaded implementation. Section VI summarizes our findings and points toward future research directions.

II. Related work

The general research area of text classification is fairly mature, and [4] presents a nice survey. In this section, we focus on providing an overview of related work from the areas of: k-NN, sparse matrix vector multiplication (SpMV), and the use of FPGAs for the accelerating machine learning algorithms.

While to our knowledge no prior work has specifically focused on accelerating k-NN text classification, there has been work that focuses on accelerating k-NN using FPGAs [12] and GPUs [13] for other applications. Unlike the use of k-NN for text classification, this other work typically has different characteristics in terms of scale of dimensionality, and the sparseness of matrices used for k-NN computation. Work also exists in the area of designing fast algorithms with classification accuracy similar to k-NN. These algorithms include support vector machines [14], [15] and dimension reduction [16].

TABLE I: Example documents for classification

	name	class	text
Training	D1	a	Autumn was it when we first met Autumn is it what I can't forget Autumn have made me alive
	D2	a	Grinning pumpkins, falling leaves, Dancing scarecrows, twirling breeze, Color, color everywhere, Autumn dreams are in the air! Autumn is a woman growing old
	D3	b	butterfly, butterfly fly in the sky butterfly, butterfly flies so high
	D4	b	Hoping to catch your eye Circling around you, oh my Butterfly, butterfly, come into the light Oh, what a beautiful sight
Testing	D5	a	Its autumn again Leaves whisper the sound of our past In loss they pay a descent To the ground we fall
	D6	b	Butterfly; butterfly fly away, teach me how to be as free as free can be. Butterfly; butterfly I see you there

		class	name	autumn	met	alive	leaves	color	growing	butterfly	fly	sky	flies	high	hoping	sight	whisper	fall	teach	free
A	training	a	D1	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		a	D2	2	0	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0
		b	D3	0	0	0	0	0	0	4	1	1	1	1	0	0	0	0	0	0
		b	D4	0	0	0	0	0	0	2	0	0	0	0	1	1	0	0	0	0
B	testing	a	D5	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0
		b	D6	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	1	2

Fig. 2: Training matrix A and testing matrix B for the documents in Table I. For simplicity, we removed some words that occur once, and removed commonly occurring words.

		testing	
		D5	D6
training	D1	3	0
	D2	3	0
	D3	0	17
	D4	0	8

Fig. 3: The AB^T matrix. This represents the relative distance values (using dot products) of each testing document from each training document.

In the area of SpMV acceleration, there is work that uses Multicore processor[17], GPU [18] and FPGA [11], [19] based approaches. For each of these platforms a different approach is taken to achieve acceleration: CPU implementations typically attempt to keep all the data associated with a vector in cache, GPUs tend to focus on efficient memory bandwidth usage, and FPGA implementations focus on compressing matrix data to reduce memory bandwidth requirements.

In the broader field of machine learning and data mining, there are many areas for which FPGAs have been used for acceleration. For example, neural nets [20], Apriori [21], support vector machines [22], and other k-NN type algorithms [23]. Recently, FPGAs have found their way into data centers, showing interest in FPGA acceleration by corporations. In [24], researchers from Microsoft used an FPGA-based design to accelerate their "Bing" web-search engine.

III. k-NN Text Classification Basics

This section walks through an illustrative example of the k-NN algorithm being used to classify a small set of documents, provided in Table I. The key idea of k-NN is to use training documents (D1-D4) to guide the algorithm in classifying new documents (D5-D6) into predefined classes (class a or b in this case). Only the k "closest" training documents to the new document are used for classifying the new document.

k-NN consists of 3 steps: 1) converting documents into term-document vectors, 2) finding the k closest training documents to the test documents (i.e. new documents), and 3) classifying the test documents.

Term-document vectors. A vector of word frequencies (a term-document vector) abstractly represents a document's location in term-document space. Figure 1 depicts the location of documents D1-D6 if each vector was composed of only 3 terms (words). Concatenating these row-vectors creates term-document matrices, shown in Fig. 2 for a 17 dimensional term-document space. Matrix A in this figure will be referred to as the training term-document matrix, and Matrix B as the testing term-document matrix.

Closest training documents. A notion of distance must be defined in order to compute how far apart documents are. In text classification, the dot product of term-document vectors works well as the distance metric [25]. As an example, the dot product (i.e. distance) between D1 and D5 equals 3. It should be noted that unlike Euclidean distance, larger dot products indicate that vectors (i.e. documents) are closer. Finding the distance between all training document / testing document pairs equates to matrix-matrix multiplication. Fig. 3 shows the resulting matrix (AB^T) containing the distance between all such pairs. With respect to the individual computations performed, AB^T reduces to a series of dot products.

Classifying. Only the k closest training documents are used to classify testing documents. For this example we let $k = 2$, in practice k is experimentally chosen. Fig. 4 shows the training documents sorted by their distance for each testing document, as well as the $k = 2$ cut-off depicted as a dotted line.

Once the $k = 2$ closest training documents for each testing document (D5 and D6) have been determined, then

	D5	D6
k	D1,a,3	D3,b,17
	D2,a,3	D4,b,8
	D3,b,0	D1,a,0
	D4,b,0	D2,a,0
sum	**a=6**,b=0	**b=25**,a=0

Fig. 4: In this figure each element is a tuple with the values: (document, document's class, distance from testing document). The sums are segregated by class and only the tuples above the k cutoff line get included in the sums.

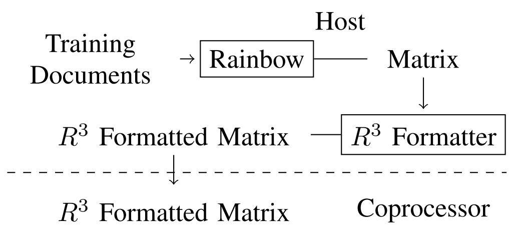

Fig. 5: The dataflow of the training phase, when using the FPGA coprocessor.

for each testing document the dot product is summed for each document class. In this case, for D5 the 2 closest training documents are both type a, and for D6 they both happen to be type b. D5 has a "score" of 6 for class a and 0 for class b, and D6 a score of 0 for class a and 25 for class b, thus D5 is classified as type a, and D6 is classified as type b. In other words, the class for which a testing document scores the highest is its classification.

IV. Implementation

In this section, we describe the implementation details of our design for each of the three k-NN Text Classification steps discussed in Section III: 1) creating term-document matrices, 2) computing document distances, and 3) classifying test documents.

Fig. 5, 6, and Algorithm 1 provide a high-level overview of these steps, and depict if computation and data reside on our system's CPU-based Host or FPGA-based coproccesor (partition indicated by a dotted line). Fig. 5 summarizes the creation of the training term-document matrix, this is often referred to as the training phase of k-NN. Fig. 6 and Algorithm 1 summarize what is typically called the testing phase. In this phase, the testing term-document matrix is generated, distance between training and testing documents are computed, and testing documents are classified.

A. Creating Term-Document Matrices

The training term-document matrix A and testing term-document matrix B are generated on the system's host CPU using an open source software package called Libbow [25]. More specifically, an executable within this package called Rainbow generates our matrix A and B. These matrices are transferred to the coprocessor for the document distance computation step of k-NN. In addition to generating term-document matrices, Rainbow can also perform the full k-NN Text Classification algorithm. We use the classification results generated by Rainbow as a correctness validation of our implementation, and provide Rainbow's run time (Fig. 9) as a reference.

B. Computing Document Distances

To compute the columns of the AB^T matrix, where each column represents the distance between a given testing document and all training documents, [26] suggests expanding one row of B at a time and running SpMV by starting with

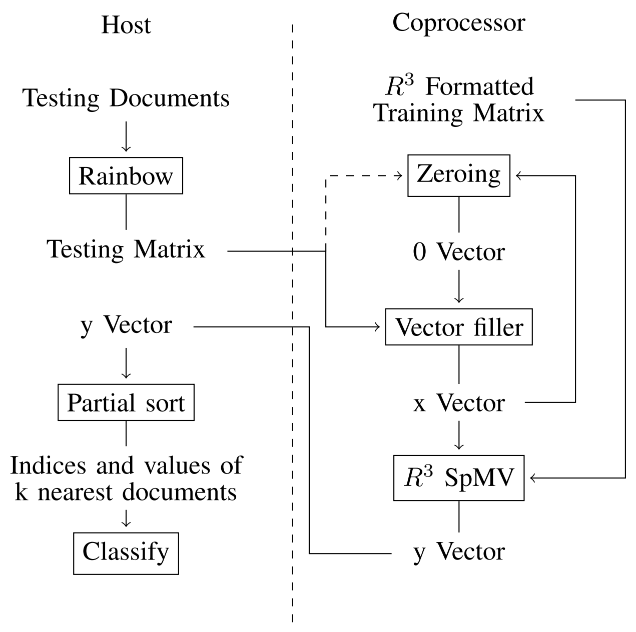

Fig. 6: The testing phase dataflow, when using the FPGA coprocessor.

Algorithm 1 Testing Phase

```
Use Rainbow [25] to convert documents to test matrix $B$
$x \leftarrow \{0,0,0,\dots,0\}$ {Initialize and zero vector $x$ with length equal to the height of $B^t$ on coprocessor}
for $i$ := 0 to $width(B^T)$ do
  Let the $i^{th}$ sparse column of $B^T$ equal $C$.
  for $j$ := 0 to $size(C)$ do
    $x[index(C[j])] \leftarrow val(C[j])$ {fill in $x$ with $C$, the column in the test matrix $B$}
  end for
  $y \leftarrow A \cdot x$ {run sparse matrix vector multiplication}
  for $j$ := 0 to $size(C)$ do
    $x[index(C[j])] \leftarrow 0$ {set $x$ back to 0 using $C$}
  end for
  Move $y$ from coprocessor to host.
  $bestMatches[0 \rightarrow k] \leftarrow \{0,0,0,\dots,0\}$
  for $j$ := 0 to $height(A)$ do
    for $l$ := 0 to $k$ do
      if $y[j] > bestMatches[l]$ then
        $bestMatches[l] \leftarrow y[j]$
      else
        break
      end if
    end for
  end for
  Decide classification of document based on $bestMatches$.
end for
```

a zero vector and then filling in the nonzero values with the sparse vector data. Before moving onto the next vector, the expanded vector needs to be reset to zero. Given that we are assuming that most vectors are sparse, significant time can be saved by just zeroing out the nonzero entries of the previous sparse vector, as opposed to always generating a brand new zero vector.

Referring back to the example in Section III to help explain this process, the term-document vector of document five, {1,0,0,1,0,...,0,1,1,0,0}, has a sparse format of: Indices={1,4,14,15} and Values={1,1,1,1}. Starting with a zero vector, {0,0,0,0,0,...,0,0,0,0,0}, the vector expansion only requires 4 writes: the first, value 1 written to index 1, {1,0,0,0,0,...,0,0,0,0,0}, the second, value 1 written to index 4, {1,0,0,1,0,...,0,0,0,0,0}, the third, value 1 to index 14, {1,0,0,1,0,...,0,1,0,0,0}, and lastly the fourth, value 1 to index 15, {1,0,0,1,0,...,0,1,1,0,0}. The indices also help reset the vector faster: the first 0 written to index 1, {0,0,0,1,0,...,0,1,1,0,0}, the second, to index 4, {0,0,0,0,0,...,0,1,1,0,0}, the third, to index 14, {0,0,0,0,0,...,0,0,1,0,0}, and lastly the fourth, to index 15, {0,0,0,0,0,...,0,0,0,0,0}.

1) Parallel CPU Platform: We used 2, 8-core, 16-thread, Intel E5-2650 CPUs for our software implementation (a total of 32 theads). To run SpMV on the CPU, we wrote a C function, and compiled it with gcc with speed optimization (*gcc -O3*). This achieves performance comparable to Intel's published performance [27]. To parallelize the software, we split the SpMV function into 32 threads and gave each a 32nd of the matrix split horizontally.

2) FPGA Platform: We used a Convey HC-2 [28] as our FPGA platform, with R^3 as our SpMV implementation. This platform has 4 Xilinx Virtex-5 LX330 [29] FPGAs. At the high level (see Fig. 7), R^3 places as many processing elements as possible on the FPGAs. R^3 occupies 75%, 72%, 75%, and 86% of the registers, LUTs, DSP blocks (multipliers), and BlockRAMs on the FPGA respectively, and runs at 150Mhz. Similar to the CPU implementation, R^3 needs to split the matrix into 64 horizontal pieces to parallelize the SpMV task, one for each processing element.

At the low level (see Fig. 8), a R^3 processing element has three main features which enable high performance. First, R^3 reduces the size of the matrix through index and value compression. This increases the rate information flows from RAM to the FPGA. The compression overhead amortizes across the test documents.

Second, R^3 has a multiply-accumulator that handles 32 rows at a time. This ensures the accumulator does not stall when the accumulator finishes the current row and moves to the next row.

Third, the ability to process multiple rows at a time means the elements in those rows of the matrix can be traversed in any order. R^3 uses a hybrid column and row traversal. This traversal, called Global Row Major Local Column Major traversal, reduces the number of vector value accesses.

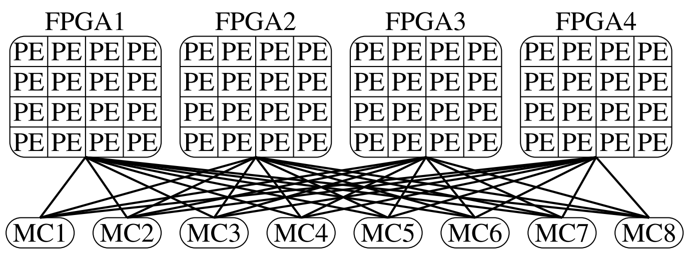

Fig. 7: R^3 implementation on the Convey HC-2 coprocessor: 4 Virtex-5 LX330 FPGAs tiled with 16 R^3 SpMV processing elements (PE) each. Each Virtex-5 chip connects to all 8 memory controllers (MC), which enables each chip to have access to all of the coprocessor's memory.

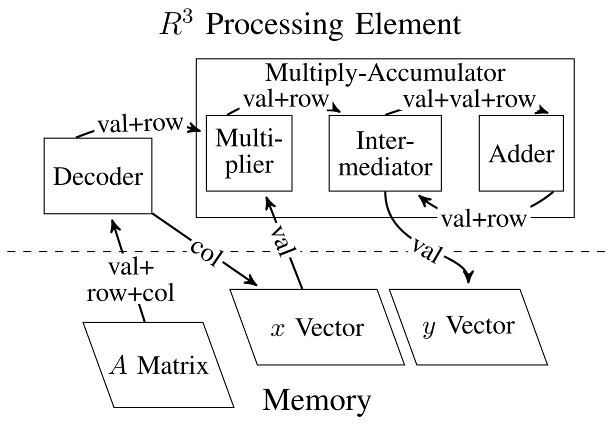

Fig. 8: A single R^3 processing element. The arrows show the flow of data through the processing element.

C. Document Classification

The last step of k-NN involves finding the k ($k = 30$ in our case) largest values that have just been calculated (recall for the dot-product a larger value corresponds to a smaller distance). Sorting the entire set of distances does find the largest distances, but requires a non-trivial amount of time. However, only partially sorting the list (i.e. keeping track of the k largest values) requires a much smaller computation [30]. We implement partial insertion sort to accomplish this.

To explain partial insertion sort and its benefits, we refer back to the example in Section III, specifically the distances from document five: {3,3,0,0}, which we want to extract the 2 largest values. Our partial insertion sort starts with a zero array: {0,0}. When inserting the first value 3, the array changes from {0,0} to {3,0} to {0,3}. When inserting the second value, also 3, the array changes from {0,3} to {3,3}. The final two values, both 0, do not get pass the first value in the array and get discarded.

This algorithm's efficiency comes from the fact that a high percent of the values get discarded after comparing to the first value in the sorted array.

TABLE II: Matrix statistics

	Height[1]	Width[2]	nnz[3]	nnz/row[4]
Twenty Newsgroups	train:11,314 test:7,531	114,545	train:1,082,852 test:705,576	94.9
NSF Abstracts	374,989	261,976	30,182,110	80.5
Reuters News	804,414	276,167	61,439,527	76.4

[1] Documents in the dataset.
[2] Unique words in the dataset.
[3] Number of nonzeros or term-document elements.
[4] Average unique words per document.

V. Results

For benchmarking, we used 3 datasets: *20 newsgroups* [31], *NSF abstracts* [32], and *Reuters News Corpus* [33]. *20 newsgroups* consists of 18,845 post on 20 Usenet newsgroups. *NSF abstracts* consists of abstracts of NSF grants from 1976 to 2014. *Reuters News Corpus* consists of news stories from 1996 to 1997.

Fig. 10 shows the matrix density plots after Rainbow processes them. The dimensions, number of nonzero elements and sparsity of the matrices in Table II provide insights into performance differences of different datasets.

The quality of results (correct classifications) equals Rainbow's implementation because our program performs the same operation as Rainbow. (Each implementation's results match Rainbow's results exactly.) For simplicity, the NSF and Reuters datasets only have one test document each. Only the runtime of k-NN is being tested.

For simplicity we choose not to use any vector normalizations to achieve better quality. Normalizations do not significantly change the number of floating point operations in the k-NN calculation so the performance should remain about the same with vector normalizations. For reference, the Rainbow options we used were: "*–method=knn –knn-k=30 –knn-weighting=nnn.nnn*". With these settings Rainbow achieves 31% accuracy.

We measure the performance of 4 different implementations: Rainbow, single-threaded CPU, multi-threaded (32 threads) CPU, and FPGA. The runtimes (Fig. 9) indicate that the FPGA implementation outperforms competing implementations.

The term-document elements processed per second metric helps to compare the 4 implementations. Table III contains each performance number for each implementation. The parallel CPU averages 1.80 billion term-document elements per second, whereas the FPGA averages 2.72 billion term-document elements per second, a speed up factor of 1.5 for the dataset.

Since SpMV takes a large percent of the run time, the SpMV performance (Table IV) provides a more direct understanding for the different performance of different implementations. Rainbow and the naïve implementation perform worse for large matrices. The increased matrix size causes worse cache performance. In the parallel case, the overhead of creating 32 threads means larger matrices perform better

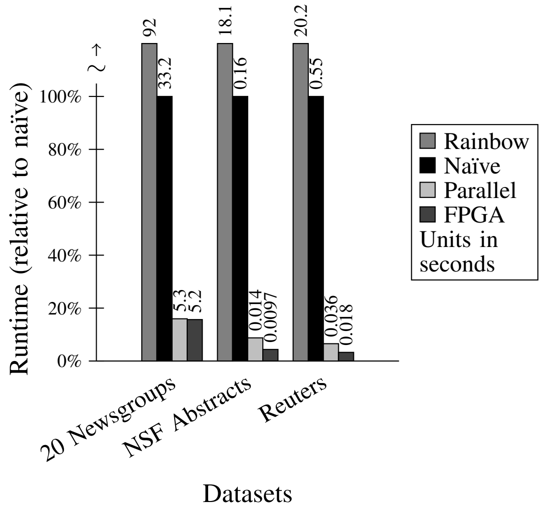

Fig. 9: The runtimes of the different implementations shows our FPGA implementation outperforms the others. It also performs better for large datasets relative to the CPU implementations.

TABLE III: Performance of each implementation, in terms of billion term-document elements processed per second.

	20 Newsgroups	NSF Abstracts	Reuters News	Average
Rainbow[25] [1]	0.089	0.002	0.003	0.031
Naïve (CPU)	0.25	0.19	0.11	0.18
Parallel (CPU)	1.54	2.16	1.71	1.80
FPGA (R^3)	**1.57**	**3.11**	**3.49**	**2.72**

[1] Included as reference.

to a point. This explains the increase in performance from the *20 newsgroups* dataset to the *NSF abstracts* dataset. The decrease in performance from the NSF dataset to the Reuters dataset can be explained using the same reason the naïve and Rainbow implementation get worse performance, worse cache performance.

R^3 performs slightly better on the NSF and Reuters datasets than the 20 newsgroups dataset. Some of this can be explained by R^3's overhead, which equals the time to process 100,000 matrix elements. However, the difference is larger than the expected .5 GFLOPS difference. It is likely the values in the NSF and Reuters matrices compress better.

VI. Conclusion

This paper has demonstrated that FPGAs are a viable means to accelerate k-NN text classification. Profiling a k-NN application revealed that its run time was dominated by sparse matrix vector multiplication (SpMV). Offloading these SpMV computations to an FPGA-based coprocessor resulted in an average speed up factor of 15 over a naïve single

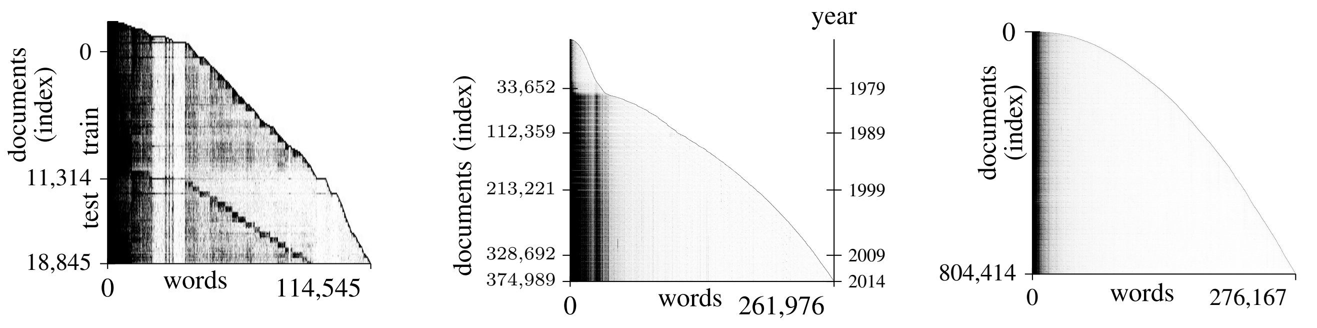

Fig. 10: The density plots of the three matrices derived from the datasets used to test k-NN implementations (20 newsgroup dataset [31], NSF abstracts dataset [32], and Reuters News Corpus (RCV1) [33]). The shading indicates the sparsity (0.1% ▨ 0%) in that location of the matrix.

TABLE IV: SpMV performance of each implementation in terms of GFLOPS (billion floating-point operations per second).

	20 Newsgroups	NSF Abstracts	Reuters News	Average
Naïve (CPU)	0.54	0.38	0.23	0.38
Parallel (CPU)	3.30	4.61	3.66	3.86
R^3 (FPGA)	**5.07**	**8.05**	**9.45**	**7.52**

threaded CPU, and a speed up factor of 1.5 over a 32-threaded parallelized CPU implementation.

Even when SpMV computations were offloaded to our co-processor, SpMV still dominated the k-NN text classification application run time (Figure 11). An interesting direction for future work would be attempting to further speed up these computations by treating them as sparse matrix-sparse matrix multiplication. As a second avenue for future research, since Libbow is such a widely used software application, integrating this FPGA-coprocessor approach into the Libbow framework would make it more attractive to the wider text classification research community.

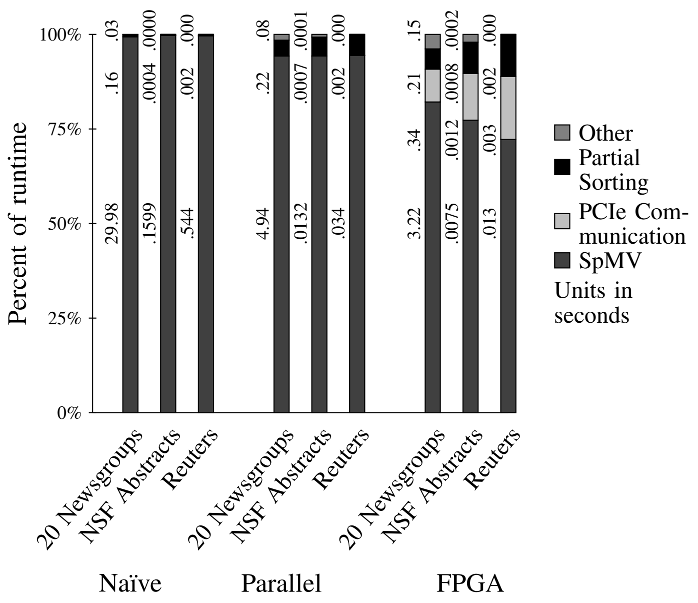

Fig. 11: Profiling of k-NN using the Naïve (one CPU thread), parallel (32 CPU threads), and FPGA (R^3) implementations.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under the awards CNS-1116810 and CCF-1149539.

References

[1] (2014) Internet live stats. [Online]. Available: http://www.internetlivestats.com/

[2] (2014) The size of the world wide web (the internet). [Online]. Available: http://worldwidewebsize.com/

[3] S. Bajaj and J. Pieprzyk, “Can we CAN the email spam,” in *Proceedings of the Cybercrime and Trustworthy Computing Workshop (CTC)*, Nov. 2013, pp. 36–43.

[4] C. C. Aggarwal and C. Zhai, *Mining Text Data*. New York, NY: Springer, Feb. 2012, ch. 6.

[5] N. Chen and D. Blostein, “A survey of document image classification: Problem statement, classifier architecture and performance evaluation,” *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 10, no. 1, pp. 1–16, May 2007.

[6] Y. Gao, A. Choudhary, and G. Hua, “A comprehensive approach to image spam detection: From server to client solution,” *IEEE Transactions on Information Forensics and Security (T-IFS)*, vol. 5, no. 4, pp. 826–836, Dec. 2010.

[7] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. New York, NY: Cambridge University Press, Jul. 2008.

[8] X. He, C. Zhu, and T. Zhao, “Research on short text classification for web forum,” in *Proceedings of the IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 2, Jul. 2011, pp. 1052–1056.

[9] N. Li and D. D. Wu, “Using text mining and sentiment analysis for online forums hotspot detection and forecast,” *Decision Support Systems*, vol. 48, no. 2, pp. 354–368, Jan. 2010.

[10] F. Sebastiani, “Machine learning in automated text categorization,” *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[11] K. Townsend and J. Zambreno, “Reduce, reuse, recycle (R^3): a design methodology for sparse matrix multiplication on reconfigurable platforms,” in *Proceedings of the IEEE International Conference on*

Application-Specific Systems, Architectures and Processors (ASAP), Jun. 2013.

[12] E. Manolakos and I. Stamoulias, "Flexible IP cores for the k-NN classification problem and their FPGA implementation," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium and PhD Forum (IPDPSPW)*, Apr. 2010, pp. 1–4.

[13] S. Liang, Y. Liu, C. Wang, and L. Jian, "A CUDA-based parallel implementation of K-nearest neighbor algorithm," in *Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems (CYBER)*, Oct. 2009, pp. 291–296.

[14] M. Papadonikolakis, C.-S. Bouganis, and G. Constantinides, "Performance comparison of GPU and FPGA architectures for the SVM training problem," in *Proceedings of the International Conference on Field-Programable Technology (FPT)*, 2009, pp. 388–391.

[15] T. Li, "On kNN and SVM text classification technology in knowledge management," in *Proceedings of the IEEE International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT)*, vol. 8, Aug. 2011, pp. 3923–3926.

[16] M. Davy and S. Luz, "Dimensionality reduction for active learning with nearest neighbour classifier in text categorisation problems," in *Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2007, pp. 292–297.

[17] E. Im, "Optimizing the performance of sparse matrix-vector multiplication," Ph.D. dissertation, University of California, Berkeley, May 2000.

[18] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," Nvidia, Technical Report NVR-2008-004, Dec. 2008.

[19] M. Gerards, "Streaming reduction circuit for sparse matrix vector multiplication in FPGAs," Master's thesis, University of Twente, Aug. 2008.

[20] T. Higuchi, M. Iwata, I. Kajitani, H. Yamada, B. Manderick, Y. Hirao, M. Murakawa, S. Yoshizawa, and T. Furuya, "Evolvable hardware with genetic learning," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, 1996, pp. 29–32.

[21] Z. Baker and V. Prasanna, "Efficient hardware data mining with the apriori algorithm on FPGAs," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2005, pp. 3–12.

[22] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A hardware efficient support vector machine architecture for FPGA," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2008, pp. 304–305.

[23] Y. Yeh, H. Li, W. Hwang, and C. Fang, "FPGA implementation of kNN classifier based on wavelet transform and partial distance search," in *Processings of the Scandinavian Conference on Image Analysis (SCIA)*, May 2007, pp. 512–521.

[24] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. young Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the IEEE International Symposium on Computer Architecture (ISCA)*, Jun. 2014, pp. 13–24.

[25] A. K. McCallum. (1996) Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. [Online]. Available: http://www.cs.cmu.edu/~mccallum/bow/

[26] S. Sun, "Analysis and acceleration of data mining algorithms on high performance reconfigurable computing platforms," Ph.D. dissertation, Iowa State University, Ames, Jan. 2011.

[27] *Intel Math Kernel Library Reference Manual*, 11th ed., Intel, 2012. [Online]. Available: http://software.intel.com/

[28] *Convey Reference Manual*, 1st ed., Convey, Richardson, TX, May 2012.

[29] *Virtex-5 Family Overview*, 5th ed., DS100, Xilinx, Feb. 2009.

[30] V. Gardia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2008, pp. 1–6.

[31] K. Lang, "Newsweeder: Learning to filter netnews," in *Proceedings of the International Conference on Machine Learning (ICML)*, Jul. 1995, pp. 331–339.

[32] (2014, May) NSF award search: download awards by year. [Online]. Available: http://www.nsf.gov/awardsearch/download.jsp

[33] D. D. Lewis, Y. Yang, T. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," in *Journal of Machine Learning Research (JMLR)*, vol. 5, Apr. 2004, pp. 361–397.