LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-CONF-234633

Language Classification using N-grams Accelerated by FPGA-based Bloom Filters

Arpith Jacob, Maya Gokhale

September 14, 2007

First International Workshop on High-Performance Reconfigurable Computing Technology and Applications
Reno, NV, United States
November 11, 2007 through November 11, 2007

Disclaimer

Language Classification using N-grams Accelerated by FPGA-based Bloom Filters

Arpith Jacob*
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, Missouri 63130–4899
jarpith@cse.wustl.edu

Maya Gokhale
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California 94551–0808
maya@llnl.gov

ABSTRACT

N-Gram (n-character sequences in text documents) counting is a well-established technique used in classifying the language of text in a document. In this paper, n-gram processing is accelerated through the use of reconfigurable hardware on the XtremeData XD1000 system. Our design employs parallelism at multiple levels, with parallel Bloom Filters accessing on-chip RAM, parallel language classifiers, and parallel document processing. In contrast to another hardware implementation (HAIL algorithm) that uses off-chip SRAM for lookup, our highly scalable implementation uses only on-chip memory blocks. Our implementation of end-to-end language classification runs at 85× comparable software and 1.45× the competing hardware design.

1. INTRODUCTION

An important problem in statistical natural language processing is the classification of documents according to language. Language classification is an important task for today's World Wide Web where an increasing number of documents are in a language other than English. Language classification finds use in search engine indexing, heuristics for spam filtering [5], information retrieval, text mining and other applications that apply language-specific algorithms. Such classification is a key step in the processing of large document streams, and is a data-intensive task.

*This work was performed when the author was a student intern at the Lawrence Livermore National Laboratory.

HPRCTA'07, November 11, 2007, Reno, Nevada, USA.

While there are a number of solutions that run on general purpose processors, the growth of document sets (for example on the World Wide Web) has far surpassed microprocessor improvements afforded by Moore's Law. Field Programmable Gate Array (FPGA) based systems offer an alternative platform that enable the design of highly parallel architectures to exploit data parallelism available in specific algorithms.

A well-known technique to classify the language of a text stream is to create an n-gram profile for the language [7]. An *n-gram* is defined as a sequence of characters of length exactly n. N-grams are extracted from a string, or a document, by a sliding window that shifts one character at a time. An *n-gram profile* of a set of documents is the t most frequently occurring n-grams in the set. The probability that an input document is in a particular language is determined by the closeness of the document's n-grams to the language profile.

In this work we use Bloom Filters [6] to improve an existing FPGA-based n-gram text categorizer, the HAIL architecture [9], to build a highly scalable design. We have implemented our design on the XtremeData XD1000 development system which offers tight coupling between an AMD Opteron microprocessor and an Altera Stratix II FPGA via the high bandwidth, low latency HyperTransport link. Preliminary results show a 1.45× throughput improvement over HAIL, and promises a throughput of 1.4 GB/sec as the communication infrastructure improves.

2. RELATED WORK

N-gram computation is a classic, simple technique underlying many classification problems [11]. Software n-gram processing modules are available in open source form, e.g. in the Apache Lucene project [1], the Ngram Statistics Package [4] and Mguesser [3], or in commercial form as the Lextek Language Identifier [2].

A recent FPGA implementation of the n-gram computation is the HAIL [9] algorithm described as follows:

1. The preprocessing step involves generating the n-gram profile for each language from a representative sample of documents.
2. Given an input document in an unknown language, its n-grams are tested for membership in each language

profile. If a document n-gram is present in a language profile it is considered an n-gram match, and a corresponding counter is incremented.

3. The language profile with the highest n-gram match count value is identified, thus indicating the most probable language of the input document.

The authors of HAIL describe a hardware architecture implemented on an FPGA to perform this computation. The source of bottleneck in the HAIL hardware architecture is the storage of the n-gram profiles for membership testing. An off-chip SRAM is used to store n-gram profiles of up to 255 languages. The amount of parallelism that can be exploited is limited by the number of off-chip SRAMs available, leading to a design that is not easily scalable.

In contrast, the main contribution of our work is the efficient use of on-chip embedded RAMs to store a limited number of n-gram profiles, leading to a more scalable, flexible, and portable architecture.

3. HARDWARE ARCHITECTURE

3.1 Parallel Bloom Filters

A *Bloom Filter* [6] is a probabilistic data structure used to test membership in a large set. The Bloom Filter uses multiple hash functions and a single memory bit-vector. The hash functions are applied to an input element generating address values used to reference the bit-vector. Initially, the entire bit-vector is reset to zero. When programming a data element (i.e, the set of elements to be tested for) into a Bloom Filter, each hash function is first applied on the data element to generate an address. These address locations are then *set* to 1 in the bit-vector. During the *test* operation, the bit values at each of the hash function addresses are read. If they are all set to 1 (i.e, the bitwise AND of all the values is 1), a match results. The Bloom Filter set and test operations are illustrated in Algorithm 1. The size of the bit-vector is typically an order-of-magnitude less than a direct lookup table. However, though a Bloom Filter does not produce false negatives it may generate false positives due to the use of hash functions.

A Bloom Filter is ideally suited for a hardware implementation due to its inherent parallelism: the hashing and the memory lookups can be performed independently. We use the large amounts of distributed on-chip embedded RAMs available on FPGAs to store the bit-vector. Since embedded RAMs have a finite number of addressing ports, there is a limit to the number of hash functions that can address the bit-vector simultaneously. As a result we use a variant, the *Parallel Bloom Filter* [10], where each hash function addresses an *independent* bit-vector implemented with one or more physically distinct embedded RAMs. The Parallel Bloom Filter is a highly scalable data structure for membership testing, and can be implemented efficiently on FPGAs using these distributed memories.

The Parallel Bloom Filter uses k hash functions of the hardware friendly H_3 family [13] and references k $1 \times m$ bit-vectors. A single Parallel Bloom Filter is used to store the n-gram profile of one language. The datapath of the

Algorithm 1 Pseudocode for the Bloom Filter algorithm

```
 1: // Program language profile L into a Bloom Filter
 2: procedure SET(Language profile L)
 3:     // Reset bit-vector
 4:     BitVector ← 0
 5:
 6:     // Program each n-gram in the language profile
 7:     for n-gram w ϵ L do
 8:         for hash function H_i do
 9:             BitVector[H_i(w)] ← 1
10:         end for
11:     end for
12: end procedure
13:
14: // Test document n-grams D
15: procedure TEST(Document n-grams D)
16:     for n-gram w ϵ D do
17:         Val ← 1
18:         for hash function H_i do
19:             Val ← Val AND BitVector[H_i(w)]
20:         end for
21:
22:         if Val = 1 then
23:             // n-gram matches profile
24:         end if
25:     end for
26: end procedure
```

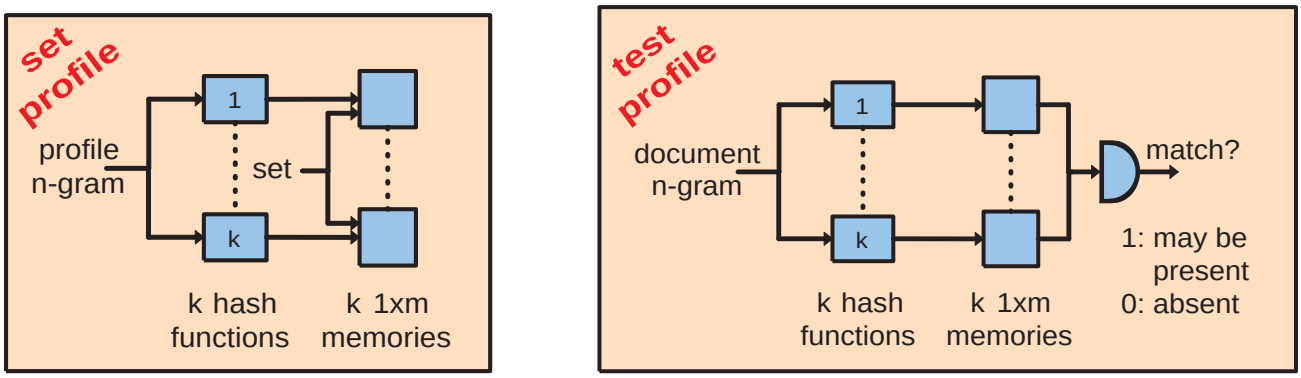
set and test operations are illustrated in Figure 1. Parallel Bloom Filters have been used for accelerating a variety of applications including networking packet inspection [8] and biosequence similarity search [10]. In what follows, we use Parallel Bloom Filters and Bloom Filters interchangeably.

The rate f of false positives of the Parallel Bloom Filter is determined by the number N of n-grams programmed, the number k of hash functions used, and the length m of its bit-vector, and is given by $f = (1-e^{-N/m})^k$. We design our filters to keep the false positive rate to at most a few percent of the sizes of the n-gram profiles used. False positives can adversely impact the language classification accuracy. We study the tradeoff between classification accuracy and the filter parameters in Section 5.2.

3.2 Multiple Language Classifier

To support classification of p languages simultaneously, the Bloom Filter is replicated p times, one for each language, with each instantiation holding a distinct n-gram profile. At initialization the n-gram profiles are programmed sequentially for each language (data path omitted in Figure 1 for brevity). Document n-grams are then tested against each Bloom Filter in parallel, and an associated language profile counter is incremented on a match. The hardware architecture of the multiple language classifier is shown in Figure 2a.

Since each bit-vector is physically implemented using embedded RAMs that are typically dual-ported on modern FPGAs, we modify the Bloom Filter to test two input n-grams simultaneously. This can be easily done by duplicating the hash functions and associated logic to support two independent data paths. Once all n-grams in the document have



(a) Parallel Bloom Filter *set* operation

(b) Parallel Bloom Filter *test* operation

Figure 1: Using a Parallel Bloom Filter for membership testing of an n-gram language profile

been tested, the match counters are read to determine the language with the highest count.

3.3 Parallel Multi-language Classifier

The main advantage of our design is its scalability. The multiple language classifier is easily replicated to test multiple document n-grams in parallel. For example, using four copies of the multiple language classifier, we can simultaneously test 8 n-grams from the input stream. An adder tree aggregates the match counts from the individual classifier modules after the final n-gram in a document is processed.

An alphabet conversion module translates 8-bit extended ASCII characters (ISO-8859) into a 5-bit code similar to HAIL [9]. Lower case characters are converted to upper case, and accented characters are mapped to their non-accented versions. All other characters are mapped to a default white space code. This conversion can be implemented using tables stored in embedded RAMs, or with comparator and muxing logic as in our case.

While our current implementation is limited to common European languages representable with extended ASCII, it can be extended to other encodings such as 16-bit Unicode that have a larger alphabet. The hash functions of the Bloom Filter would simply operate on a larger sized input n-gram, with the rest of the Bloom Filter remaining the same. This is in contrast to an approach that uses a direct memory lookup table to store the n-gram profile, which grows exponentially in the size of the alphabet.

An input word containing multiple translated characters is buffered and an n-gram is generated at each character position. We note that if on-chip memory bandwidth becomes an issue (due to limited number of embedded RAMs on the target FPGA), n-grams can be subsampled from the input stream similar to HAIL. Our implementation is currently oblivious to word boundaries and simply treats the input as a continuous stream of characters.

4. IMPLEMENTATION DETAILS

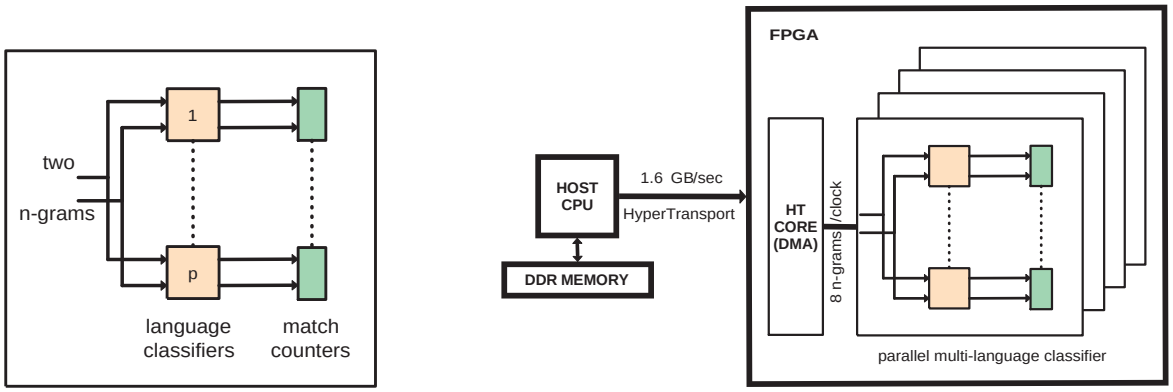
The XtremeData[1] architecture combines an AMD Opteron processor with an FPGA on a dual-socket mother board (see Figure 3). The Opteron is dual-core, and runs at 2.2 GHz. The FPGA is an Altera Stratix EP2S180F1508-C3. The Opteron and Altera each have 4 GB of DRAM, and the Altera can also use 4MB of SRAM. The processor and FPGAs communicate over non-coherent hypertransport, which has a peak bandwidth of 1.6 GB/sec in each direction. Currently, the XtremeData system's maximum throughput is 500 MB/sec. For this application, the design environment provided by XtremeData was used, including the HT core and communication protocol. The application was coded in VHDL and compiled through the Altera Quartus II tool chain version 7.1. The Bloom Filter based classifier is also easily ported to a comparable Xilinx FPGA and can be compiled using the Xilinx ISE tool chain.

Registers on the FPGA can be mapped into the operating system's memory space, providing access to hardware control registers. Bulk data transfer is done via DMA. The DMA controller reads 64-bit words from the DDR memory (4 GB on our system) connected to the Opteron processor. The DMA controller is set up for data transfers from software using the control register interface. The DMA interface is used to program n-gram profiles and transfer input documents from the host processor to the FPGA. A simplified block diagram of our n-gram classifier hardware on the XtremeData system is shown in Figure 2b.

In our implementation we use n-grams of size 4. We use the top $t = 5,000$ most frequently occurring n-grams from a language training set to generate a profile. As noted in HAIL [9] this produces over 99% accuracy for the language classifier.

During the preprocessing step, the bit-vectors are cleared and the n-gram profiles for each language are programmed. Multiple text documents stored in the processor's DDR memory are streamed to the hardware, delimited by an *End of Document* command. Since we use the register interface to send commands to the classifier module and DMA to transfer document data, they appear asynchronously (and poten-

[1]http://www.xtremedatainc.com/



(a) Multiple language classifier

(b) Parallel Multi-language Classifier on the XtremeData system

Figure 2: Parallel Bloom Filter based n-gram classifier hardware

Figure 3: XD1000 FPGA Module

tially out of order) in the hardware. To handle this issue, we first send a *size* command before each document transfer to indicate the number of 64-bit words to expect for the document. Subsequent commands are only processed once all the words expected have been received via DMA. We provide a watchdog timer to reset the state machine in case of an error.

A *Query Result* command sends the document classification results to the software. From the match counts it is possible to identify the most likely language(s) present in the document. In addition, the hardware sends an xor data checksum and other status bits used to verify a valid document transfer.

5. PERFORMANCE

The HAIL algorithm (using direct memory lookup tables) was tested for accuracy in the HAIL [9] project. In this section we compare the accuracy of the language classifier using Bloom Filters of various parameters for membership testing. We use the 768 4 Kbit embedded RAMs available on the FPGA for the bit-vectors. Increasing the number of embedded RAMs dedicated to the bit-vector increases classification accuracy, but also reduces the number of languages that can be tested simultaneously.

Table 1: Variation of classification accuracy with Bloom Filter parameters

m (Kbits)	k	False positives (per thousand)	Average Accuracy
16	4	5	99.45%
16	3	18	97.42%
16	2	69	97.31%
8	4	44	99.42%
8	3	95	97.22%
8	2	209	95.57%
4	6	123	99.41%
4	5	174	96.44%

We measured performance using the JRC-ACQUIS Multilingual Parallel Corpus, Version 3.0[2] [12]. This corpus is the body of European Union law applicable to the EU member states available in 22 European languages. We used 10 languages: Czech, Slovak, Danish, Swedish, Spanish, Portuguese, Finnish, Estonian, French and English. For our tests we parsed a subset of the corpus with only the text body saved to individual files. There were an average of $5,700$ documents for each language, with an average of $1,300$ words per document. We used 10% of the corpus as the training set for each language, and tested the classifier on the remaining documents.

5.1 Classification Accuracy

In our first experiment we used $k = 4$ hash functions and $m = 16$ Kbits. To support ten languages testing 8 n-grams per clock, the design uses 640 4 Kbit embedded RAMs. In

[2]http://wt.jrc.it/lt/Acquis/

Table 2: Resource utilization of the n-gram classifier module for various Bloom Filter parameters

m (Kbits)	k	Logic Utilization	Logic Registers	M4Ks	Frequency (MHz)
16	4	5480	3849	128	182
16	3	4441	3340	96	189
16	2	3547	2780	64	191
8	4	4760	3722	64	194
8	3	4072	3229	48	202
8	2	3363	2713	32	202
4	6	5458	4471	48	197
4	5	4983	4006	40	198

Table 3: Resource utilization of the n-gram classifier hardware for the final implementation

k, m	Languages	Logic Utilization	Logic Registers	M512s	M4Ks	M-RAMs	Frequency (MHz)
4, 16 Kbits	10	38891	27889	36	680	9	194
6, 4 Kbits	30	85924	68423	66	768	6	170

this configuration, the expected false positive rate of the Bloom Filter is five in one thousand. Tests on our corpus show that the accuracy of the classifier varies between 99.05% and 99.76% with an average of 99.45%. This agrees well with the results of the HAIL architecture which uses direct memory to perform membership testing, and is comparable to software algorithms used for language identification [12].

We note that the false positive rate of the Bloom Filter in this configuration has little effect on the overall accuracy of the classifier. In most cases, the difference in match counts between the two highest scoring languages is significantly larger than the false positive rate.

5.2 Accuracy and Parallelism Tradeoff

In this section we study the effect of the Bloom Filter parameters on its accuracy. Recall that the rate of false positives is determined by the number N of n-grams programmed, the number k of hash functions used, and the length m of its bit-vector, and is given by $f = (1 - e^{-N/m})^k$. Since N is fixed, we can vary k and m which are directly proportional to the accuracy.

We looked at reducing the number of embedded RAMs dedicated to the Bloom Filter while still achieving acceptable classification accuracy. We achieved this by decreasing the size of the bit-vector and varying the number of hash functions. By reducing the number of embedded RAMs used per Bloom Filter, and hence per language, we are able to classify a larger number of languages simultaneously, while still testing 8 n-grams per clock for maximum throughput.

Table 1 shows the classification accuracy of the corpus for various Bloom Filter parameters. The expected false positive rate is also shown for comparison.

The most conservative configuration uses $k = 4$ hash functions with $m = 16$ Kbits (four embedded RAMs on our target) allocated to each bit-vector. Though this provides the highest accuracy, an implementation on our target FPGA supports only twelve languages at the desired throughput. Keeping m constant and reducing k leads to a substantially decreased accuracy of just over 97%.

Reducing the number of embedded RAMs per bit-vector to two (8 Kbits), we achieve greater than 99% accuracy with four hash functions. We were able to maintain this accuracy using $k = 6$ hash functions and just one embedded RAM (4 Kbits) per bit-vector. This leads to the most space-efficient configuration and uses just 24 Kbits per language. Our final implementation on our target FPGA is therefore able to support thirty languages at the desired throughput.

Table 2 shows the resource utilization for the n-gram classifier hardware module (without infrastructure code) with two languages accepting eight n-grams per clock. Using smaller sized bit-vectors, or fewer hash functions has a favourable impact on the logic utilization in addition to embedded RAM usage. Additionally, with fewer embedded RAMs per bit-vector the routing of the design is made easier, thereby increasing the clock frequency.

It is important to note that these results are dependent on the languages used. We have tried to account for this fact by selecting similar languages in our corpus. For example, consistently more Spanish documents were misclassified as Portuguese, and Estonian documents as Finnish. In cases of highly similar languages, the expected false positive rate must be given more importance and a larger number of embedded RAMs dedicated to the bit-vectors, with a corresponding decrease in the number of languages supported.

There are two further avenues for increasing the number of languages supported. A large fraction of 512 bit embedded RAMs remain unutilized on the target FPGA which may be used to support an additional four languages. Secondly, it is possible to sub-sample the input stream to test only every other n-gram, similar to the HAIL implementation. This doubles the number of supported languages while maintaining satisfactory accuracy.

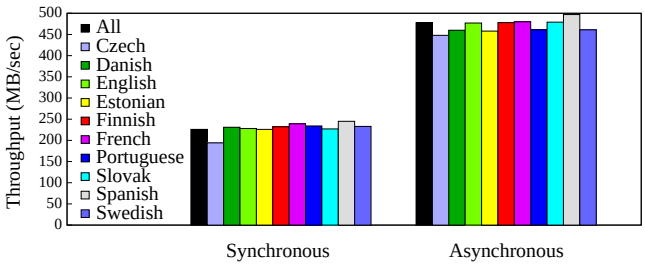


Figure 4: Throughput of the n-gram classifier hardware

5.3 Device Utilization

Table 3 shows the device utilization of the n-gram classifier hardware for two configurations on the EP2S180 FPGA, supporting 10 and 30 languages respectively. Both versions accept 8 n-grams per clock and run at 194 and 170 MHz respectively. The logic elements used vary between a third and two-thirds of the total, with less than half the total registers on the FPGA being used. These numbers include resources allocated to infrastructure code (about 10%) such as the HyperTransport core, DMA controller and command control logic. The limiting factor in our design is the number of embedded RAMs available on-chip.

5.4 System Throughput

To measure the throughput of the system we used the configuration with $k = 4$, $m = 16$ Kbits accepting 8 n-grams per clock, placed and routed at 194 MHz. The theoretical rate at which our design can accept document n-grams is therefore 194 MHz $\times 8 = 1,552$ million n-grams per second. Since each n-gram corresponds to a byte in the input stream, our design can perform language classification at a peak rate of 1.4 GB/sec. This is well within the bandwidth of 1.6 GB/sec provided by the HyperTransport bus. However, the revision of the XtremeData machine we used achieves only a maximum of 500 MB/sec and so limits the practical performance realized by our design.

In addition to the hardware, the software is an important component of the system. Our first version of the software had tight synchronization between the hardware and software components. After a successful transfer of a document via the DMA interface, the software requests a hardware interrupt after which the match counters are read and the results displayed to the user. The hardware interrupt was used as a synchronization point to ensure correctness.

Our second version removed explicit synchronization and was coded without interrupts. The hardware was modified to stop accepting commands until an entire document was read via DMA. With this modification synchronization is no longer necessary and ensures correct operation. A software thread then sends multiple documents without synchronization, while another waits for classification results returned by an FPGA initiated DMA transfer.

To measure system performance, we used the test document set from the JRC-ACQUIS Multilingual Parallel Corpus mentioned previously. The average file size of a single language corpus was 48 MB and the average size of an individual file was 10 KB. We measured the wall clock time for the transfer of documents and receipt of results in memory. The measured time did not include the Bloom Filter programming time or I/O time to read documents from secondary storage to memory. The former is a setup cost that can be amortized over large runs. With regard to the latter, our current system is not optimized for high bandwidth reads from secondary storage, so our performance measure does not take this into account. Finally, the preprocessing step to generate the n-gram profiles was not included in the timing as it is considered a one-time cost prior to classification.

Table 4: Comparison of n-gram based language classifiers

System	Type	Throughput (MB/sec)
Mguesser	AMD Opteron Workstation	5.5
HAIL	Xilinx XCV2000E-8 FPGA	324
BloomFilter	Altera EP2S180 FPGA	470

Figure 4 plots the throughput in MB/sec of the language classifier for the ten sets of documents. In addition, we measured the throughput of documents from all ten languages which consisted of $52,581$ documents in a corpus of size 484 MB. The version of the software with tight synchronization shows half the throughput of the asynchronous version. Clearly, interrupt based synchronization produces detrimental performance for a streaming architecture.

We observed a throughput of 228 MB/sec and 470 MB/sec for the two versions respectively. This remained consistent across the document sets of the various languages and holds for files with sizes varying from a few Kilobytes to several Megabytes. If the Bloom Filter programming time is included, the throughput of the asynchronous version drops to 378 MB/sec. However as mentioned previously, this is easily amortized over larger datasets.

We also note that our observed throughput is close to the maximum bandwidth available on the current revision of the XtremeData system. We expect it to increase substantially as the communication infrastructure improves.

5.5 Comparison with other solutions

To measure comparative performance of our Bloom Filter based classifier, we measured the system throughput of Mguesser [3], an optimized version of the n-gram based text categorization algorithm [7]. Mguesser was run on an AMD Opteron Processor at 2.4 GHz with 16 GB of memory. Ten languages were used for identification and the program was run on documents of size 81 MB. The documents were cached in memory before a timing run. Table 4 compares the performance of the two systems. The average throughput of Mguesser was measured to be 5.5 MB/sec. In comparison, our implementation is 85× faster.

Compared to the HAIL implementation [9] our hardware runs 1.45× faster on ten languages. We note that while

HAIL is able to classify up to 255 languages at this rate, our hardware is limited to between 10 - 30 languages by the number of on-chip embedded RAMs. The advantage of our approach is that the design is flexible and scalable, allowing the designer to trade off between number of hash functions and bit vector size. In addition, it does not depend on a specific external memory configuration, making it easily portable to other boards. As higher input bandwidth becomes available, our design can take advantage of parallelism offered by Bloom Filters implemented on-chip. In contrast, HAIL is limited by the number of off-chip SRAM resources.

Once the HyperTransport communication infrastructure is improved, we expect to see performance increase closer to the theoretical peak of 1.4 GB/sec for ten languages. At this rate, our system is 260× faster than the software baseline and 4.4× faster than HAIL.

6. CONCLUSIONS

We have developed a new FPGA implementation of n-gram frequency counting and demonstrated its use in language classification. Our hardware/software design has been mapped to a modern FPGA co-processor on the XtremeData XD1000. Our design has been optimized for maximum scalability, flexibility, and portability, and can trade off between number of hash functions and size of look up table depending on the accuracy and number of languages to be processed. The design runs at 85× software and 1.45× a comparable hardware implementation. As the XD1000 hardware/software communication bandwidth improve, our design is capable of processing ten languages at 1.4 GB/sec.

7. ACKNOWLEDGMENTS

We gratefully acknowledge assistance provided by Dr. Marcus Miller and Lisa Corsetti with the administration of the XtremeData machine. This work was funded by the Lawrence Livermore National Laboratory LDRD program's Storage-Intensive Supercomputing project under DOE contract W-7405-ENG-48.

8. REFERENCES

[1] Apache Lucene. `http:lucene.apache.org`.

[2] Lextek Language Identifier. `http://www.lextek.com/langid/li/`.

[3] Mguesser. `http://www.mnogosearch.org/guesser/`.

[4] Ngram Statistics Package. `http://ngram.sourceforge.net`.

[5] SpamAssassin. `http://spamassassin.apache.org/`.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[7] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[8] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Deep packet inspection using parallel bloom filters. *IEEE Micro*, 24(1):52–61, 2004.

[9] C. M. Kastner, G. A. Covington, A. A. Levine, and J. W. Lockwood. HAIL: A hardware-accelerated algorithm for language identification. In *15th Annual Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug. 2005.

[10] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, A. Jacob, and J. Lancaster. Biosequence similarity search on the *Mercury* system. *Journal of VLSI Signal Processing*, published online July 2007.

[11] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge (Mass.) and London, 1999.

[12] S. Ralf, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, Tufis, and D. Varga. The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In *5th International Conference on Language Resources and Evaluation (LREC'2006)*, Genoa, Italy, May 2006.

[13] M. V. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, 46:1378–1381, 1997.