Document Classification Systems in Heterogeneous Computing Environments

Nasibeh Nasiri, Philip Colangelo, Oren Segal, Martin Margala
Electrical and Computer Engineering Department
University of Massachusetts Lowell
Lowell, USA
Nasibeh_Nasiri@uml.edu; Philip_Colangelo@student.uml.edu; Oren_Segal@student.uml.edu; Martin_Margala@uml.edu

Wim Vanderbauwhede
School of Computing Science
University of Glasgow
Glasgow, UK
Wim.Vanderbauwhede@glasgow.ac.uk

***Abstract*—**Datacenter workloads demand high throughput, low cost and power efficient solutions. In most data centers the operating costs dominates the infrastructure cost. The ever growing amounts of data and the critical need for higher throughput, more energy efficient document classification solutions motivated us to investigate alternatives to the traditional homogeneous CPU based implementations of document classification systems. Several heterogeneous systems were investigated in the past where CPUs were combined with GPUs and FPGAs as system accelerators. The increasing complexity of FPGAs made them an interesting device in the heterogeneous computing environments and on the other hand difficult to program using Hardware Description languages. We explore the trade-offs when using high level synthesis and low level synthesis when programming FPGAs. Using low level synthesis results in less hardware resource usage on FPGAs and also offers the higher throughput compared to using HLS tool. While using HLS tool different heterogeneous computing devices such as multicore CPU and GPU targeted. Through our implementation experience and empirical results for data centric applications, we conclude that we can achieve power efficient results for these set of applications by either using low level synthesis or high level synthesis for programming FPGAs.

I. Introduction

The explosive growth of data is evident in the increase in network usage over the internet in the past decade or so. Different types of data such as video, image, html and emails all need to be searched, classified and filtered. The need for efficient methods to sift through the massive amounts of information that is generated on a daily basis motivates us to search for better implementations in terms of speed, energy and cost.

Heterogeneous systems have the potential to be more power efficient than their homogenous counterparts [1] because of the ability to use specialized hardware to tackle different types of algorithms. FPGAs, GPUs, and CPUs are three of the major off-the-shelf computing platforms which should be considered in heterogeneous system design, each with its own set of merits.

CPUs are general purpose processing units that have evolved to be relatively efficient at high level generated code which translates to instructions that can be branched, with several levels of cache hierarchy that can assist in extracting locality from code and data. GPUs are known for their ability as massive number crunchers [2][3] using large SIMD vectorized units to parallelize similar code that runs on large amounts of data. FPGAs are reprogrammable integrated circuits which can be tailored and customized for a specific application. There are many programmable building blocks in the state of the art FPGAs including ALMs, variable precision DSPs, on-chip memory blocks, high speed transceivers, etc. In addition, FPGAs have low core clock frequency resulting in low power architectures and high memory bandwidth, which makes them a logical choice for data-centric applications.

One of the main issues delaying wide spread adoption of FPGAs in data-centric applications is the difficulty in programming them. FPGAs are programmed with Hardware Description Languages (HDLs) which require good knowledge of the underlying hardware and a significant engineering effort, when compared to high level software programming. Algorithm design in HDLs requires the implementation of all the communications and service layers that are provided with no extra effort to a high level programmer through operating system services and drivers. For FPGAs to be considered as standard heterogeneous system components they need to be accessible as normal devices in an operating system environment. In addition, maintaining a single code base across different devices and platforms serves an important software principle of code reuse.

OpenCL [7] is an open standard framework for parallel programming of heterogeneous computing platforms in which data and task-based parallel programming models are supported. The introduction of OpenCL for FPGAs opens new possibilities for the usage of FPGAs, as standard system accelerators.

In this work, the low level synthesis results on DE5-NET FPGA board are collected for two data centric applications. In addition, these two classification systems are implemented in OpenCL on a DE5-NET FPGA board, K40C GPU and a HP DL180 G6 with dual Intel Xeon L5630 2.13GHz processors. The implementations of these classification systems are compared in terms of power consumption and performance while running on synthetic data sets.

There are four main contributions to this paper. The first contribution of this paper is the development of a novel integrated parser and scorer for two document classification systems [8]. Using the bag of words, meaning processing the documents on the host side, in the previous publications has been the performance bottleneck. The second contribution of this paper is the programming the DE5-Net FPGA board using both low level synthesis and high level synthesis. The third contribution of this work is the development and investigation of two cross platform high performance document classifications OpenCL code base for CPUs, GPUs and FPGAs with minimal code variations. The fourth one is the comparison study of two different document classification systems and evaluating the efficiency of each of these implemented systems.

The remainder of this work is organized as follows: In section II and III we discuss related work and HLS versus low level synthesis for FPGAs. The document classification system is explained in details in section IV. V. The HDL implementation of document classifications are elaborated in section V. Profile memory and the provided data to this memory are explained in section VI. We present and analyze the results obtained from the Dual Xeon processor, K40C GPU and FPGA implementations in section VII and finally in section VIII we draw conclusions.

II. Related Work

Previous research on FPGA-CPU-GPU implementations of document classification or filtering has shown that FPGAs offer power efficient implementations compared to the CPU and GPU implementations. The following studies investigated the feasibility of document classification. In [4], the authors programmed an FPGA using Mitrion-C but the bag of words is used and was the bottleneck for the further parallelization. In [5], the authors also developed a multi-FPGA converted bag of words system in which performance of the system yielded a tenfold speedup when combined with a conventional CPU and excellent cost-performance as well as energy consumed for the calculation. Similarly in [6], the authors assume that documents have been converted to the bag-of-words format and focus only on the document scoring portion of the application.

In this paper, we integrated the scoring portion with a parser and scored them document by document against different profiles. We have investigated two different implementations. The first implementation uses a classification system based on a Naïve Bayesian Classifier and a finite state machine based parsing method to classify HTML documents. The second one uses the same classification method but with a sliding window based parsing method.

Handwritten Verilog implementations of these applications are used to program the FPGA and compared to the results of high level synthesis. OpenCL is used to parallelize not only the scoring part but also the parser part of document classification systems and targeted three platforms (FPGA, GPU and multi-core CPU) with the same OpenCL code for each alternative classification system with only minor optimizations and changes. Thus the need for the pains taking development of a separate FPGA model in low level RTL is removed when using OpenCL implementations.

III. High Level Synthesis vs Low Level Stnthesis

Traditional FPGA design involves describing state machines, data paths, arbitration, interfaces to external memory, buffering etc. using HDLs. A knowledgeable engineer would describe the system specification down to Register Transfer Level. The FPGA programmer takes care of synthesis, place and route; timing closures etc. to complete the design flow. Verification of design at various steps of design flow to remove the discrepancies of design and specification is necessary. These steps need significant engineering effort and good knowledge of hardware to utilize the available hardware resources of the target FPGA.

On the other hand in high level programming of FPGAs, the compiler takes care of all these steps and constraints. Consequently the programmer can concentrate on the optimization of the design without getting involved in the RTL details of the design.

The increasing complexity of FPGAs made them an attractive device in the heterogeneous computing environments but programming these devices has become more challenging using traditional approaches. For nowadays application demand, the need for HLS is inevitable. HLS is a new trend in programming FPGAs and evolving significantly. HLS improves the design productivity by automating the refinement from the algorithm level to Register Transfer Level. When HLS is used to describe a system, the number of code lines is usually reduced which resulting in less mistakes and making the debugging the code faster. The verification time sometimes exceeds the design time; although HLS generates some testbenches to automatically verify the generated HDL design.

There are some metrics in order to pick an HLS such as learning curve, amount of effort to write source code, designer experience, documentation and tool capabilities. There have been different HLS tools with different trade-offs. Since in this study, the aim is to target different platforms with single code, the Altera OpenCL HSL [8] was chosen. OpenCL provides a single design environment for heterogeneous systems. Altera OpenCL translates the high level code into a pipelined hardware circuit in where each stage of the pipeline executes a different thread. In OpenCL a C based kernel is replicated and run in parallel on multiple hardware compute units. Each parallel run is assigned an ID which allows the kernel work on a subset of the data that is associated with it.

IV. DOCUMENT CLASSIFICATION SYTEMS

A document classification application classifies a stream of HTML (or email) documents in “relevant” or “not relevant”, where relevance is determined by a profile of keywords provided by the user or trained using some set of data. There are three stages for classification of documents against different profiles:

1. *Pre-processing* for training
2. *Training*
3. *Classification* of HTML documents.

In the *pre-processing* stage, all HTML tags are stripped off, all stop words, i.e. words that appear frequently but have low content discriminating power, are removed from each document. During the *training* stage, a model is built based on the characteristics of each category in a pre-classified set of documents. *Classification* can be done using Naïve Bayesian classifier [10]. The Naive Bayesian classifier is based on the Bayesian theorem which assumes attributes have independent distributions. A Naive Bayesian model is fast and space efficient to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods. Two different implementations of document classification are implemented in this work in a heterogeneous computing environment and compared in terms of throughput and performance per watt in the results section. The detailed implementation and steps for each of these implementations are elaborated in this section.

In the first implementation, the stream of HTML documents is parsed by going through a finite state machine, compressed and stored on a document-by-document basis. Bigrams and trigrams (group of two and three adjacent words) are created to improve the accuracy of the document classification. For example "cheap" and "Rolex" don't exist in the profile but "cheap Rolex" a bigram does exist in the profile. In order to accelerate the look-up of the terms (i.e. unigrams, bigrams and trigrams), we use a Bloom Filter to reject terms not present in the profile. The Bloom Filter, stored in the on-chip memory, determines the membership of a term in the profile however some false positives are possible. If the Bloom Filter returns a hit, the term will be looked up in the profile memory residing in (off-chip) SDRAM. The hit rate of the Bloom Filter is proportional to the size of profile.

The flow of the implemented design follows below:

1. The input stream will go through the parser and once a word is detected and encoded using compression coding, it will be sent for scoring.

2. For higher precision filtering purpose the bigram and trigram of each word is generated and were looked for in the profile.

3. N hashing function has been used to evaluate the membership of each unigram, bigram or trigram so at most N×3 accesses are needed to the Bloom Filter(s).

4. If the retuned values of N hashing functions are all one for each unigram, bigram or trigram, this can be a sign of existence of these word(s) in the profile so the next step is looking up in the profile however false positives are possible.

5. When it is required to refer to the profile, the rest bits of the hit word are compared with the rest bits of the profile then if they are matched, the weight bits of this element is accumulated to the score.

6. The final result of classification will be made using naïve Bayesian classifier.

The second classification approach parses the HTMLs differently and skips generating the bigrams and trigrams [11]. The flow of the implemented design follows below:

1. The input stream will go through a window and the output of window is encoded using compression coding, it will be sent for scoring.

2. N hashing function has been used to evaluate the membership of the outputs of parser.

3. If the retuned values of N hashing functions are all one, the next step is looking up in the profile.

4. When it is required to refer to the profile, the rest bits of the hit word are compared with the rest bits of the profile, then if they are matched, the weight bits of this element is accumulated to the score.

5. The final result of the classification will be made using a Naïve Bayesian classifier.

In previous implementations we had to use background processing of data i.e. parsing on CPU effectively decreasing overall performance of the system. The current algorithm implementations allow us to feed real time data e.g. HTML or network traffic into the accelerator. In this paper, for the purpose of testing the system, we used predefined data sets as explained in section VI.

V. HDL Implementation of Document Classification

The HTML documents are stored in one of the SDRAM DDR3 memories. The second DDR3 memory is reloaded in every experiment with the target profile data. The HDL implementation of parsing and scoring are deeply pipelined. In order to remove the performance bottleneck of slow PCIe data transfer rate to the FPGA, PCIe is interfaced to DDR3 in order to transfer the data back and forth to DDR3 memory. The HTML documents are transferred to the first DDR3 and also profile data set to the second DDR3. Final score for each document is written to the second DDR3 memory as well.

Two different parsers are implemented in this work. The first parser works based on a finite state machine. The goal is to discard the HTML tags and parse the document and generate bigram (two adjacent words) and trigram (three adjacent words) for each word and send it to be scored. Each character is compressed to a 5-bit code, which results in a more efficient design. A sliding window based parser [11] is replaced by the finite state machine based parser in the second implementation.

Some Bloom Filters, highly scalable data structures, are used for membership testing and speeding up the look ups from the profile which is residing in the external memory for both implementations. The bloom filters avoid most of the unnecessary accesses to the external memory; however some false positives are possible. The Bloom Filters are implemented using distributed Block RAMs on the FPGA.

A Bloom Filter can be implemented and replicated with limited number of hash functions. We have used two hash functions in our experiments for these implementations.

Two hashing functions are used for each unigram, bigram and trigram in order to look them up in the Bloom Filter

memories. If both accesses of any word, bigram or trigram are hit then the appropriate address will be generated to be looked up in the external memory.

VI. Data Set and Profile

Ideally the performance of the implemented application should be evaluated using real world input data but since our access to the real data sets was limited, we relied on synthetic document collections that are statistically similar to the real-world collections. To generate synthetic data sets, we used summary information from several document collections (TREC Aquaint). These collections provide good coverage on the impact of different document lengths and sizes on filtering time. TREC Aquaint has 1,033,461 documents in which with the average document length of 437 words and the distribution of unique terms is about 169.

As explained in section V, the profile which is stored on off-chip memory can be loaded in every experiment with different weights obtained from different algorithms for each of the existing feature in the profile. In our experiments we have used the naïve Bayesian Classification algorithm in order to classify the documents against a profile data set. Profile files with different content types implied by their names: Entertainment Financial, Entertainment International, Entertainment Political, Entertainment Washington, Sports Financial, Sports International, Sports Political, Sport Washington, USA Financial, USA International, USA Political and USA Washington have been used to test the implementation in different platforms.

VII. EXPERIMENTS AND RESULTS

The following devices were used to test execution and power efficiency of our applications:

1) HP DL180 G6 with dual Intel Xeon L5630 2.13GHz processors and 144GB DDR3, 1333MHz RAM

2) GPU – NVIDIA K40C

4) FPGA – DE5-NET board, Altera Stratix V

The host system was equipped with 8GB RAM connected through a PCIe connection to the host system. The system was running Linux Centos 6.4. Total system power is measured using a watts-up pro power meter connected to the power outlet of the system. Specialized scripts monitor the life time of the benchmarked processes and the average system power consumption during their execution.

Kernel execution time and performance per watt of the document classification applications are shown in Table 1 for two different implementations. Whole system power measurements were obtained using a watts-up pro power meter. Average power consumption while running in CPU mode is 181W to 211W, in GPU mode is 91W to 100W and in FPGA mode is 65W to 90W. The results show that the power efficiency of the fastest FPGA version is slightly higher than the fastest CPU version.

The results of the document classifications using Bayesian classification show that the highest throughput was obtained while running on FPGA with Bloom Filter and finite state machine based parsing method. The GPU is not fully utilized in our experiments since the experiments with the same data size are repeated on GPU.

Table 1. Kernel execution time and performance per watt on FPGA, CPU, GPU

	Device	Kernel Execution Time (s)	Performance per watt(MB/Watts)
No Bloom Filter Bayesian - FSM	FPGA (Verilog)	0.46	8.36
	FPGA (HLS)	0.7	4.51
	CPU	0.81	1.47
	GPU	0.57	4.32
Bloom Filter Bayesian-FSM	FPGA (Verilog)	0.33	11.66
	FPGA (HLS)	0.55	4.92
	CPU	0.73	1.64
	GPU	0.57	4.59
No Bloom Filter-Bayesian-Window	FPGA (Verilog)	1.7	2.26
	FPGA (HLS)	3.62	0.86
	CPU	4.03	0.32
	GPU	3.33	0.75
Bloom Filter-Bayesian-Window	FPGA (Verilog)	0.62	6.20
	FPGA (HLS)	3.12	1.14
	CPU	3.38	0.38
	GPU	3.21	0.82

As we see in Table 1, the implementation using the Bayesian Classifier with Bloom Filter has the highest performance and performance per watt compared to the other variation of implementations.

The hardware utilization of low level synthesis is reported in Table 2. The low level synthesis has its own merits in terms of low hardware resource usage on FPGA and the fastest implementation. The low level synthesis results show that the same application runs from 1.5 to 5 times faster compared to those of high level synthesis implementations.

The hardware utilization of the FPGA platform using high level synthesis is reported in Table 3. The results were obtained by combining the auto resource-driven optimizer (O3 compile option- default value of 85% utilization) and manual optimizations on each kernel in an effort to reach the best possible performance. Note that even though memory blocks usage looks similar on both Bloom and non-Bloom versions, the Bloom version uses *Local_mem_resources* and *Local_mem_ram_resources* while the non-bloom does not use any. The only manual optimizations and code changes that were used in the FPGA version were the *restrict* keyword and pragma lines used for things such as loop unrolling and workgroup size settings. To obtain the best performing Bloom kernel version we ran our program with different workgroup

sizes starting from 16 to 2048 (only powers of two). The maximum speed ups for the FPGA Bloom Filter-Bayesian-FSM and Bloom Filter-Bayesian-Window implementation were achieved when the workgroup size was set to 1024 this is due to the fact that initializing the local memory for the Bloom Filter is done once in the kernel, on a per workgroup basis by copying the Bloom Filter from global memory. When work group sizes are small, the overhead of local memory initializations and memory copies offsets the advantages of using local memory. On a CPU or GPU setting a workgroup size is limited and tied to the specific hardware device implementation. On the other hand, on the FPGA kernel version, we added a single pragma line (*max_work_group_size*) which instructs the compiler to generate hardware that will be able to efficiently handle bigger workgroup sizes, which led to significant speed gains. This serves as an example of the advantage of using an FPGA for algorithm design combined with the ease of OpenCL development.

Table 2. Logic utilization of document classification on DE5-NET FPGA board using low level synthesis

	Logic utilization	**Dedicated logic reg.**	**Memory blocks**	**DSP blocks**
No Bloom Filter Bayesian -FSM	11%	10%	2%	0
Bloom Filter Bayesian-FSM	12%	11%	3%	0
No Bloom Filter-Bayesian-Window	15%	10%	2%	0
Bloom Filter-Bayesian-Window	16%	12%	4%	0

Table 3. Logic utilization of document classification on DE5-NET FPGA board using high level synthesis

	Logic utilization	**Dedicated logic reg.**	**Memory blocks**	**DSP blocks**
No Bloom Filter Bayesian -FSM	86%	39%	78%	13%
Bloom Filter Bayesian-FSM	68%	38%	72%	8%
No Bloom Filter-Bayesian-Window	75%	43%	65%	7%
Bloom Filter-Bayesian-Window	63%	41%	55%	4%

VIII. CONCLUSION

Our experiments show that FPGAs are a promising platform for classification of documents. They not only provide high throughput, but also high performance/watt when compared to GPUs and CPUs. High level development of FPGA designs is making FPGAs an attractive platform for data-centric applications even though it is not as efficient as handwritten Verilog implementation. The most perplexing fact is that an FPGA running at a clock frequency that is an order of magnitude lower than CPUs and GPUs is able to outperform them. When it comes to power efficiency (performance per watt), however, both CPU and GPU lag behind the FPGA. Our future work will be constructing profiles dynamically according to the newly coming input. We also plan on investigating several machine learning algorithms to construct the profile and update it automatically.

ACKNOWLEDGMENTS

We would like to thank Altera, Terrasic and NVidia for their generous hardware and software donations.

REFERENCES

[1] Chung, Eric S., et al. "*Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? ,*" Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2010.

[2] Reyes, R., Lopez, I, Fumero, J.J., and De Sande, F., "*Directive-based Programming for GPUs: A Comparative Study,*" High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), on pp.410,417, 25-27 June 2012.

[3] Huang, S., Xiao, S., and Feng, W., "*On the energy efficiency of graphics processing units for scientific computing,"* Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on , pp.1,8, 23-29 May 2009

[4] Vanderbauwhede, W., Azzopardi, L., and Moadeli, M., "*FPGA-accelerated Information Retrieval: High-efficiency document filtering*," Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on, pp.417-422, Sept 2009.

[5] Vanderbauwhede, W., Frolov, A., Rahul Chalamalasetti, S., and Margala,M., "*A Hybrid CPU-FPGA System for High Throughput (10Gb/s) Streaming Document Classification*," in Proceedings of 4th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART), June 2013.

[6] Chen, D., and Singh, D., "*Invited paper: Using OpenCL to evaluate the efficiency of CPUS, GPUS and FPGAS for information filtering,*" Field Programmable Logic and Applications (FPL), 22nd International Conference on, pp.5,12, August 2012.

[7] Khronos OpenCL Working Group. "*OpenCL-The open standard for parallel programming of heterogeneous systems [Online]. DOI= http://www. khronos.org/opencl,*" 2011.

[8] Nasibeh Nasiri, Oren Segal, Martin Margala, Wim Vanderbauwhede, Sai Rahul Chalamalasetti, "*High Level Programming of Document Classification Systems for Heterogeneous Environments using OpenCL*," ACM International Symposium on Field-Programmable Gate Array, 22 – 24 February 2015

[9] *Altera* SDK for OpenCL. http://www.altera.com/literature/lit-opencl-sdk.js

[10] Sahami, M. "*A Bayesian approach to filtering junk e-mail,*" Proceedings of AAAI-98 Workshop on Learning from Text Categorization. 1998

[11] Gordon V. Cormack, Mark D. Smucker, Charles L. A. Clarke, "*Efficient and Effective Spam Filtering and Re-ranking for Large Web Datasets,*" Information retrieval 14 (5), 441-465, 2011