Evaluating FPGA-acceleration for Real-time Unstructured Search

SaiRahul Chalamalasetti, Martin Margala, Wim Vanderbauwhede[+], Mitch Wright[*], Parthasarathy Ranganathan[#]
University of Massachusetts Lowell, Lowell, MA, USA [*]Hewlett Packard, Houston, TX USA
[+]University of Glasgow, Scotland, UK [#]Hewlet Packard Labs, Palo Alto, CA, USA

***Abstract*—Emerging data-centric workloads that operate on and harvest useful insights from large amounts of unstructured data require corresponding new data-centric system architecture optimizations. In particular, with the growing importance of power and cooling costs, a key challenge for such future designs is to achieve increased performance at high energy efficiency. At the same time, recent trends towards better support for reconfigurable logic enable the use of energy-efficient accelerators. Combining these trends, in this paper, we examine the applicability of acceleration in future data-centric system architectures. We focus on an important class of data-centric workloads, real-time unstructured search, or information filtering, where large collections of documents are scored against specific topic profiles, and present an FPGA-based implementation to accelerate such workloads. Our implementation, based on the GiDEL PROCStar IV board using Altera Stratix IV FPGAs, demonstrates excellent performance and energy efficiency, 20 to 40 times better than baseline server systems for typical usage scenarios. Our results also highlight interesting insights for the design of accelerators in future data-centric systems.**

I. Introduction

The amount of digital data is growing exponentially, even outpacing Moore's law. A recent report [1] estimated (conservatively) that enterprise servers processed and moved over 9 zettabytes (1 zettabyte = 10^{21} bytes) in 2008; this number is projected to double every two years. An increasing amount of this data growth is attributed to unstructured or semi-structured data such as, documents, web pages, images, videos, etc. For example, Youtube reports that more than 13 million hours of video were uploaded in 2010 alone, and approximately 48 hours of video continue to be uploaded every minute. Facebook estimates more than 100 TB of log data generated every day. This explosive growth in data has led to a corresponding growth in new workloads – consumer, enterprise, scientific – to collect, store, access, visualize, analyze, and interpret all this new information. Some recent examples include search, live business analytics, social correlation, collaborative filtering, etc. A common characteristic of these emerging workloads is the need to achieve better performance for deeper insights at larger scale (and lower costs).

At the same time, power and cooling have become important constraints in the design and operation of data centers. Many cloud data centers ("warehouse-scale computers" [2]) reportedly spend millions of dollars every month in electricity costs, and face significant challenges in the design of the associated power delivery and cooling infrastructure. Also, the energy consumption of such computing equipment has a significant environmental impact in terms of unwanted carbon emissions and destruction of natural resources. Several recent initiatives, from both governmental agencies like the EPA and industry consortiums like the Green Grid, now seek to address the environmental impact of datacenter operation.

The combination of these two trends motivates the need for new energy-efficient system designs for future data-centric workloads. Given the parallelism in many data-centric algorithms, FPGA-based acceleration can potentially help achieve these goals. Additionally, several other recent trends also motivate a focus on FPGAs. First, FPGA platforms are gaining adoption in the enterprise market, as evidenced by recent products from Fusion-IO and Netezza (IBM), and the growing adoption of FPGAs in the high-performance computing community [3], [4], [5]. Also, many vendors have recently announced products that support better integration of FPGAs and general-purpose CPUs, such as Xilinx' Extensible Processing Platform, Altera's Embedded Initiatives with ARM dual-core Cortex-A9 processors, and the Intel E600C products with integrated Atom cores and FPGAs. In addition, several vendors (e.g., the GiDEL PROCStar or PLDA XpressGX4 development platforms) now provide pre-defined modules for interfaces and common functions that significantly simplify the porting of algorithms to FPGA-based accelerators.

We therefore evaluate the potential of FPGA-based acceleration for energy-efficient data-centric system design. We focus on an important class of data-centric workloads – real-time unstructured search, or information filtering, where given a set of information needs, documents are matched against topic profiles. This task is representative of a large class of important data-centric workloads, for example detecting spam, comparing patent applications against existing patents, monitoring communications for terrorist activity, news story topic detection and tracking, etc. We present an FPGA design to accelerate the computationally intensive part of this workload and implement our design on an existing FPGA platform (GiDEL PROCStar + Altera Stratix IV). Our results show that even a conservative FPGA design can achieve high performance at low power – nearly 800 million terms/second with 6W per FPGA chip – and with a relatively small utilization of the FPGA (4% logic utilization and 22% memory utilization). Compared to baseline server systems, our FPGA implementation shows significantly higher performance and energy efficiency, 20 to 40 times for typical usage scenarios, and lesser variation to changes in input parameters such as the document and profile sizes.

978-1-4673-1146-5/12/$31.00 ©2012 IEEE

The rest of the paper is organized as follows. Section II discusses our choice of workload and describes the algorithm we focus on. Section III describes our FPGA platform and our design. Section IV presents our evaluation results and our analyses. Section V discusses related work, and Section VI concludes the paper.

II. Real-time Unstructured Search

A. Choice of Workload

There is a rich diversity in data-centric workloads and how they work with data. Of the various operations that can be performed on data – collect and distribute, maintain and manage, organize and analyze – in this paper, we mainly focus on the last category. There are different classes of operations associated with organization and analysis of data including ad-hoc retrieval, classification, clustering, and filtering. However, fundamentally, all of these tasks require the matching between information items and information needs; for example, a user query or a pre-determined profile matched against data content. Representative of these operations, in this paper, we focus on real-time unstructured search or information filtering where *given a collection of unstructured data sources (e.g., documents), we identify the match to a pre-determined weighted feature vector profile (e.g., topic signatures, keywords)*. With the growing increase in unstructured and semi-structured data, this class of workloads is likely to be more important in the future. Some example applications include searching patent repositories for related work comparison, searching emails and sharepoints for enterprise information management, detecting spam in incoming emails, monitoring communications for terrorist activity, news story topic detection and tracking, searching through books, images, and videos for matching profiles.

B. Algorithm Description

In this work, we employ the model proposed by Lavrenko and Croft [6] for our algorithm. Applied to the task of information filtering, the idea is that the odds of an incoming document being relevant to a topic profile is determined using a generative probabilistic language model. If a document scores above a user defined threshold then it is considered relevant to the topic profile. The algorithm we implement can be expressed as follows:

- Each document is modeled as a "bag of words", i.e. a set D of pairs (t, f) where $f \triangleq n(t, d)$ is the number of occurrences of the term t in the document d; $t \in \mathbb{N}$ is the term identifier. We assume all documents are converted to the bag of words representation through one prior offline sequential pass. (This approach is also followed by the Google n-gram project [7].) Note that a "term" can correspond to a single word, a *n*-gram, or a fixed sequence of *n* words. Inclusion of bigrams and trigrams, removal of stop words and stemming (reducing words to their root) leads to better search results.[1]
- The profile M is a set of pairs $p = (t, w)$ where the weight $w \triangleq \log\left(\frac{(1-\lambda)P(t|M)}{P(t)} + \lambda\right)$. Typically, a bayesian algorithm is used offline to precompute the profile based on specific user requirements.
- For T, a set of terms occurring both in D and M, the score of a given document against a given profile is then given by:

$$score(D, M) = \sum_{\forall k \in T} f_k w_k \quad (1)$$

This function is representative of the dominant kernel of most filtering algorithms, the main difference being the weighting of terms in profiles. For example, the kernel can be used for Naïve Bayes spam filtering, Support Vector Machine classification, Relevancy Feedback information filtering and even image recognition.

[1]Note that the inclusion of n-grams makes our vocabulary size much larger than expected. For example, the Oxford Dictionaries FAQ indicates about a quarter of a million distinct English terms, but the vocabulary size for our collection is 16 million terms.

III. FPGA Implementation

A. Platform

We implemented our algorithm on the GiDEL PROCStar-IV development board (Figure 1). This system provides an extensible high-capacity FPGA platform and supports the GiDEL PROC-API library-based developer kit for interfacing with the FPGA.

Hardware. Each board contains four Altera Stratix-IV 530 FPGAs running at 150MHz. Each FPGA supports a five level memory structure, with three kinds of memory blocks embedded in the FPGA:

- 6,640 MLAB RAM blocks (320 bits each),
- 1280 M9K RAM blocks (9K bits each), and
- 64 M144K blocks (144K bits each)

and two kinds of external DRAM memory:

- 512 DDR2 SDRAM on-board memory (Bank A) and
- two 2GB SODIMM DDR2 DRAM memories (Bank B and Bank C).

The embedded FPGA memories run at a maximum frequency of 300MHz, Bank A, Bank B and Bank C at 667MHz. The FPGA-board is connected to the host platform via a PCI Express bus. The host computer transfers data to the FPGA using 32-bit DMA channels. We studied two different host systems: a system based on HP BL460 blade servers with quad-core 64-bit Intel Xeon X5570 at 2.93GHz and 3.5GB DDR2 DRAM memory, running 32-bit Windows XP, and a second system based on a dual-core 64-bit Intel Atom board at 1 GHz with 4GB DDR2 DRAM memory, running 64-bit Ubuntu GNU/Linux; in this paper, we primarily focus on the former.

Development environment. FPGA-accelerated applications for the PROCStar board are implemented in C++ using the GiDEL PROC-API libraries for interacting with the FPGA. This API defines a hardware abstraction layer that provides control over each hardware element in the system – for example, Memory I/O is implemented using the GiDEL MultiFIFO and MultiPort IPs. To achieve optimal performance, we implemented the FPGA algorithm in VHDL (as opposed to Mitrion-C). The Altera Quartus toolchain is employed to create the bitstream for the FPGA.

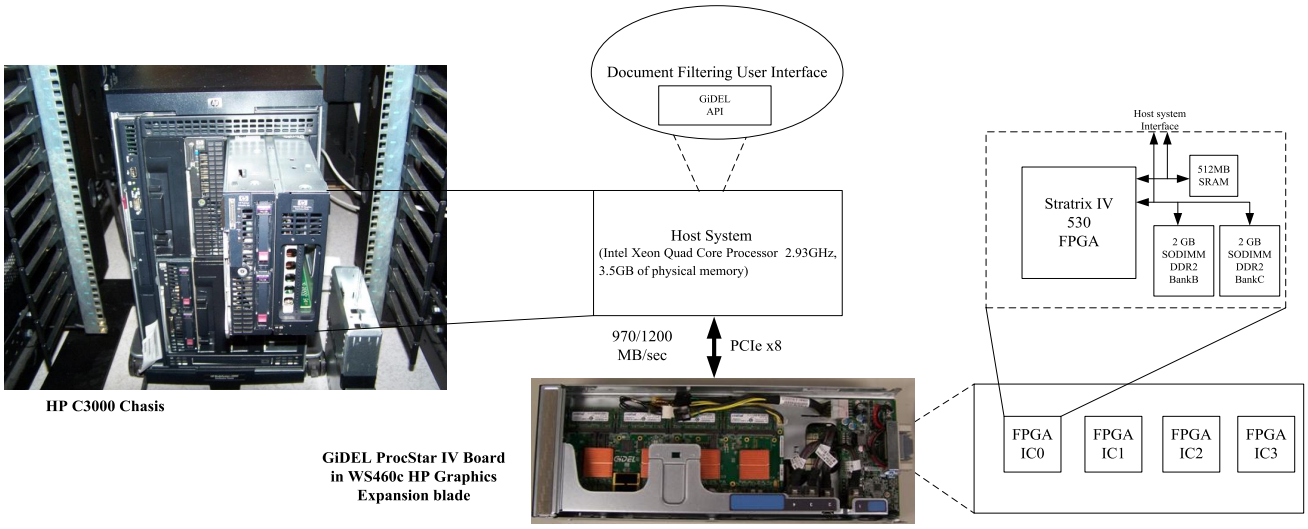

Figure 1. Block diagram of FPGA platform and photograph of experimental hardware

B. FPGA Implementation Description

Figure 2 presents the overall workflow of our implementation. The input stream of document term pairs is read from the two SDRAM blocks via a FIFO. A Bloom filter is used to discard negatives (terms that do not appear in the profile) for multiple terms in parallel. Profile weights are read corresponding to the positives, and the scores are computed for each term in parallel and accumulated to achieve the final score described in Equation 1. Below, we describe the key modules for the implementation: document streaming, profile negative hit filtering, and profile lookup and scoring.

Document streaming. Using a bag-of-words representation (see Section II-B) for the document, the document stream is a list of *(document id, document term tuple set)* pairs. Physically, the IO width of the FPGA is 64 bits for each SDRAM memory. As the SDRAM is clocked at 667 MHz, the ProcMultiPort FIFO combines two contiguous 64-bit words into a single 128-bit word for each memory bank. The document term tuple $d_i = (t_i, f_i)$ can be encoded in 32 bits: 24 bits for the term id (supporting a vocabulary of 16 million terms) and 8 bits for the term frequency, so the algorithm can process 8 terms (256 bits) in parallel. To mark the start and end of a document we insert a marker word (64 bits) followed by the document id (64 bits).

Profile negative hit filtering. For every term in the document, the application needs to look up the corresponding profile term to obtain the term weight. As the profile is stored in the external SDRAM, this is an expensive operation (typically 20 cycles per access). The purpose of document filtering is to identify a small amount of relevant documents from a very large document set. As most documents are not relevant, most of the lookups will fail (i.e. most terms in most documents will not occur in the profile). Therefore, it is important to discard the negatives first. For this reason, we implemented a Bloom filter in the FPGA's on-chip MRAM (M9K blocks).

a) "Trivial" Bloom Filter : A Bloom filter [8] is a data structure used to test membership of a set. False positives are possible, but false negatives are not. In this context, the design we use to reject negatives is a Bloom filter. However, in most cases a Bloom filter uses a number (k) of hash functions to compute several keys for each element in the set, and adds the element to the table (assigns a "1") if element is in the set. As a result, hash collisions can lead to false positives.

Our Bloom filter is a "trivial" edge case of such a more general implementation: our hashing function is the identity function $key = elt$, and we only use a single hash function ($k = 1$) so every element in the set corresponds to exactly one entry in the Bloom filter table. As a result, the size of the Bloom filter is the same as the size of the set and there are no false positives. Furthermore, no elements are added to the set at run time.

b) Bloom Filter Dimensioning: The internal block RAMs of Altera Stratix-IV FPGA that support efficient single-bit access are M9K memory modules (around 1280 blocks available on Stratix-IV 530 FPGA). The current generation of the Bloom Filter is limited to 4Mb; however, for future work, we will port our design to an 8Mb Bloom Filter to use all 1280 M9K blocks [9].On the other hand, the vocabulary size of our document collection is 16M terms (based on English documents using unigrams, diagrams and trigrams). We therefore used a very simple "hashing function", $key = elt \gg 2$. Thus we obtain one entry for every four elements, which leads to three false positives out of four on average. This obviously results in a four times higher access rate to the external memory than if the Bloom filter would be 16 Mb. As the number of positives in our application is very low, the effect on performance is limited. The higher internal bandwidth of the MRAMs leads to very fast rejection of negatives. Although the MRAM is fast, concurrent lookups lead to contention. To reduce contention we designed a distributed Bloom filter. The Bloom filter memory is distributed over a large number of banks (16 in the

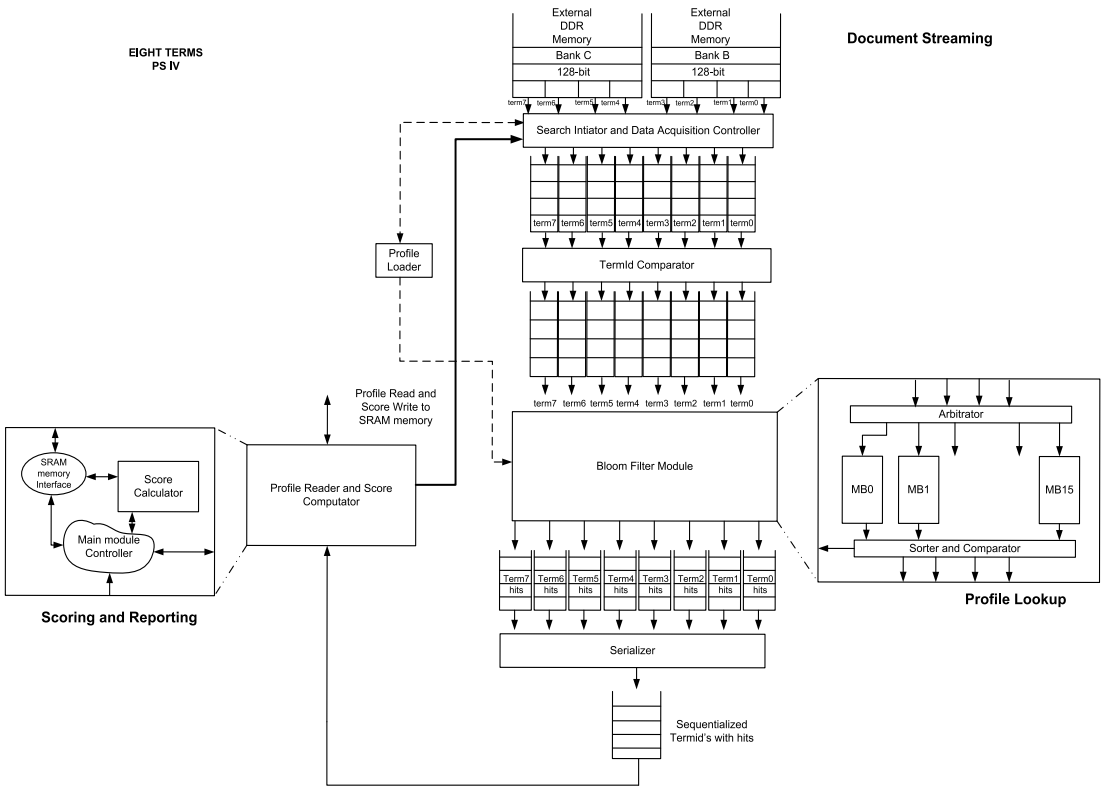

Figure 2. Overall block diagram of FPGA implementation.

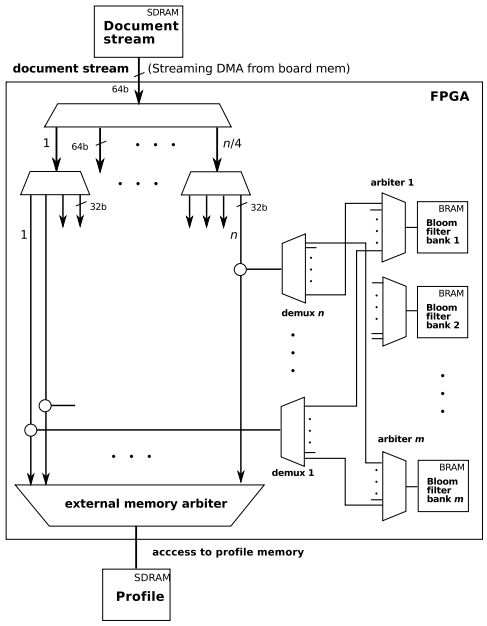

Figure 3. Parallelizing lookups using a multi-bank Bloom filter

current design) and a cross-bar switch connects the document terms streams to the banks. In this way contention (when multiple tuples access same bank) is significantly reduced.

Profile lookup and scoring. Because of the need for lookup, the profile must be implemented as some type of map (dictionary). A hash function is an obvious approach; however, as the size of the profile is not known in advance, it is impossible to construct a perfect hash; imperfect hashes suffer from collisions which deteriorate the performance. When the key space is very large, the contention probability is high.

Our solution is simply to use the term id as the memory address and to implement the weight lookup from the profile as a content-addressable data structure from the on-board SDRAM (Bank A). As the upper limit to the vocabulary size in our case is 16 million terms, we require 128MB of memory; the PROCStar-IV provides 512MB of on-board SDRAM. Note that it is possible to increase the vocabulary size to exceed the memory capacity, with a very low performance penalty [10].

Using the lookup table architecture and document stream format as described above, the actual lookup and scoring system is quite straightforward: the input stream is scanned for header and footer words. The header word action is to set the document score to 0; the footer word action is to collect and output the document score. For every two terms in the document, first, the Bloom filter is used to discard negatives and then the weights corresponding to positives are read from the SDRAM. The score is computed for each of the terms in parallel and added. The score is accumulated for all terms in the document and finally the score stream is filtered against a limit before being output to the host. Figure 4 summarizes the implementation of the profile lookup and scoring.

Discussion. The implementation above leverages the advantages of an FPGA-based design, in particular the memory

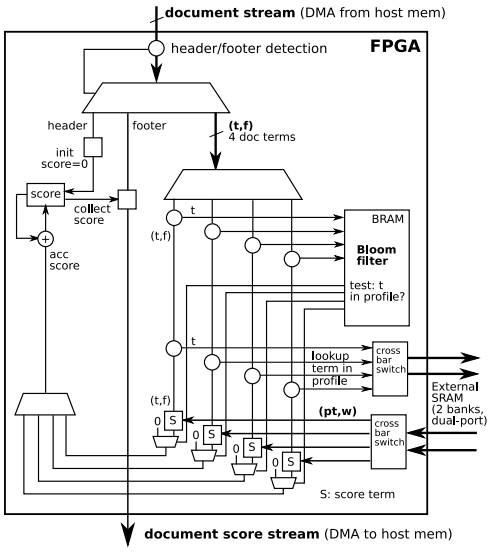

Figure 4. Implementing profile lookup and scoring.

architecture of the FPGA; on a general-purpose CPU-based system it is not possible to create a very fast, very low-contention Bloom filter to discard negatives. Also, a general-purpose CPU-based system only has a single, shared memory. Consequently, reading the document stream will contend for memory access with reading the profile terms, and as there is no Bloom filter, we have to look up each profile term. (We could of course implement a Bloom filter but as it will be stored in main memory as well, there is no benefit: looking up a bit in the Bloom filter is as costly as looking up the term directly). Furthermore, the FPGA design allows for lookup and scoring of several terms in parallel.

FPGA utilization details. Our implementation used only 17,652 of the 424,960 Logic Elements (LEs) or a 4% utilization of the logic in the FPGA, and 4,579,824 out of 21,233,664 bits for a 22% utilization of the RAM. Of the 17,652 LEs utilized by whole design on the FPGA, the actual Document Filtering algorithm only occupied 4,561 LEs, which is less than 1% of utilization, and rest was used by the GiDEL Memory IPs. The memory utilized for the whole design (4,579,824 bits) was mainly for the Bloom Filter that is mapped on Embedded Memory blocks (i.e., M9k). The Quartus PowerPlay Analyzer tool estimates the power consumption of the design to be 6W. The largest contribution to the power consumption is from the memory I/O.

IV. Evaluation Results

Below, we discuss our evaluation results. We first discuss our experimental methodology, and then present data summarizing the performance of our FPGA implementation and comparison with non-FPGA-accelerated baselines, and finally conclude with the learnings from our experiments.

A. Creating Synthetic Data Sets

To accurately assess the performance of our FPGA implementation, we need to exercise the system on real-world input data, however it is hard to get access to such real-world data: large collections such as patents are not freely available, and governed by licenses that restrict their use. For example, although the researchers at Glasgow have access to the TREC Aquaint collection and a large patent corpus, they are not allowed to share these with a third party. In this paper, therefore, we use synthetic document collections statistically matched to real-world collections. Our approach is to leverage summary information about representative data-sets to create corresponding language models for the distribution of terms and the lengths of documents; we then use these language models to create synthetic data sets statistically identical to the original data sets. In addition to addressing IP issues, synthetic document collections have the advantages of being fast to generate, not taking up large amounts of disk space and easy to experiment with.

Collection	# Docs	Avg. Doc. Len.	Avg. Uniq. Terms
Aquaint	1,033,461	437	169
USPTO	1,406,200	1718	353
EPO	989,507	3863	705

Table I
Summary statistics from representative real-word collections that we used as templates for our synthetic data sets.

Real-world document collections. We use summary information from several document collections – a newspaper collection (TREC Aquaint) and two collections of patents from the US Patent Office (USPTO) and the European Patent Office (EPO). These collections provide good coverage on the impact of different document lengths and sizes of documents on filtering time. We used the Lemur[2] Information Retrieval toolkit to determine the rank frequency distribution for all the terms in the collection. Table I shows the summary data from the collections we studied as templates.

Modeling distributions of terms. It is well known (see e.g [11]) that the rank-frequency distribution for natural language documents is approximately Zipfian:

$$f(k;s;N) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^s}$$

where f is frequency of term with rank k in randomly chosen text of natural language, N is number of terms in the collection, and s is an empirical constant. If $s > 1$, the series becomes a value of a Riemann ζ-function and will therefore converge. This type of distribution approximates a straight line on a log-log scale. Consequently, it is easy to match this distribution to real world data with linear regression.

Special purpose texts (scientific articles, technical instructions, etc.) follow variants of this distribution. Montemurro [12] has proposed an extension to Zipf's law which better captures the linguistic properties of such collections. His proposal is based on the observation that in general, after some pivot point p, the probability of finding a word of rank r in the text starts to decay much faster than in the beginning. In

[2]www.lemurproject.org

other words, in log-log scale, the low-frequency part of the distribution has a steeper slope than the high-frequency part. Consequently, the distribution can be divided into two regions each obeying the power law, but with different slopes:

$$F(r) = \begin{cases} a_1 r + b_1 & r < p \\ a_2 r + b_2 & otherwise \end{cases}$$

We determine the coefficients a_1, a_2, b_1, b_2 from curve-fitting on the summary statistics from the real-world data collections. Specifically, we use the sum of absolute errors as the merit function combined with a binary search to obtain the pivot. We then use a least-squares linear regression, with χ^2 statistics as a measure of quality (taken from [13]). A final normalization step is added to ensure that the piecewise linear approximation is a proper probability density function.

Modeling document lengths. Document lengths are sampled from a truncated Gaussian. The hypothesis that the document lengths in our template collections have a normal distribution was verified using a χ^2 test with 95% confidence. The sampled values are truncated at the observed minimum and maximum lengths in the template collection.

Once the models for the distribution of terms and document lengths are determined, we use these models to create synthetic documents of varying lengths. Within each document, we create terms that follow the fitted rank-frequency distribution. Finally, we convert the documents into the standard bag-of-words representation, i.e., as a set of unordered (term, frequency) pairs.

B. Experimental Parameters

Statistically, the synthetic collection will have the same rank-frequency distribution for the terms as the original data sets. Consequently, the probability that a term in the collection matches a term in the profile will be the same in the synthetic collection and the original collection. The performance of the algorithm on the system now depends on (1) the size of the collection, (2) the size of the profile, and (3) the "hit probability" of the bloom filter, i.e., the probability that the profile corresponding to a term has a non-zero weight. To evaluate these effects, we studied a number of different configurations - with different document sizes, different profile lengths, and different profile constructions. Specifically, we studied profile sizes of 4K, 16K, and 64K terms, the first two are of the same order of magnitude as the profile sizes for TREC Aquaint and EPO as used in our previous work [10] and the third, larger profile was added to investigate the impact of the profile size. We studied two different document collections: 128K documents of 2048 terms, which is representative for the patent collections, and 512K documents of 512 terms, similar to the Aquaint collection. Note that the total size of the collection is not important for the performance evaluation: for both the CPU and FPGA implementation, the time taken to filter a collection is proportional to its size.

We evaluated four ways of creating profiles. The first way ("Random") is by selecting a number of random documents from the collection until the desired profile size is reached. These documents were then used to construct a relevance model. The relevance model defined the profiles which each document in the collection was matched against (as if it were being streamed from the network). The second type of profiles ("Selected") was obtained by selecting terms that occur in very few documents (less than ten in a million). This is most representative of real-world usage, and we hence focus on these types of profiles in our results. For our performance evaluation purpose, the main difference between these profiles is the hit probability, which was 10^{-5} for the "Random" profiles and 5.10^{-4} for the "Selected" profiles. For reference, we also compared the performance against an "empty" profile (one that results in no hits) and a "full" profile (where every term is a hit). While "empty" and "full" are unrealistic scenarios, they help in bracketing the benefits from our design.

Case	#Cycles/Term
Best Case	0.125
Worst Case	27
Full Bloom Filter Contention	1.2
External Access	18

Table II
FPGA CYCLE COUNTS FOR DIFFERENT CASES

C. FPGA Performance Results

FPGA baseline performance. The performance of the FPGA was measured using a cycle counter. The latency between starting the FPGA and the first term score is 22 cycles. For the subsequent terms, the delay depends on a number of factors. We considered four different cases:

- "Best Case": no contention on the Bloom filter access and no external memory access
- "Worst Case": contention on the Bloom filter access and external memory access for every term
- "Full Bloom Filter Contention": contention on the Bloom filter access for every term but no external memory access
- "External Access": no contention on the Bloom filter access, external memory access for every term

These cases were obtained by creating documents with contending/not contending term pairs and by setting all Bloom filter bits to 0 (no external access, which corresponds to an empty profile) or 1 (which correspond to a profile that would contain all terms in the vocabulary).

The results are shown in Table II. As we read 8 terms in parallel, the Best Case demonstrates that the FPGA works at I/O rates.

As explained earlier, the Bloom filter contention depends on the number of Bloom filter banks (16 in the current design). The probability for external access depends on the actual document collection and profile, but as the purpose of a document filter is to retrieve a small set of highly relevant documents, this probability is typically very small ($<$0.00001), as demonstrated by the experiments discussed in the next section. Consequently, the typical performance is determined by the cycle counts for Best Case and Bloom Filter Contention. At a clock speed of 150 MHz this results in a throughput of 772 million terms per second (772 MT/s) per FPGA.

Full workload results. Table III presents performance results for our FPGA implementation for various workload types. Focusing on a *selected* profile of 16K terms for 128K documents, our measured performance (shown in column 4) is 3090 million terms/second for the FPGA system (772 million terms/second *per FPGA*). Table III also shows the sensitivity to various other parameters. The performance of the FPGA design is comparable for different profile sizes and document sizes. However, as expected, the performance varies based on different hit probabilities for different profiles. In particular, there is a big drop-off in performance with the *full* profile, which is expected, as this profile has a hit probability of 1. Note that as discussed earlier, this is a very artificial case (since it requires the profile to be identical to the entire vocabulary of the collection), but helps provide bounds on our benefits.

D. Comparison with baseline

To compare the FPGA performance against a conventional CPU, we ran the experiments discussed in Section IV-A on an optimized multi-threaded reference implementation, written in C++, compiled with g++ with optimization -O3, and run on two different platforms: *System1* has an Intel Core 2 Duo Mobile E8435, 3.06 GHz and 8GB RAM, 1067 MHz bus; *System2* has an quad-core Intel Core i7-2600, 3.4 GHz, with 16GB RAM, 1333MHz bus. These higher memory baselines are required to enable sufficient memory for the algorithm. We keep the entire data set in memory because the memory I/O is much higher than the disk I/O. (We could of course run the algorithm several times on smaller data sets but then in that case the time required to read the data from disk would dominate the performance.)

While an in-memory approach might not be practical on a CPU-based system, on the FPGA-based system, this is entirely practical as the PROCStar-IV board has a memory capacity of 32GB. For example, the Novo-G FPGA supercomputer, which hosts 24 PROCStar-IV boards, can support a collection of 768GB. Note also that the format in which the documents are stored on the disk is a very efficient bag-of-words representation, which is much smaller than the actual textual representation of the document.

Performance. The results are summarized in Table III. For example, focusing on one example case, for the selected profile with 64K terms and 1M documents, compared to the 3090 million terms/second performance achieved by our design, *System2* system achieves 136 million terms/second and *System1* system achieves 82 million terms/sec. This translates to a 38× speedup for the FPGA-based design relative to *System1* system and a 23× speedup relative to *System2* system.

Additionally, examining the results for various workload configurations, the FPGA's performance is relatively constant across different workload inputs apart from the "Full" profile. This bears out the rationale for our design: because in general hits are rare, the FPGA works at the speed determined by I/O and Bloom Filter performance. The artificial "Full" profile results in all terms being a hit, and thus the performance for that case is determined by the external memory access. Unlike the FPGA-based design, the *System2* system sees more variation in performance with profile size (degraded performance with increased profile size) and document size (degraded performance with larger documents) and a bigger drop-off in performance between various profile types compared to the FPGA-based design.

Performance-per-Watt. We also measured the power consumption of our systems using the WattsUp Pro power meter. The measured power consumption for the maximum number of threads (Table IV) is 67W for *System1*, 141W for *System2*. In contrast, the FPGA-based design consumes 81W (35W of which is consumed by the host system). Clearly, the FPGA-based design achieves improved energy efficiency compared to the baselines. The FPGA-based design achieves energy-efficiency improvements of 31× and 40× for the *System1* and *System2* respectively. Though the FPGA speedup relative to *System1* was higher than that relative to *System2*, the overall energy-efficiency improvements are very similar. Our results illustrate the potential improvements in performance and energy efficiency relative to traditional baseline implementations.

Performance versus Cost. We use the cost model presented in [14], which computes the equivalent monthly cost of running a large data center.

The cost for space, power and cooling is calculated as

$$c_{SPC} = uc_S.nu_S + (1 + K_1 + L_1 + K_2L_1).uc_P.nu_P \quad (2)$$

with uc the unit costs (i.e. \$/sqft/month for space, \$/W/month for power/cooling) and nu the number of units (sqft resp. W). The factors K_1, K_2, L_1 represent power and cooling burdening and load, as explained in the paper. Typical values are \$1000/month for space per rack and \$0.072/W/month for power.

The cost IT_{dep} for the servers is calculated assuming \$100,000 per rack with a 3-year estimated lifetime. The monthly cost for software, licensing and personnel on a per-rack basis is estimated at \$10,000. We assume similar assumptions to Shah and Patel's work [14] for server costs (\$2500) and overheads and use the measured power from the previous sections for operational costs.

We model the cost of the FPGA system at \$8000. This is not based on current prices for high-end FPGA cards, because these prices tend to be very high because of the low volume involved. A single 10MW data centre would require about 50,000 cards, which is much more than the total amount of high-end FPGA cards sold today. Instead, we use the cost of an NVIDIA Tesla M2050 as a reference, this is typically \$2500. We believe this is representative as the Tesla board contains a GPU made with a similar process and of comparable gate count as the FPGA, and a similar amount of SDRAM. As the GiDEL PROCStar-IV board combines 4 FPGAs, it would be slightly cheaper than 4 separate boards. Our baseline costs comparisons hence use \$8000 for the FPGA system, but we also present results showing the sensitivity to this parameter. The license cost for the FPGA tools and the additional cost of developing the application are one-off cost that do not scale with the number of servers, and are therefore negligible for a large data center.

We calculate the increase in costs as a result of adding an FPGA card to each server, and based on these costs we calculate the performance/cost figure using the performance

Profile	System1	System2	FPGA board
Random, 4K	269	416	3090
Random, 16K	245	324	3090
Random, 64K	223	379	3090
Selected, 4K	118	232	3088
Selected, 16K	107	164	3088
Selected, 64K	82	136	3088
Empty, 4K	710	1564	3090
Empty, 16K	711	1664	3090
Empty, 64K	710	1338	3090
Full, 4K	8	11	36
Full, 16K	8	12	36
Full, 64K	9	10	36

(a)

Profile	System1	System2	FPGA board
Random, 4K	292	1118	3090
Random, 16K	288	1014	3090
Random, 64K	253	945	3090
Selected, 4K	120	309	3088
Selected, 16K	94	350	3088
Selected, 64K	72	183	3088
Empty, 4K	911	2005	3090
Empty, 16K	844	1976	3090
Empty, 64K	877	1952	3090
Full, 4K	7	10	36
Full, 16K	8	12	36
Full, 64K	8	11	36

(b)

Table III
THROUGHPUT OF DOCUMENT FILTERING APPLICATION (M TERMS/S) FOR (A) 256K DOCUMENTS OF 4096 TERMS AND (B) 1M DOCUMENTS OF 1024 TERMS

#Threads	System1	System2	FPGA System
0 (Idle)	40	67	35
1	67	93	61.5
2	67	107	68
4	67	135	74.5
8	67	141	81

Table IV
POWER CONSUMPTION OF DOCUMENT FILTERING APPLICATION (W) FOR 1M DOCUMENTS OF 1024 TERMS, PROFILE: SELECTED, 64K

Cost Breakdown	CPU	CPU+FPGA
Space	21M$/y	
Power & Cooling	52M$/y	29M$/y
IT Infrastructure	59M$/y	248M$/y
Total	132M$/y	299M$/y
Performance (single system)	136Mops/s	3090Mops/s
Performance/Cost	**32Mops/$**	**330Mops/$**

Table V
PERFORMANCE VERSUS COST

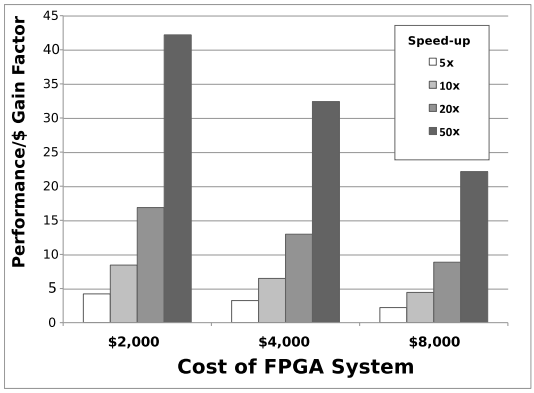

Figure 5. Performance/Cost versus FPGA system cost and performance gains

results from the previous section (averaged over all "Random" and "Selected" runs from Table III). The results are shown in Table V .

As we can see from the table, the main increase in cost is due to our high estimate for the price of the FPGA cards: the IT infrastructure cost is about 4× higher because of this. The cost for power and cooling decreases with almost a factor of 2 (from 52M$/y to 29M$/y); space costs are not affected. As the FPGA card results in a large increase in performance, the final Performance/Cost figures show that the FPGA-accelerated solution is almost ten times more cost efficient than the traditional one, for the typical case.

To explore the effect of the FPGA system's cost and performance, we varied both parameters as shown in Figure 5. The graph shows the speed-up required for a given FPGA system cost to achieve a given factor improvement in performance/$. The most important points are that even in the worst case the FPGA-based system still outperform the CPU-only solution, and that the potential for cost reduction is very great.

E. Discussion

Improving benefits from FPGA implementation. While our results clearly illustrate the potential benefits from FPGA-based acceleration, they can potentially be improved further. The current design uses a 16-bank Bloom filter, which is not optimal for scoring 8 parallel terms. Extending the design to 64 banks would increase the throughput by almost 40% (from 772MT/s to 1053MT/s). Furthermore, the current Bloom filter combines 4 terms per bit. We can double the Bloom filter size (i.e., 8Mb), leading to 2 terms per bit, which will reduce the rate of false positives accordingly. For the given application of scoring a known collection of documents, we could also reorder the terms in each document to reduce contention. Combined, these improvements can potentially result in a throughput very close to the I/O limit of 1200MT/s.

Comparison of FPGA versus other alternatives.

ASIC Bloom Filter: As mentioned earlier, the main performance improvement from our approach over a general-purpose CPU is that we can use bit-accessible Bloom filters to discard

negatives. If we could create an ASIC for this purpose it could potentially have an order-of-magnitude better performance. It is important to note that the FPGA runs at a very low clock speed (150MHz), which is the main reason why it is a low power technology. Consequently, the FPGA implementation can only win by having more parallelism or by having a better memory architecture. The Bloom filter is just that: a better memory-optimized architecture compared to the CPU cache. In our particular implementation, we have 8x parallelism (per FPGA) for accessing the document collection, and 16x for accessing the Bloom filter. The former is dictated by the IO width and DRAM clock speed; the latter can still be improved as explained above.

*GPGPU Implementation:*The same observations as for the CPU also apply to GPGPUs, with different parameters: the GPU clock speed is usually only about 2× lower than the CPU (instead of 20× for the FPGA). Memory I/O is comparable. Again, the crucial difference, and the reason why the FPGA can still outperform the GPU for this application, is in the memory architecture. The GPGPU has a scratch pad per streaming multiprocessor. However, the size of this memory is typically 16KB, up to 128KB for high-end GPUs. This is much too small to store a Bloom filter of the size that we require (4Mb). So we can't implement a high-bandwidth Bloom filter on the GPU local memory to discard negatives. Furthermore, although the GPU has a large number of data parallel threads inside each multiprocessor, they contend for the global memory access, so this becomes the bottleneck. That means that there is little benefit in the large number of parallel cores provided by the GPU for this application.

Future work. Considering the wide potential application domain of information filtering, and the need for power- and cost-effective system architectures, our future work will explore a number of different avenues. An optimized FPGA implementation as explained in Section IV-E is one of them. We also want to perform a more in-depth characterization of other diverse workloads. Based on the learnings from this work, we also want to explore different system architectures with a low-power, relatively low-performance CPU host (e.g. ARM-based), as the host power consumption in idle mode is currently the dominant factor.

Important concerns which hamper the adoption of FPGAs are the programming complexity and the lack of standardized APIs. The former can be addressed by the use of high-level languages such as the C-based Impulse-C, Catapult-C and others [15], [16] or the MORA framework [17], which offers a C++ API for high-level FPGA application programming. To address the latter, it is important to put forward a standard. OpenCL [18] is the emerging industry standard for programming of multicore and manycore devices, in particular GPUs. It provides a flexible API for host-device communication and a C-like language for device kernel programming. Adding FPGAs to the set of platforms supported by OpenCL is therefore very attractive. We are currently working on extending the MORA framework for an OpenCL-compliant implementation and plan to port our current design using this framework.

V. Related Work

The present work improves on our previous work [10] where we implemented a Document Filtering algorithm on an older-generation Xilinx Virtex IV FPGA as part of a relevancy feedback-based search application for patents. Although the previous project addresses the application of FPGA for Information Retrieval and showed good speed-ups over the software reference implementation, it was not optimized to leverage the advantages of higher Input/Output (I/O) data bandwidth and parallelism in the FPGA. As explained above, the key contribution of the current work is the Bloom Filter design. Our original Document Filtering implementation uses a single large Bloom Filter that acts as a lookup table for a tuple. We modified the Bloom Filter module to be mapped on multiple banks, which dramatically reduces the contention. As a result, the current implementation outperforms our original implementation by more than a factor of 10. (Note also that compared to a Lemur-based baseline implementation in the prior work, our study uses an optimized multi-threaded baseline as well.)

Apart from our work on FPGAs for Information Retrieval Algorithms, some corporate and academic research systems have exploited FPGAs to improve the performance of Information Retrieval. Netezza [19] is a data warehouse appliance designed for rapid analysis of large data volumes, with the potential for performance improvements at a much lower cost compared to traditional database designs. The FPGAs are used to filter content as fast as it streams off the disk, so that irrelevant data does not need to be processed by the CPU. The differences with our current work are two-fold: first, disk I/O is orders of magnitude slower than memory I/O; second, Netezza filters textual data while we filter a bag-of-words representation. Overall, our application performs at a much higher I/O rate (800MB/s per FPGA, 3.2GB/s for the board) than required by Netezza. Other FPGA systems have also been studied for Information Retrieval, specifically for mapping Language Classification algorithms using N-grams [20], [21]. Lockwood [20] implemented a Language Classification algorithm to differentiate 255 languages. Since the embedded memory available on FPGAs is limited, they mapped the Bloom Filter on external memory resulting in lower throughputs due to the random memory access. Jacob [21] extended this work to map the Bloom Filter on embedded memory on FPGA. However, due to the constrained availability of Block Memory on FPGA, only a smaller Bloom Filter for each language can be mapped, which results in higher false positive rates, correspondingly affecting the accuracy of their algorithm and increasing the misidentification of languages.

A number of researchers (e.g. van Lunteren [22]) have implemented a regular expresssion matching system on FPGA. While similar in that regular expression matching can be considered a specific type of search, our work differs in not being concerned with the actual matching of terms, but with attributing a score to a document based on the importance of the terms. Thus, our work can potentially complement such studies. Specifically, every regular expression could constitute an entry in our "profile". In this way our algorithm will

perform a relevancy judgement based on the matches, rather than performing the actual term matching.

Ding et al. [23] have investigated the use of GPUs for high-performance Information Retrieval. In principle, GPUs are very interesting target platforms for IR because many IR problems are parallelisable. In this work, the authors report on the design of a basic system architecture for GPU-based high-performance query processing. While thematically similar to this work, query-based search is actually the dual of document filtering: as explained in [24], in query-based search, the query is dynamic (and typically small) and the collection is static, and abstracted into an "inverted index" datastructure. In filtering, the query is static (and typically large) and the collection is dynamic. The work in [23] is a framework for query-based search; while a direct comparison to our work is not possible, it is worth noting that the , so a direct comparison with our work is not possible. However, the obtained speed-ups for query processing are moderate (factor 2-3), and in addition, the GPU does not have the benefit of low power consumption offered by the FPGA.

VI. Conclusions

Recent trends on "big data" and data-centric computing in combination with a growing emphasis on "green computing" motivate new system designs for high performance, and at improved energy efficiency. In this paper, we present a new FPGA-accelerated system design for an important class of data-centric workloads, namely, information retrieval or unstructured search. We implemented our design on the GiDEL ProcStar IV board using Altera Stratix IV 530 FPGAs, and our results show significant performance (nearly 800 million terms per second per FPGA) at low power (6W). Compared to baseline server systems running the same algorithm, our FPGA-based design achieves performance speedup of 23× to 38×, and energy efficiency improvements of 31× to 40×, for typical workload inputs. These results demonstrate that the usage of FPGAs as "greener hardware" can deliver tremendous benefits by both reducing the power consumed, and also increasing the speed of execution.

Future work will be directed towards: (i) scaling the prototype up to data center scale, and (ii) implementing more sophisticated filtering algorithms, along with other document filtering tasks such as classification/clustering (e.g. spam filtering) where efficiency is also paramount. Our analysis of the benefits of acceleration also illustrates interesting insights that we believe can be ported to more general-purpose designs, and we plan to explore these further as well.

VII. Acknowledgements

We would like to thank the anonymous reviewers, Trey Cain, Jichuan Chang, and Kevin Lim for their useful feedback. We would also like to thank Kenneth Jansen at HP, Syam Uma Chander, Baoz Agur at GiDEL and Mike Strickland at Altera for their support. This research was partly funded by a HP grant.

References

[1] J. Short, R. Bohn, and C. Baru, "How Much Information? 2010 Report on Enterprise Server Information," http://hmi.ucsd.edu/pdf/HMI_2010_EnterpriseReport_Jan_2011.pdf, University of California, San Diego, 2011.

[2] L. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[3] "Hybrid-core: The big data computing architecture," http://www.conveycomputer.com/sc11/DIS-G500.Convey.Final.pdf, Convery Computers, 2011.

[4] J. Coyne, J. Allred, V. Natoli, and W. Lynch, "A field programmable gate array co-processor for the basic local alignment search tool," http://www.stoneridgetechnology.com/uploads/toolshed/blast_fpl09.pdf, Stone Ridge Technology, 2009.

[5] N. A. Wood, "Fpga acceleration of european options pricing," http://www.xtremedata.com/images/pdf/MonteCarloXtremeRNGWhitePaper_April2008v1_0.pdf, XtremeData, 2008.

[6] V. Lavrenko and W. B. Croft, "Relevance based language models," in *Proc. of the 24th ACM SIGIR Conference*, 2001, pp. 120–127.

[7] "Google n-gram project," http://ngrams.googlelabs.com/ngrams/info, Google, 2010.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Com. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[9] "Stratix iv handbook," http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf, Altera.

[10] W. Vanderbauwhede, L. Azzopardi, and M. Moadeli, "Fpga-accelerated information retrieval: High-efficiency document filtering," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 417–422.

[11] R. Losee, "Term dependence: A basis for luhn and zipf models," *Journal of the American Society for Information Science and Technology*, vol. 52, no. 12, pp. 1019–1025, 2001.

[12] M. Montemurro, "Beyond the zipf-mandelbrot law in quantitative linguistics," *Physica A: Statistical Mechanics and its Applications*, vol. 300, no. 3-4, pp. 567–578, 2001.

[13] W. Press, *Numerical recipes: the art of scientific computing*. Cambridge Univ Pr, 2007.

[14] C. Patel and A. Shah, "Cost model for planning, development and operation of a data center," *Hewlett-Packard Laboratories Technical Report*, 2005.

[15] J. Xu, N. Subramanian, A. Alessio, and S. Hauck, "Impulse c vs. vhdl for accelerating tomographic reconstruction," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2010, pp. 171–174.

[16] B. Holland, M. Vacas, V. Aggarwal, R. DeVille, I. Troxel, and A. George, "Survey of c-based application mapping tools for reconfigurable computing," in *Proceedings of the 8th International Conference on Military and Aerospace Programmable Logic Devices (MAPLD05)*.

[17] W. Vanderbauwhede, M. Margala, S. R. Chalamalasetti, and S. Purohit, "A c++-embedded domain-specific language for programming the mora soft processor array," in *Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on*, July 2010, pp. 141 –148.

[18] J. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science Engineering*, vol. 12, no. 3, pp. 66 –73, May-June 2010.

[19] "Netezza appliance architecture," http://www.Netezza.com, IBM, 2011.

[20] S. Eick, J. Lockwood, R. Loui, A. Levine, J. Mauger, D. Weishar, A. Ratner, and J. Byrnes, "Hardware accelerated algorithms for semantic processing of document streams," in *Aerospace Conference, 2006 IEEE*, 2006, p. 14.

[21] A. Jacob and M. Gokhale, "Language classification using n-grams accelerated by fpga-based bloom filters," in *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07*. ACM, 2007, pp. 31–37.

[22] J. van Lunteren, "High-performance pattern-matching for intrusion detection," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, april 2006, pp. 1 –13.

[23] S. Ding, J. He, H. Yan, and T. Suel, "Using graphics processors for high performance ir query processing," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 421–430.

[24] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Edinburgh Gate: Pearson Education Limited., 1999.