J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A hardware compilation framework for text analytics queries

Raphael Polig [a,*], Kubilay Atasu [a], Heiner Giefers [a], Christoph Hagleitner [a], Laura Chiticariu [b], Frederick Reiss [b], Huaiyu Zhu [b], Peter Hofstee [c]

[a] IBM Research — Zurich, Saumerstr.4, Rueschlikon, Switzerland
[b] IBM Research — Almaden, 650 Harry Road, 95120-6099 San Jose, CA, USA
[c] IBM Research — Austin, 11501 Burnet Road, 78758-3400 Austin, TX, USA

HIGHLIGHTS

- A complete hardware compilation framework is presented that transforms text analytics queries into a synthesizable hardware description.
- The accelerator architecture is coherently integrated with a general purpose processor demonstrating an up to 60 times faster processing rate.
- The performance of two generations of systems is evaluated demonstrating the improvements gained by advances in technology.

ARTICLE INFO

Article history:
Received 12 May 2016
Received in revised form 19 May 2017
Accepted 24 May 2017
Available online xxxx

Keywords:
Text analytics
FPGA
Query compilation
Accelerator

ABSTRACT

Unstructured text data is being generated at an unprecedented rate in the form of Twitter feeds, machine logs or medical records. The analysis of this data is an important step to gaining significant insight regarding innovation, security and decision-making. The performance of traditional compute systems struggles to keep up with the rapid data growth and the expected high quality of information extraction. To cope with this situation, a compilation framework is presented that can transform text analytics queries into a hardware description. Deployed on an FPGA, the queries can be executed 60 times faster on average compared to a multi-threaded software implementation. The performance has been evaluated on two generations of high-end server systems including two generations of FPGAs, demonstrating the performance gains from advanced technology.

1. Introduction

Digital data is being generated in every aspect of our daily lives [13]. We create more than 2.5 billion gigabytes of data per day [9], from sensor data to online shopping transactions. A great part of this data is represented by un-structured and semi-structured text documents. Such as the 500 million Twitter messages per day [11], uncountable number of daily news entries around the world or fewer but more complex scientific research papers. All of this data may or may not contain valuable information for different groups of people such as medical scientists or marketing experts [19]. Extracting the specific information from this text-based data is the task of text analytics.

Although many improvements have been applied to the underlying frameworks and algorithms, text analytics continues to be computationally very intensive; also, because the level of detail at which data is analyzed continues in order to increase to improve the quality of the results achieved. Furthermore, these frameworks only benefit a little from new features in modern microprocessors such as wide single-instruction multiple-data units [8]. This leads to a performance gap between the ever faster growing amounts of data and the moderate performance enhancements of new processor generations.

Text analytics can be easily parallelized as extraction queries are run for each text document individually. Cloud computing offers a way to cope with the lack of single node performance by massively parallelizing the task on a large compute cluster. Thus, each thread in a cluster can operate independently from each other and will be utilized because the amount of data to be analyzed can be scaled. But, this comes with a drop of efficiency due to the increased management and communication overheads. To counter this problem, system designers turn to heterogeneous platforms which include special purpose processors such as graphics processing units (GPUs), digital signal processors (DSPs) or field-programmable gate arrays (FPGAs).

This paper extends previous work [16,17] on an accelerator framework for FPGAs that can execute query-based text analytics. The framework consists of a compiler that transforms a user-specified query into a hardware description which is then

* Corresponding author.
E-mail address: pol@zurich.ibm.com (R. Polig).

http://dx.doi.org/10.1016/j.jpdc.2017.05.015
0743-7315/

implemented on an FPGA. Additional operator modules and a set of query optimizations are presented that improve the resource utilization of the compiled queries. System integration is presented where the FPGA is integrated into two generations of enterprise server systems based on IBM's POWER® processor. Using the coherent accelerator processor interface (CAPI) and it is predecessor, the FPGA can operate as an integral part of a software process. The framework is integrated into a Java application demonstrating end-to-end acceleration for the evaluated queries. The main contributions over previous work are

- Evaluation of the framework on a novel FPGA generation and host system generation based on IBM's POWER8® processor.
- In-depth evaluation of the compiler optimizations that are specifically applied to queries running on FPGAs.
- The performance on two different generations of systems is evaluated, and it is demonstrated that both CPUs and FPGAs benefit from a new semi-conductor technology generation.
- Power measurements are provided for both system generations using the in-system power management system.

Next, Section 2 will introduce the concept of query-based text analytics before related work is outlined in Section 3. Section 4 presents the various operators and their according hardware implementations which can be used by the compiler discussed in Section 5. Software and system integration is described in Section 6. The framework is evaluated on two enterprise server systems in Section 7 before concluding this work.

2. Background

This work is based on text analytics systems, which utilize rules to define the information which should be extracted from a set of documents called a corpus. A representative framework is SystemT [12], which was developed by a team at IBM Research — Almaden and is used in several IBM products such as IBM Notes or Infosphere BigInsights. It aims to overcome some of the difficulties that arise from using cascaded grammar-based approaches and their extensions. By simplifying the work-flow, the interaction between various tasks becomes maintainable and can be expressed in a cleaner way. This also allows to decouple the way rules are expressed in a language and how they will be executed by a system.

To achieve this, SystemT uses a declarative rule language called *Annotation Query Language* (AQL), which is very similar to the Structured Query Language (SQL) [5] known from relational database applications. While keeping many relational operations from SQL like e.g. Select, Union or Join, AQL adds text-level features such as regular expressions and dictionary matching that operate on an entire text document or a segment of it. Such an expressive language allows users to define rules or queries in a modular and maintainable way, and are independent of the actual implementation of the operators. Fig. 1 provides an example AQL query, extracting person names from a text document. Entire queries are often referred to as *extractors*.

SystemT is implemented in Java and consists of two main components: a compiler and a runtime. In a first step, the AQL compiler translates the query into an annotation operator graph (AOG). The AOG is an acyclic dependency graph, where the nodes represent individual operators that work on the incoming data of their input edges. The optimizer then derives an execution plan for an AOG by applying transformations and using a cost-based optimization model. This can involve profiling the analysis of a set of reference documents to choose the best performing execution plan.

Once the user is satisfied with the results the developed query produces and the execution plan has been established, it can be deployed on the SystemT runtime. The runtime can be embedded

```
create view Last as
        extract dictionary lastnames.dict on D.text
        as name from Document D;

create view First as
        extract dictionary firstnames.dict on D.text
        as name from Document D;

create view Caps as
        extract regex /[A-Z][a-z]*/ on D.text
        as name from Document D;

create view Person as
select S.name as name
from (
        ( select CombineSpans(F.name, C.name) as name
          from First F, Caps C
          where FollowsTok(F.name, C.name, 0, 0))
   union all
        ( select CombineSpans(F.name, L.name) as name
          from First F, Last L
          where FollowsTok(F.name, L.name, 0, 0))
   union all
        ( select *
          from First F )
) S
consolidate on name;
```

Fig. 1. Example query written using the annotation query language (AQL).

into any Java application that requires text analytics capabilities. This can range from local email clients to large analytics applications. In the latter case, SystemT is often deployed on a large cluster of compute nodes using the Hadoop framework [1]. The SystemT runtime executes the query plan individually on each document it receives. The query plan is completely executed by a single thread over an entire document. Thus, multiple threads are running independently from each other and exploit parallelism from the large number of individual documents. This type of large scale analytics is usually running continuously, processing online data, or for multiple hours on a given large set of documents.

Fig. 2 represents the annotation operator graph for the AQL query shown in Fig. 1. The two main categories of operators used are *extraction* operators and *relational* operators. The extraction operators consist primarily of pattern matching steps such as regular expression matching or dictionary matching and are run over the entire document. This has the effect that these types of operators are often located at the very top or beginning of an AOG. These operators are creating a sequence of segments, which match the pattern they are looking for. Such segments are referred to as *Spans* and are described using a character-based start offset and an end offset. If an extraction operator follows a previous extraction operator, it will operate within the spans produced by its predecessor.

In most cases, the extraction operators are followed by a usually larger number of relational operators. These operators operate on the offset values of the spans produced by parent nodes, creating a new set of spans. A prominent exception is the *Select* operation, which when using a regular expression in its conditional expression requires access to the actual document data of the span it currently operates on.

3. Related work

The presented system is the first to our knowledge that provides end-to-end acceleration of text analytics queries on unstructured data. This makes a quantitative comparison difficult. Other work has focused on various components such as string pattern matching for network intrusion detection or acceleration of relational database queries. The presented approach combines pattern matching on unstructured data with the immediate execution of relational algebra operations on a hardware platform. On a qualitative level, this section provides an overview of related work in

Please cite this article in press as: R. Polig, et al., A hardware compilation framework for text analytics queries, J. Parallel Distrib. Comput. (2017), http://dx.doi.org/10.1016/j.jpdc.2017.05.015.