MASCARA-FPGA cooperation model: Query Trimming through accelerators

VAN LONG NGUYEN HUU [123], JULIEN LALLET [13], EMMANUEL CASSEAU [2], LAURENT D'ORAZIO [23],

[1] Nokia Bell Labs, Lannion, France

[2] Univ Rennes, CNRS, IRISA, Lannion, France

[3] B<>com, Lannion, France

[1]{long.nguyen_huu, julien.lallet}@nokia.com, [2]{emmanuel.casseau, laurent.dorazio}@irisa.fr

Processing sequences of online queries with an efficient response time is crucial for the new data management systems. To address this challenge, we proposed the ModulAr Semantic CAching fRAmework (MASCARA), in which Semantic Caching (SC) is used for a fast and accurate logs processing tool in a security monitoring system. Although MASCARA pointed out the expensive modules, we have not yet discussed an architecture in which they leverage the accelerators of Field Programmable Gate Array (FPGA). Therefore, in this paper, we propose a MASCARA-FPGA cooperation model and related major accelerators. We also explain the pipeline model in which multiple accelerators can run in parallel. Because Query Trimming can reduce the response time by factors of up to 2.8x for a single kernel, we confirm the potential of building the full MASCARA-FPGA model.

Additional Key Words and Phrases: FPGA acceleration, semantic caching, query optimisation

1 INTRODUCTION

In recent years, Big Data has led to the emergence of new data management systems as an alternative to Relational Database Management Systems (RDBMS), such as: column-stores, or Massive Parallel Processing (MPP). Some of these systems can be deployed on a specified query engine, such as Spark [1], on top of a distributed file system. As a case study, an IoT security monitoring system with the HTTP log analysis tool is designed to ensure the safety by observing unusual events. This monitoring tool is based on a variety of specified information reasoning from an enormous number of access logs. For example, this tool can consist of select-project queries to get the traffic for a given time. Thus, reducing the response time of these online queries is a real challenge.

To achieve this goal, there are two basic approaches: *1) Speeding up the execution time of a query by offloading it from CPU to a high-speed computing unit*; or *2) Exploiting resources and knowledge contained in the queries themselves by a caching system.* With the first approach, to overcome the CPU limits, the trend is evolving to explore the acceleration of query execution via Field Programmable Gate Arrays (FPGAs) [5]. FPGAs with partial runtime-reconfiguration capability can close the gap between flexibility of CPU and the performance of specialized hardware accelerators. In the mean time, with the second approach, the query engines have not yet implemented any technique to exploit the semantic relevance between $Q1$ and $Q2$. *Therefore, our goal is to build a query processing platform on FPGA with semantic caching (SC) as a combination of these two approaches to optimize the performance.*

Related works On one hand, we have some noticeable accelerating approaches on FPGA without constructing a cache or applying the semantic theory. Sukhwani et al. [12] proposed FGPA accelerated database system in which the analytics queries are offloaded from the host running the workload to the FPGA composable accelerators. Mueller et al. [8] proposed Glacier, the query to hardware compiler, which can compose a digital circuit using various implemented operators. This approach is only suitable for a known query set. Dennl et al. [3] presented concepts for multi-processing queries in MySQL. This approach focuses on restriction and aggregation operators and thus cannot evaluate more complex queries. Sidler et al. [11] presented DoppioDB, which extends MonetDB on FPGAs. In cooperation with DoppioDB, Owaida et al. [9] provided the Centaur framework to bridge the gap

between CPUs and FPGAs. Recently, Wang et al. [14] approached the advantage of programmability offered by the OpenCL to generate an optimal plan for relational query processing on FPGA.

On the other hand, there are some remarkable researches or proposals regarding semantic exploitation. SC was studied and proved by Dar et al. [2] or Keller et al. [7] as an effective solution compared to pages or tuples caching. Moreover, we have the research of Zhou et al. [15] to exploit the semantic equivalence between queries. Although these researches were not presented in the context of acceleration on FPGA, they confirmed that data reasoning based on the semantic approach could speed up the performance of the data management system. Meanwhile, Salami et al. [10] proposed an efficient cache on FPGA to store partly or entirely the hash table for hash join operator where CPU is difficult to have strong robustness and high speed. Unfortunately, their cache did not exploit the advantages of SC or query equivalence.

Our previous works. *To the best of our knowledge, proposing SC on FPGA has not yet been studied.* Thus, we had a vision of multi-layer framework for IoT security monitoring tool [4] without addressing the semantic processing. Later, we presented ModulAr Semantic CAching fRAmework (MASCARA) [6] that describe the semantic processing on host only. We also pointed out which module in MASCARA is expensive in computing time so that it can be interesting to offload and accelerate by FPGA.

Contributions. This paper address the following challenge: *we have not yet had an MASCARA architecture from the hardware perspective in which the modules are expressed on FPGA to leverage its computing kernels.* Therefore, in order to address this challenge, our objective is to present the following new contributions:

(i) A cooperation model in which MASCARA's stages or modules are addressed on FPGA as computing kernels.

(ii) Query Trimming kernels in the model of pipeline execution .

(iii) The preliminary results from single to multiple kernels to confirm the acceleration of Query Trimming on FPGA.

2 MASCARA ARCHITECTURE

This section presents briefly the SC as an optimization solution in which the modules of MASCARA are described.

Semantic Caching. SC enables data reasoning effectively, reducing data transmission on servers, and exploiting the relevance of query [2], [7]. Consider that SC stores a semantic entity *SE*: $(date <' 17/03/2020-10:00:00')$ that points to a result set of data in cache. Then, an incoming query *INCQ*: $(ip =' 208.115.111.72')$ would be divided into: *PROBEQ*: $(date <' 17/03/2020-10:00:00' \wedge ip =' 208.115.111.72')$ processed in cache to retrieve the portion of the result available. Meanwhile, *REMAINQ*: $(date \neq' 17/03/2020-10:00:00' \wedge ip =' 208.115.111.72')$ has to retrieve any missing tuples from the server.

MASCARA overview. Based on this approach, the MASCARA [6] is composed of 4 main stages: *(1) Query Broking*, *(2) Query Trimming*, *(3) Semantic Management*, and *(4) Result refining*. The modular approach allows us to reuse cache services or focus only on developing a cache solution. In MASCARA's previous work, all of the modules are deployed on the host at the software level.

Within MASCARA, the Query Broking stage materializes the query into various types of elements. The elements of an incoming query would be turned and stored as a tuple $Q < Q_R, Q_A, Q_P, Q_F, Q_C >$ with R is a Relation, A is an Attribute, P is a Predicate, F is an aggregate Function (SUM, MAX, MIN), and C is the number of records. This semantic tuple Q will be trimmed to Probe Queries PQ(s) and Remainder Queries RQ(s) by Query Trimming stage based on its relationship with semantic segments $S < S_R, S_A, S_P, S_F, S_C >$ that are stored in Semantic Management stage. According to SC theory, each S_i points to the result set of data that is presented by Semantic Data Region in the cache. Last, the Result Refining stage will create the final answer for the user by combining the result sets of PQ(s) and (RQ)s together.

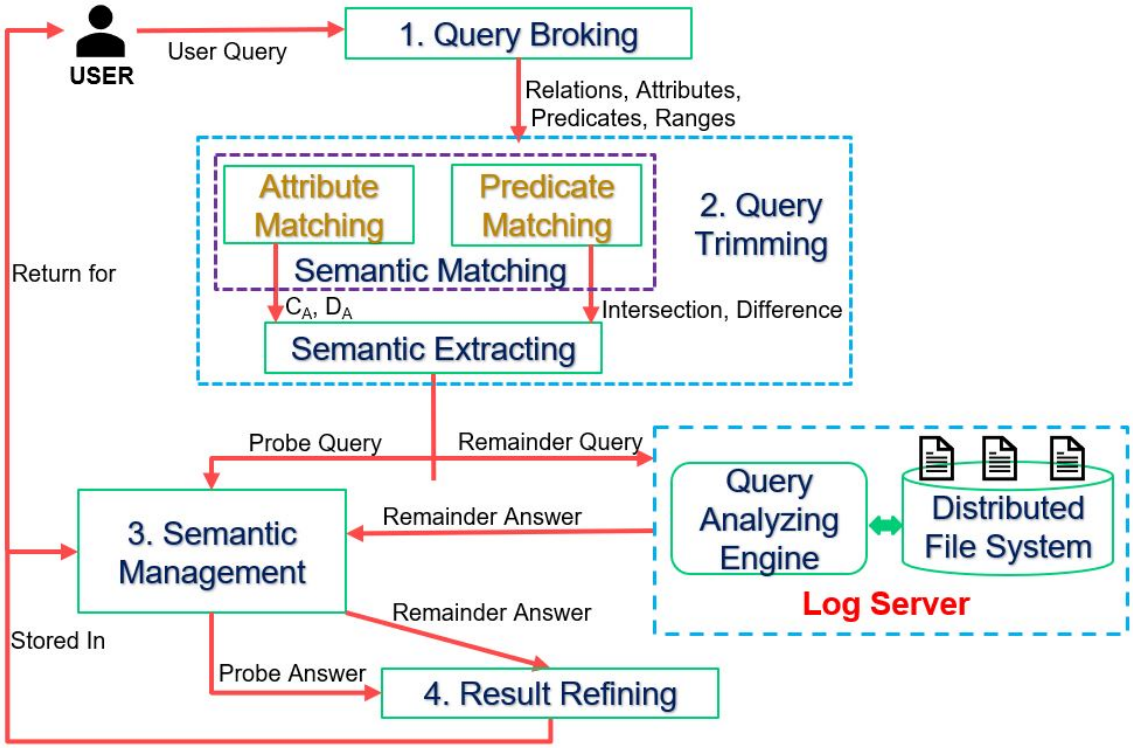

Fig. 1. An overview of MASCARA architecture

Query Trimming. This stage plays an important role due to its complex procedure of rewriting the incoming query to PQ(s) and RQ(s) [6]. Thus, we proposed two sub-stages: *Semantic Matching* and *Semantic Extracting*, that could be executed in parallel.

Semantic Matching is a process in which we have Attribute Matching of (Q_A, S_A) and Predicate Matching of (Q_P, S_P) from the input pair (Q, S). The Attribute Matching computes the Common Attribute $(C_A = Q_A \cap S_A)$ and Difference Attribute $(D_A = Q_A - S_A)$ from (Q_A, S_A). Meanwhile, the Predicate Matching identifies the Intersection $(Q_P \cap S_P)$, Difference $(Q_P \setminus S_P)$ and Implication $(Q_P \rightarrow S_P)$ relationship from (Q_P, S_P).

Based on the results of them, the output of Semantic Matching could be one of the following case: Totally Contained, Horizontal Matching, Vertical Matching, Mixed Matching and NO Matching that are presented as semantic box approach in [2]. For each case, the Semantic Extracting will create the PQ(s) and RQ(s) as outputs.

Limitation. Up to now, MASCARA hasn't addressed the stages or modules as accelerated kernels on FPGA, such as: the Query Trimming engine, or the cache in Semantic Management. Thus, to address this challenge, our vision is to propose a cooperation model in which we leverage the computing kernels of FPGA.

3 MASCARA-FPGA COOPERATION MODEL

In this section, we introduce a hardware perspective model to address the cooperation between MASCARA and relevant major components on FPGA for semantic processing and query executing (as shown in Figure 2). Then, we present the accelerators of Query Trimming within a pipeline execution model.

Query Reformer. Besides the MASCARA's stages, we also have Query Reformer engine to remove the unnecessary elements in queries before materializing them into Q by Query Broking stage. As a result, by improving the appearance of the query, this engine will reduce the complexity of generated remainder queries later.

FPGA-Adapter. To maintain the communication between modules on host and FPGA, we use an adapter to transfer Q and also the data blocks through the PCIe interface. This FPGA-Adapter has to compose a foreign-data wrapper, i.e., Java Wrapper, to build the calling method of accelerators.

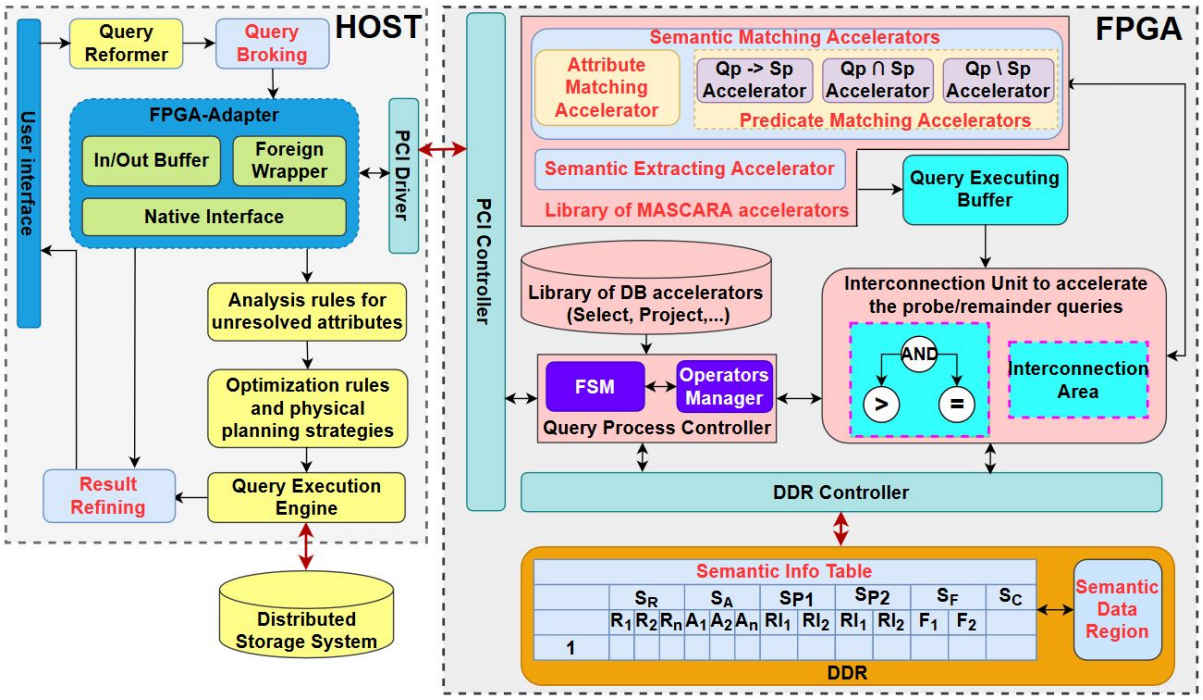

Fig. 2. MASCARA-FPGA co-operation model and its major components. The stages of MASCARA are expressed in red font, i.e. Semantic Matching Accelerator

MASCARA on FPGA. Because the Query Trimming is expensive in computing time compared to the other stages of MASCARA, we want to offload it to FPGA [6]. Moreover, Query Trimming needs to cooperate with Semantic Management to generate the output PQ(s) and RQ(S). Thus, placing also the Semantic Management on FPGA is a complementary solution to reduce the data communication between host and FPGA. Our vision is implementing the modules of Query Trimming and Semantic Management on FPGA using the same concepts of query rewriting, cache organization and replacement strategy that are presented in our previous work [6].

For Query Trimming, we construct different sets of accelerators that are relevant to semantic matching and extracting, such as accelerators of Attribute Matching or Predicate Matching. They can be invoked from the host or other accelerators directly according to execution data flow. Meanwhile, we place the Semantic Management on DDR of FPGA and divide it into two parts: Semantic Info Table to store list of S_i and Semantic Data Region to store relevant result set of S_i. These DDRs have a high bandwidth to and from the configurable region of MASCARA to ensure the performance of Query Trimming's kernels. For each S_i, we support to store the element

$< S_P >$ in forms of Disjunctive Normal Form (DNF) in which we have $P_1 \vee P_2 \vee ... \vee P_n$. In detail, a predicate P_i is separated to the Range Indicators RI in forms of Conjunctive Normal Form (CNF): $P_i = RI_1 \wedge RI_2 \wedge ... \wedge RI_n$. A RI is an object that could be represented as an interval $(LowerBound, UpperBound)$ or an $ExactValue$. Currently, we provide the comparison operators: $<, >, =, \geq, \leq, \neq$ for RI.

Query Process Controller. QPC controls the trimming life cycle of Q from Semantic Matching to Semantic Extracting accelerators. To leverage the most out of FPGA's power, PQ(s) and RQ(s) are considered to be accelerated by using the library of database (DB) operators within the QPC's management. If PQ(s) or RQ(s) contains any unsupported operator, they would be transferred and executed on the host instead. Of course, for the execution of RQ(s), we take into account the challenge of traditional data offloading mechanism where the overhead of data transmission from host to FPGA accelerators can occur.

Based on the status of Operator Manager, the Finite State Machine (FSM) creates the control signals to the Interconnection Unit in which a physical planning is constructed for query executing.

3.1 MASCARA accelerators

We present the data flow of Query Trimming on FPGA in Figure 3 to emphasize the effectiveness of accelerators, such as: Attribute Matching, Predicate Matching, and Semantic Extracting. Thus, the most advantage of FPGA kernels is that the throughput of Query Trimming (PQ, RQ) increases by processing each pair (Q, S_i) in parallel from matching to extracting.

Attribute Matching kernel. We take $Q < Q_{A1,A2,...,An} >$ from Q and $S < S_{A1,A2,...,Am} >$ from S_i as two inputs. Using a nested iteration, we compare the attributes of two input arrays. As a result, the outputs could be maintained as the collection: C_A and D_A.

Predicate Matching kernel. We take $Q < Q_{P1,P2,...,Pn} >$ and $S < S_{P1,P2,...,Pm} >$ as two inputs. The Predicate Matching accelerator is more complex than the Attribute Matching accelerator. We have to implement three engines: *Intersection*, *Difference*, and *Implication* from predicates to ranges. Predicate Matching is expensive due to the complexity of Difference engine when its computing becomes intense quickly with a large number of Q_P or S_P.

Semantic Extracting kernel. Using the outputs from Attribute and Predicate Matching, we can construct PQ and RQ(s) in the Semantic Extracting kernel. For example, D_A is used to build $RQ1 < RQ_A >$ meanwhile D_P is used to build $RQ1 < RQ_P >$. As soon as PQ and RQ(s) appear, they will be stored in the buffer of execution and wait for the decision of QPC that they could be executed/accelerated on FPGA or not based on the available DB accelerators.

3.2 Pipeline model for multiple accelerators

Because these MASCARA accelerators can be executed in parallel, we gain two main benefits: 1) We can apply a pipeline model to process the pair of (Q, S_i) as soon as the accelerators are available; 2) Suppose that hardware resources of FPGA are sufficient to create the multiple instances of accelerators, we could reduce the total matching time between Q and a set of S_i.

Assuming that we have two instances for each MASCARA's accelerators. From a theoretical point of view, we consider the execution times of accelerators are equal without regarding the complexity of their prototypes. As shown in Figure 4, we can execute $(Q1, S_1)$, $(Q1, S_2)$ in parallel and forward result to Extracting as next stage of pipeline. Thus, the throughput of Query Trimming with multiple accelerators in the pipeline model will be increased compared to the sequential or purely pipeline version. Meanwhile, from the practical point of view, to build this pipeline model for multiple accelerators, we have to cope with defining the streaming connections and managing the data flow between pipeline stages.

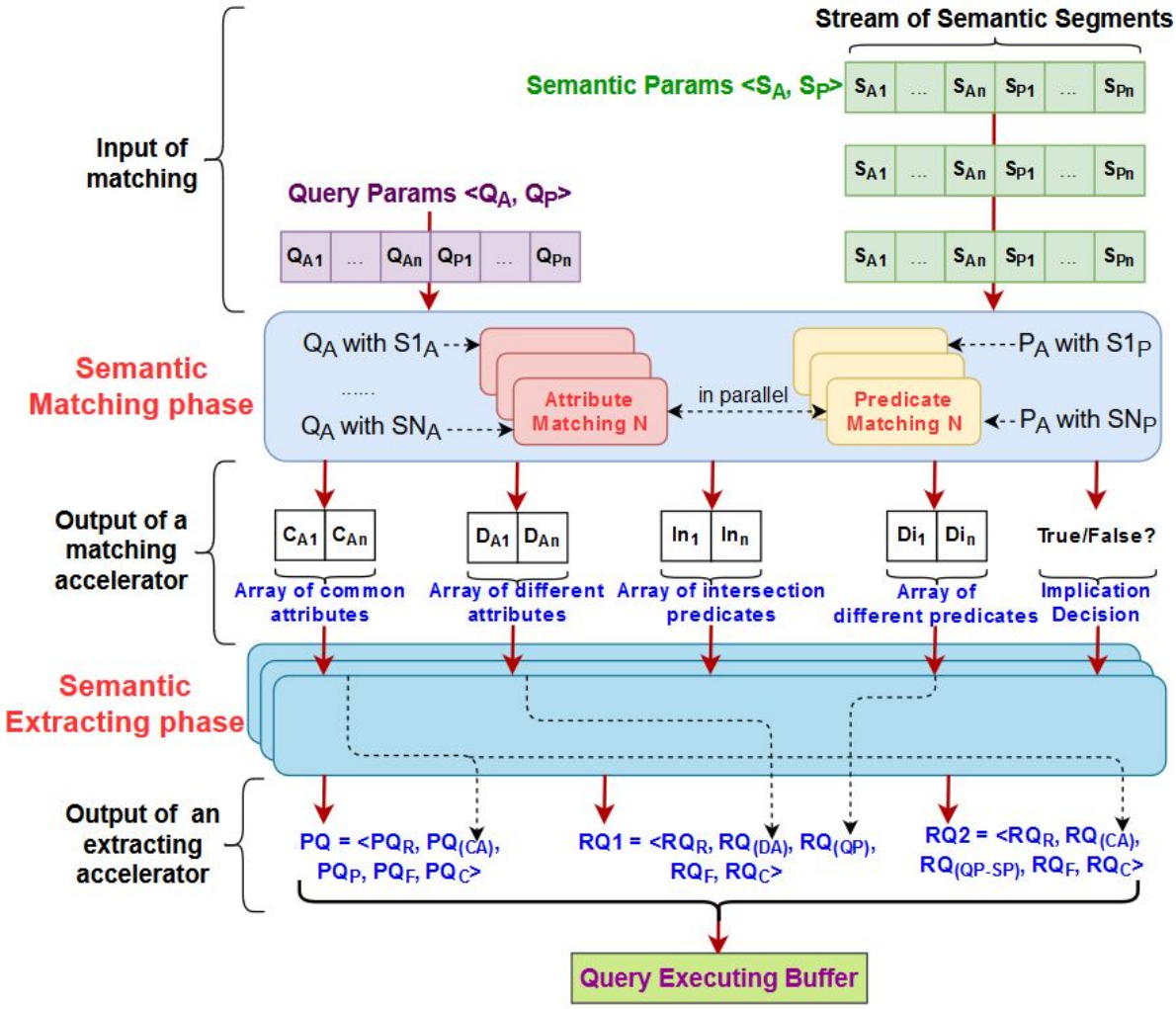

Fig. 3. Dataflow of Query Trimming on FPGA

4 PRELIMINARY EXPERIMENTS

The completely MASCARA-FPGA becomes a real challenge in which we need the kernels to accelerate: 1) Query Trimming, and 2) Semantic Data Management for query execution. Therefore, to show the potential of our model, in this experiment, we currently focus on confirming the capability of Query Trimming on FPGA without accelerating their outputs (PQs, RQs) based on Semantic Data Management.

Configuration. The host is an Intel® Core™ i7-6700 CPU 3.40GHz, and FPGA is an Alveo U200 Xilinx Card. To develop the accelerators, we use the Vitis™ platform from Xilinx.

Workloads. Because we have not accelerated yet the PQ(s) or RQ(s) from Query Trimming, we only need to construct a list of segments S to match and extract with query Q. Thus, we initialize Semantic Info Table on DDR with 100 segments that has up to 15 attributes based on EDGAR log file data set of SEC.gov [13]. We create

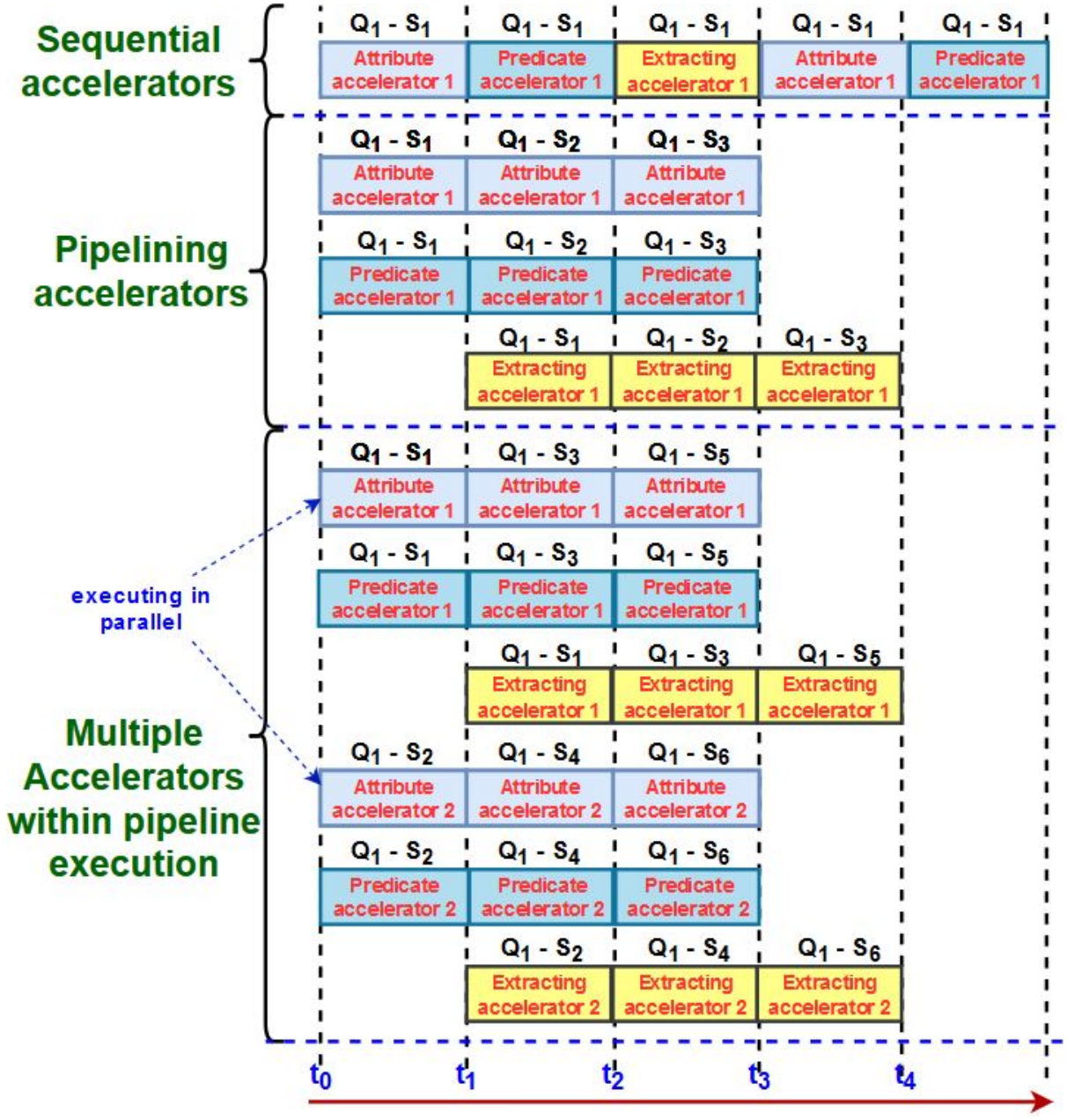

Fig. 4. Query Trimming kernels in pipeline model

four sets of Select-Project queries but later, for the completely MASCARA-FPGA model, we plan to use TPC-DS benchmark.

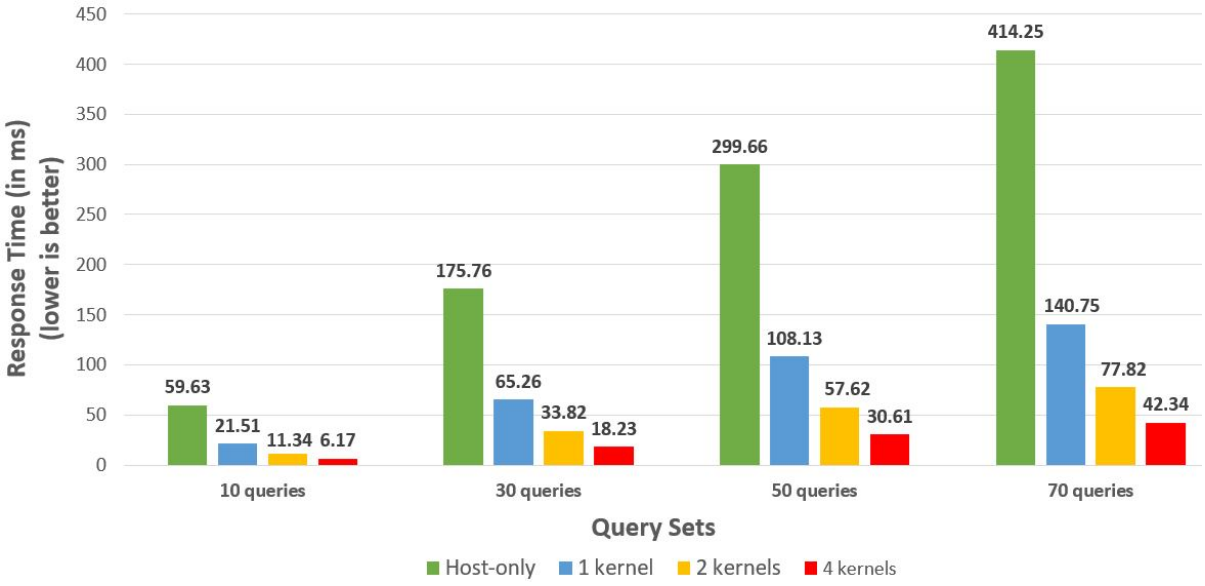

Fig. 5. Total response time of Query Trimming stage

Evaluation. An advantage of Query Trimming is that we do not need too much hardware resources. In detail, it consumes 5703 FFs, 7535 LUTs and 31 BRAMs. We compare the response time of Query Trimming through host only version and multiple kernels on FPGA. As shown in Figure 5, the host only version (green) always has the lowest performance. In particular, if we just use only one Query Trimming kernel (blue), the average response time is reduced by factors of up to **2.8x**. Moreover, we confirm that the more kernels we have, the better speed-up we can get. For example, with 4 kernels (red), we have a better response time by factors of up to **9.6x**.

The management of multiple kernels was presented through Figure 4 in section 3.2. In complete MASCARA-FPGA later, although we can create more kernels, we have to optimize their instances in context of the outputs execution (PQs, RQs).

5 CONCLUSION & FUTURE WORKS

In this paper, we propose the MASCARA-FPGA cooperation model in which stages or modules are addressed as computing kernels. Through preliminary results of Query Trimming, we confirm that building a completely MASCARA-FPGA is potential.

Future work includes the full implementation of MASCARA-FPGA for Select-Project-Join queries with Aggregation. Moreover, we have to study the effectiveness of data organization, i.e, dictionary-based approach instead of raw data, or replacement strategy for Semantic Data Region on FPGA.

REFERENCES

[1] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia). 1383–1394.

[2] Shaul Dar, Michael J. Franklin, Björn Þór Jónsson, Divesh Srivastava, and Michael Tan. 1996. Semantic Data Caching and Replacement. In *Proceedings of the 22th International Conference on Very Large Data Bases* (Mumbai, India). 330–341.

[3] Christopher Dennl, Daniel Ziener, and Jürgen Teich. 2013. Acceleration of SQL Restrictions and Aggregations through FPGA-Based Dynamic Partial Reconfiguration. In *21st IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM* (Seattle, WA, USA). 25–28.

[4] Laurent d'Orazio and Julien Lallet. 2018. Semantic Caching Framework: An FPGA-Based Application for IoT Security Monitoring. *OJIOT* 4 (2018), 150–157.

[5] Jian Fang, Yvo T. B. Mulder, Jan Hidders, Jinho Lee, and H. Peter Hofstee. 2020. In-memory database acceleration on FPGAs: a survey. *VLDB J.* 29 (2020), 33–59.

[6] Van Long Nguyen Huu, Julien Lallet, Emmanuel Casseau, and Laurent d'Orazio. 2020. MASCARA (ModulAr Semantic CAching fRAmework) towards FPGA Acceleration for IoT Security Monitoring. *OJIOT* 6 (2020), 14–23.

[7] Arthur M. Keller and Julie Basu. 1996. A Predicate-Based Caching Scheme for Client-Server Database Architectures. *VLDB J.* 5 (1996), 035–047.

[8] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2010. Glacier: A Query-to-Hardware Compiler. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA). 1159–1162.

[9] M. Owaida, D. Sidler, K. Kara, and G. Alonso. 2017. Centaur: A Framework for Hybrid CPU-FPGA Databases. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (Napa, CA, USA). 211–218.

[10] Behzad Salami, Oriol Arcas-Abella, Nehir Sönmez, Osman S. Unsal, and Adrián Cristal Kestelman. 2016. Accelerating Hash-Based Query Processing Operations on FPGAs by a Hash Table Caching Technique. In *2016 High Performance Computing - Third Latin American Conference (CARLA)* (Mexico City, Mexico). 131–145.

[11] David Sidler, Zsolt Istvan, Muhsen Owaida, Kaan Kara, and Gustavo Alonso. 2017. DoppioDB: A Hardware Accelerated Database. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA). 1659–1662.

[12] Bharat Sukhwani, Mathew Thoennes, Hong Min, Parijat Dube, Bernard Brezzo, Sameh Asaad, and Donna Dillenberger. 2015. A Hardware/Software Approach for Database Query Acceleration with FPGAs. *International Journal of Parallel Programming* 43 (2015), 1129–1159.

[13] U.S.Securities and exchange commission. 2018. EDGAR Log File. Retrieved March 15, 2021 from https://www.sec.gov/dera/data/edgar-log-file-data-set.html

[14] Z. Wang, J. Paul, H. Y. Cheah, B. He, and W. Zhang. 2016. Relational query processing on OpenCL-based FPGAs. In *26th International Conference on Field Programmable Logic and Applications (FPL)* (Lausanne, Switzerland). 1–10.

[15] Qi Zhou, Joy Arulraj, Shamkant Navathe, William Harris, and Dong Xu. 2019. Automated Verification of Query Equivalence Using Satisfiability modulo Theories. *Proc. VLDB Endow.* 12 (2019), 1276–1288.