Thesis for The Degree of Licentiate of Engineering

Streaming Aggregation using Reconfigurable Hardware

Prajith Ramakrishnan Geethakumari

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2019

Streaming Aggregation using Reconfigurable Hardware

Prajith Ramakrishnan Geethakumari

Advisor: Ioannis Sourdis, Prof. at Chalmers University of Technology
Co-Advisor: Pedro Trancoso, Prof. at Chalmers University of Technology
Examiner: Ulf Assarsson, Prof. at Chalmers University of Technology
Discussion Leader: Jens Teubner, Prof. at TU Dortmund

Technical Report No 205L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2019.

Abstract

High throughput and low latency streaming aggregation is essential for many applications that analyze massive volumes of data in real-time. In many cases, high speed stream aggregation can be achieved incrementally by computing partial results for multiple windows. However, for particular problems, temporarily storing all incoming raw data to a single window before processing is more efficient or even the only option. This thesis presents the first FPGA-based single window stream aggregation designs for tuple-based and time-based windowing policies. The proposed approach is able to support challenging queries required in realistic stream processing problems. More precisely, holistic, distributive, and algebraic aggregation functions, as well as custom ones can be supported. Our designs offer aggregation for large number of concurrently active keys and handles large window sizes and frequent aggregations. Maxeler's dataflow engines (DFEs), which suit well the stream processing characteristics, are used to implement the designs. DFEs have a direct feed of incoming data from the network as well as direct access to off-chip DRAM. The tuple-based single window DFE processes up to 8 million tuples-per-second (1.1 Gbps) offering 1-2 orders of magnitude higher throughput than a state-of-the-art stream processing software system. The processing latency is less than 4 µsec, 4 orders of magnitude lower latency than software. The time-based single-window stream aggregation DFE offers high processing throughput, up to 150 Mtuples/sec, similar to related GPU systems, which however do not support both time-based and single windows. It also offers an ultra-low processing latency of 1-10 µsec, at least 4 orders of magnitude lower than software-based solutions.

Keywords

Streaming, Aggregation, Dataflow, FPGA

Acknowledgment

I would like to extend my deepest gratitude to my advisor, Yiannis, for being this pillar of support guiding me through the ups and downs during the course of my studies. I would like to thank my co-advisors Pedro and Joel for their valuable help and support during my studies. A big thanks to Vincenzo for his valuable insights and suggestions for my research. I would also like to thank Maxeler Technologies for providing the infrastructure for running the experiments. Also a big thanks to all the nice people at Chalmers for creating a great environment to work in.

This work was partly funded by the Swedish Research Council under the ScalaNetS project (2016-05231).

List of Publications

Appended publications

This thesis is based on the following publications:

[A] Prajith Ramakrishnan Geethakumari, Vincenzo Gulisano, Bo Joel Svensson, Pedro Trancoso and Ioannis Sourdis
"Single Window Stream Aggregation using Reconfigurable Hardware"
International Conference on Field Programmable Technology (ICFPT) 2017, Melbourne, Australia.

[B] Prajith Ramakrishnan Geethakumari, Vincenzo Gulisano, Bo Joel Svensson, Pedro Trancoso and Ioannis Sourdis
"Time-SWAD: A Dataflow Engine for Time-based Single Window Stream Aggregation"
International Conference on Field Programmable Technology (ICFPT) 2019, Tianjin, China.

Other publications

The following publications were also published during my PhD studies. However, they are not appended to this thesis because their contents are not related to the thesis.

[a] Andreas Brokalakis, Nikolaos Tampouratzis, Antonios Nikitakis, Ioannis Papaefstathiou, Stamatis Andrianakis, Danilo Pau, Emanuele Plebani, Marco Paracchini, Marco Marcon, Ioannis Sourdis, Prajith Ramakrishnan Geethakumari, Maria Carmen Palacios, Miguel Angel Anton and Attila Szasz
"COSSIM: An Open-Source Integrated Solution to Address the Simulator Gap for Systems of Systems"
Euromicro Conference on Digital System Design (DSD) 2018, Prague, Czech Republic.

[b] Yang Ma, Prajith Ramakrishnan Geethakumari, Georgios Smaragdos, Sander Lindeman, Vincenzo Romano, Mario Negrello, Ioannis Sourdis, Laurens W.J. Bosman, Chris I. De Zeeuw, Zaid Al-Ars and Christos Strydis
"Towards real-time whisker tracking in rodents for studying sensorimotor disorders"
International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) 2017, Pythagorion, Greece.

Contents

Chapter 1

Introduction

The number of connected devices grows rapidly along with the amount of data they produce and exchange. Processing such *big data* brings tremendous opportunities in various domains (e.g. financial, transportation) enabling real-time sophisticated decisions that were never possible before. However, realtime analysis of unbounded streams is challenging, it requires *high processing throughput* to cope with massive volumes of data and *low latency* to respond fast.

Streaming aggregation is one of the fundamental and computationally challenging types of stream processing. Streams of values (tuples) are handled in windows of a particular size, W_{S}, which are "slid" by a particular window advance, W_{A}. Then, an aggregation output is produced per window, computed based on a *function* that uses as input the window values. The result of a Sliding Window streaming AGgregation (SWAG) is a stream of aggregated values [1]. Often values in a stream are grouped by a key, then, values of different keys are processed separately.

Some aggregation functions can be computed incrementally using either multiple windows [2] or panes [3, 4]. *Incremental aggregation* computes and stores partial results, rather than storing all the incoming values of a window before computing the full function. In doing so, usually memory pressure is reduced (both capacity and bandwidth) and performance is improved [5]. However, for some queries, especially with small W_{A}, incremental aggregation has the opposite effect causing an excessive number of memory accesses that limit performance [5], e.g., cases of processing geo-tagged data [6], social media data [7] or manufacturing equipment data [8]. Then, a *single window*, non-incremental approach that explicitly stores all the incoming values in a single window before processing is more efficient [5]. More importantly, storing values in a single window is unavoidable when computing some holistic functions; these are functions that have no constant bound on the size required to store a partial result, such as `median` [9]. In general, non-incremental, single-window SWAG is more suitable for general data mining and machine learning functions such as classification (e.g., decision trees, random forest, support vector machines, KNN), as well as for most of the inductive machine learning algorithms [10]. However, *temporarily storing all incoming values before processing puts significant pressure to the memory*, which often becomes

the bottleneck.

Stream processing and stream aggregation systems in particular have been implemented on various compute platforms. Multicores and GPU are able to sustain high processing throughput but fall short in delivering low latency [11]. They require redundant memory accesses managed by an operating system to store incoming tuples in DRAM even before processing starts. This, besides the long latency, wastes a significant fraction of valuable memory bandwidth reducing performance. On the contrary, FPGAs provide both high processing throughput and low latency. Customized dataflow engines, which naturally match the stream processing characteristics, can be implemented in reconfigurable hardware achieving high throughput [11]. Furthermore, incoming tuples can be fed to an FPGA through a direct network connection with low latency, avoiding unnecessary DRAM accesses.

So far, current FPGA solutions focus on incremental aggregation approaches using multiple window or pane-based designs [2–4]. As a consequence, queries that require small W_A or use holistic functions have poor performance or are not supported at all. Moreover, most existing FPGA designs do not use external DRAM and therefore support a single key or at most a handful of keys, and small window sizes, which are not practical for many real stream processing problems, such as the ones mentioned before [6–8].

This thesis addresses the above problems proposing a single window approach for streaming aggregation using FPGAs for computing holistic aggregation functions with high throughput and low latency.

The rest of this introductory Chapter is organized as follows: The problem statement is presented in Section 1.1 followed by a discussion of the objectives of this thesis in Chapter 1.2. Finally, the contributions of this thesis are summarized in Section 1.3.

1.1 Problem Statement

The primary problem addressed in this thesis is to achieve high throughput and low latency SWAG for holistic aggregation functions. The problem is solved for SWAGs for the two alternative *windowing policies* which dictate the way the window is defined.

One way is to define the window based on the *number of tuples* it contains. *Tuple-based* windows always contain (and are slid by) a fixed number of tuples. They are suitable for applications with fixed data rates and have a fixed memory footprint.

Alternatively, a sliding window can be defined by a *time interval*. *Time-based* SWAG is more commonly used as it is less restrictive allowing varying data arrival rates which naturally fits the time-series data produced by most Internet-of-Things (IoT) devices [6, 10, 12]. The number of tuples contained in a time-based window can vary making the memory and compute resources needed to produce the aggregation result unpredictable.

This thesis addresses the problem of achieving high throughput and low latency SWAG for holistic aggregation functions for both the above windowing policies, namely, tuple-based and time-based.

1.2 Thesis Objectives

Below follows a more detailed description of the objectives with some related work and the approach pursued in this thesis.

1.2.1 Tuple-based Stream Aggregation for Holistic Functions

The first design aims to achieve high throughput and low latency tuple-based SWAG for holistic functions. The objectives of this design are to:

- *Support holistic functions;*
- *Support aggregation queries with large window sizes, small window advances and multiple aggregation functions;*
- *Minimize memory accesses to maximize processing throughput;*
- *Utilize dataflow computing to maximize processing throughput; and*
- *Utilize direct network and off-chip memory connectivity to minimize latency.*

Related Work: Software-based distributed stream processing engines are easy to configure, flexible to allow for a multitude of operations and analysis on the data [13–15]. Nevertheless, as with any general-purpose software implementation, their performance depends on the underlying hardware and can never match the throughput or latency offered by dedicated implementations (e.g. custom FPGA-based systems). In the particular focus of this thesis, software approaches are not able to cope with the challenges posed by aggregating on large windows (large W_S) and at high rates (small W_A). Other CPU-based sliding window aggregation algorithms use data structuring and algorithmic techniques [1, 16] and improve latency of in-memory aggregation but are constrained to associative aggregation functions.

GPU based stream processing systems achieves high throughput but at high latency of hundreds of milliseconds for aggregation queries [10, 17]. Moreover, [17] only supports incremental aggregate computations utilizing the commutative and associative property of some aggregation functions and therefore can implement only distributive (`count`, `sum`) and algebraic (`average`) functions. As opposed to [17], [10] supports holistic functions at high processing throughput by offloading batches containing, in the order of thousands of windows to the GPU, which negatively impacts the processing latency. Moreover, [10] supports only a single key.

FPGA-based stream processing systems supports sliding window aggregation for distributive (`count`, `sum`, `min`, `max`) and algebraic (`average`) aggregation functions [2,4]. Nevertheless, both designs use only the on-chip BRAMs for storing aggregation states and do not use DRAM. In turn, this prevents the design from performing stream aggregations of realistic sizes (number of keys, W_S). Moreover, [2, 4] does not support holistic functions, as it relies on incremental aggregations.

Thesis Approach: The *paper A* [5] in this thesis addresses the above limitations describing a FPGA-based design for *single window tuple based* stream aggregation.

The performance of an aggregation engine is mostly limited by the DRAM bandwidth. The single window approach fits best large windows producing continuous up-to-date results (*i.e.*, with small window advance) for multiple aggregation functions by incurring fewer memory accesses compared to incremental aggregation algorithms (multi-window and pane-based). As a consequence, single window is a faster stream aggregation choice for queries with such aggregation parameters. The single window approach supports holistic aggregation functions as maintaining all input tuples is necessary independently of whether the function can be computed incrementally. This is due to the non-associative nature of the holistic functions. For such functions, a single window implementation is more efficient. Multi-window would require storing all incoming tuples in each one of the multiple windows resulting in many duplicates, exacerbating the performance bottleneck and pane-based approaches would not be able to maintain partial results.

Our design is dataflow, matching well the stream processing characteristics, and is implemented in a Maxeler N-series FPGA card with direct network and DRAM interfaces [18]. The Dataflow Engine (DFE) is fed with incoming tuples through a direct network connection and provides direct access to DRAM through its own memory controller. With the single window approach, the DFE is able to implement challenging realistic queries of any holistic, distributive or algebraic aggregation function and support large number of keys and window sizes. The single window design provides a high throughput and the DFE's direct network connection and access to external DRAM, enables to achieve a low latency compared to other software and GPU implementations.

1.2.2 Single Window Time-based Stream Aggregation

The second design described in this thesis is to support *time-based* stream aggregation with the benefits offered by the single window approach as described in the previous section.

This design has the following objectives in addition to the ones presented in the previous design:

- *Support fluctuating data arrival rates for time series data; and*
- *Alleviate memory pressure of the single window approach especially for skewed data distributions.*

Related Work: Generic software approaches, such as Apache Flink, Spark, and Storm, running on general-purpose CPU offer a wide range of stream processing capabilities, including support for time-based and holistic aggregations with ease of deployment but have limited throughput and high latency [13–15]. GPU systems are able to achieve high processing throughput, but similar to CPU solutions, they fall short in delivering low latency [11] as they have redundant memory accesses managed by an operating system to store incoming tuples in DRAM even before processing starts [10, 17]. [17] supports only incre-

mental aggregate computations. [10] supports holistic functions but supports only tuple-based windowing policy for a single key.

In *paper A* [5] describing *tuple-based* stream aggregation, dependencies (read after write hazards) between tuples of the same key are resolved with concurrency control queues; this leads to poor performance in case of skewed key distributions which is common in real world data.

A considerable number of previous works have focused on accelerating In-Memory Database (IMDB) operators using FPGAs. One point to note here is that most of the IMDB group-by aggregation designs have some form of optimization for alleviating memory pressure and handling skewed data distributions better. [19] performed only IMDB *group-by* incremental aggregation (`count`) using CAMs to cache recently produced partial results and improve performance. Furthermore, István *et al.* implemented an on DRAM hash table [20], which was subsequently improved using a Cuckoo hash table [21], similar to our design. [20] used a FIFO+CAM structure to implement a write through caching mechanism to specifically prevent read after write hazards in the DRAM read-modify-write pipleline caused due to skewed data distributions. As a side effect, the caching mechanism in [21, 22] manages to reduce only the DRAM reads. As the read queue allows duplicate addresses in [22], the capacity of the cache is also not efficiently utilized.

Thesis Approach: The *paper B* in this thesis introduces the Time-based Single Window stream Aggregation Dataflow engine (Time-SWAD). Time-SWAD addresses the above two challenges of time-based single window stream aggregation and accelerates it using reconfigurable hardware. First, the unbounded number of tuples in a time-based sliding window is facilitated by a flexible circular buffer that stores the window values. We apply the idea of panes [3] to a single window [5] creating a circular buffer that supports bulk evictions. In addition, this buffer can be expanded dynamically with one or more unused identical buffers originally meant for other keys. Thereby, time-based windows of varying size can be stored. Second, the memory pressure of single windows, caused by their need to store all incoming data, is alleviated with a caching scheme. Time-SWAD uses external DRAM in order to support a large number of keys and sufficient volumes of stored values. However, DRAM bandwidth is limited and a caching mechanism is used to merge multiple requests to the same DRAM location without limiting performance in skewed key distribution. Our design is dataflow, matching well the stream processing characteristics, and is implemented in a Maxeler N-series FPGA card with direct network and DRAM interfaces [18].

1.3 Contributions

Following the above approaches towards achieving the objectives of this thesis, the following contributions are made and presented in the papers included in the thesis:

1.3.1 Tuple based Single Window Stream aggregation

We introduce the first FPGA-based design for single window tuple-based stream aggregation. Our approach:

- uses a Maxeler's Dataflow Engine and deep pipelining to provide processing throughput of up to 8 million tuples per second (1.1 Gbps), which is 1-2 orders of magnitude higher processing throughput than a state-of-the-art stream processing software system;
- has a direct network connection to feed incoming tuples as well as direct access to DRAM offering ultra low processing latency of up to 4 µsec, at least 4 orders of magnitude faster than software;
- is at least 1 order of magnitude more energy efficient than a state-of-the-art stream processing software;
- is able to support realistic streaming queries with:
  - multiple holistic and arbitrary user-defined aggregation functions, as well as distributive and algebraic ones;
  - up to 1 million concurrently active keys; and
  - large window sizes storing up to 6144 values per key.

1.3.2 Time-based Single Window Stream aggregation

We introduce the first accelerator for time-based single window stream aggregation. Our approach:

- offers a flexible buffer for variable sized time-based windows;
- introduces a novel caching mechanism to reduce the memory pressure of single window aggregation;
- supports realistic streaming aggregation queries with a large number of concurrently active keys;
- allows to handle multiple, arbitrary (and holistic) aggregation functions;
- achieves high processing throughout of up to 150 Mtuples/sec similar to related GPU systems;
- achieves ultra low processing latency of 1-10 µsec which is at least 4 orders of magnitude lower than CPU and GPU solutions; and
- is 2-3 orders of magnitude more energy efficient than a state-of-the-art stream processing software system.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the design and evaluation of tuple-based single window stream aggregation. Chapter 3 presents the design and evaluation of Time-SWAD, a dataflow engine for time-based single window stream aggregation.