Journal of Systems Architecture 66-67 (2016) 48–60

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Customization methodology for implementation of streaming aggregation in embedded systems

Lazaros Papadopoulos[a,*], Dimitrios Soudris[a], Ivan Walulya[b], Philippas Tsigas[b]

[a] School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), 9 Heroon Polytechneiou, Zographou Campus, 157 80 Athens, Greece
[b] Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 28 November 2015
Revised 3 March 2016
Accepted 23 April 2016
Available online 27 April 2016

Keywords:
Design space exploration
Data streaming
Embedded systems
Performance
Energy

ABSTRACT

Streaming aggregation is a fundamental operation in the area of stream processing and its implementation provides various challenges. Data flow management is traditionally performed by high performance computing systems. However, nowadays there is a trend of implementing streaming operators in low power embedded devices, due to the fact that they often provide increased performance per watt in comparison with traditional high performance systems. In this work, we present a methodology for the customization of streaming aggregation implemented in modern low power embedded devices. The methodology is based on design space exploration and provides a set of customized implementations that can be used by developers to perform trade-offs between throughput, latency, memory and energy consumption. We compare the proposed embedded system implementations of the streaming aggregation operator with the corresponding HPC and GPGPU implementations in terms of performance per watt. Our results show that the implementations based on low power embedded systems provide up to 54 and 14 times higher performance per watt than the corresponding Intel Xeon and Radeon HD 6450 implementations, respectively.

1. Introduction

Efficient real-time processing of data streams produced by modern interconnected systems is a critical challenge. In the past, low-latency streaming was mostly associated with network operators and financial institutions. Processing of millions of events such as phone calls, text messages, data traffic over a network and extracting useful information is important for guaranteeing high Quality of Service. Stream processing applications that handle traditional streams of data were mostly implemented by using Stream Processing Engines (SPEs) running on high performance computing systems.

However, nowadays digital data come from various sources, such as sensors from interconnected city infrastructures, mobile cameras and wearable devices. In the device-driven world of Internet of Things, there is a need in many cases for processing data on-the-fly, in order to detect events while they are occurring. These data-in-motion come in the form of live streams and should be gathered, processed and analyzed as quickly as possible, as they are being produced continuously. Low-power embedded devices or embedded micro-servers [1] are expected not only to monitor continuous streams of data, but also to detect patterns through advanced analytics and enable proactive actions. Applying analytics to these streams of data before the data is stored for post-event analysis (data-at-rest) enables new service capabilities and opportunities.

Streaming aggregation is a fundamental operator in the area of stream processing. It is used to extract information from data streams through data summarization. Aggregation is the task of summarizing attribute values of subsets of tuples from one or more streams. A number of tuples are grouped and aggregations are computed on their attributes in real-time fashion. High frequency trading in stock markets (e.g. continuously calculating the average number of each stock over a certain time window), real time network monitoring (e.g. computing the average network traffic over a time window) are examples of data stream processing, where streaming aggregation along with other operators is used to extract information from streams of tuples.

Streaming aggregation performance is affected a lot by the cost of data transfer. So far, streaming aggregation scenarios have been implemented and evaluated in various architectures, such as

* Corresponding author.
E-mail addresses: lpapadop@microlab.ntua.gr (L. Papadopoulos), dsoudris@microlab.ntua.gr (D. Soudris), ivanw@chalmers.se (I. Walulya), philippas.tsigas@chalmers.se (P. Tsigas).

http://dx.doi.org/10.1016/j.sysarc.2016.04.013
1383-7621/© 2016 Elsevier B.V. All rights reserved.

GPUs, Nehalem and Cell processors [2]. Indeed, there is a trend to utilize low power embedded platforms on running computational demanding applications in order to achieve high performance per watt [3–6].

Modern embedded systems provide different characteristics and features (such as memory hierarchy, data movement options, OS support, etc.) depending on the application domain that they target. The impact of each one of these features on performance and energy consumption of the whole system, when running a specific application, is often hard to predict at design time. Even if it is safe to assume in some cases that the utilization of a specific feature will improve or deteriorate the value of a specific metric in a particular context, it is hard to quantify the impact without testing. This problem becomes even harder when developers attempt to improve more than one metric simultaneously. A similar problem is the porting of an application running on a specific system to another with different specifications. The application usually needs to be customized in the new platform differently, in order to provide improved performance and energy efficiency. The typical solution followed by developers is to try to optimize the implementation of the application on the embedded platform in an ad-hoc manner, which is a time consuming process that may yield suboptimal results. Therefore, there is a need for a systematic customization approach: Exploration can assist the effective tuning of the application and platform design options, in order to satisfy the design constraints and achieve the optimization goals.

Towards this end, in this work, we propose a semi-automatic step-by-step exploration methodology for the customization of streaming aggregation implemented in embedded systems. The methodology is based i) on the identification of the parameters of the streaming aggregation operator that affect the evaluation metrics and ii) on the identification of the embedded platform specification features that affect the evaluation metrics when executing streaming aggregation. These parameters compose a design space. The methodology provides a set of implementation solutions. For each solution, the application and the platform parameters have different values. In other words, each customized streaming aggregation implementation is tuned differently, so it provides different results for each evaluation metric. Developers can perform trade-offs between metrics, by selecting different customized implementations. Thus, instead of evaluating solutions in ad-hoc manner, the proposed approach provides a systematic way to explore the design space.

The main contributions of this work are summarized as follows:

i. We present a methodology for efficient customization of streaming aggregation implementation in embedded systems.
ii. We show that streaming aggregation implemented on embedded devices yields significantly higher performance per watt in comparison with corresponding HPC and general purpose GPU (GPGPU) implementations.

Finally, based on the experimental results of the demonstration of the methodology, we draw interesting conclusions on how each one of the application and platform parameters (i.e. design options) affects each one of the evaluation metrics. The methodology is demonstrated in two streaming aggregation scenarios implemented in four embedded platforms with different specifications: Myriad1, Myriad2, Freescale I.MX.6 Quad and Exynos 5 octa. The evaluation metrics are throughput, memory footprint, latency, energy consumption and scalability.

The rest of the paper is organized as follows. Related work on streaming aggregation and stream processing on embedded systems is presented in Section 2. Section 3 describes the streaming aggregation operator and the design challenges. The design space and the exploration methodology are presented in Section 4. Section 5 presents the demonstration of the methodology and in Section 6 we draw conclusions.

2. Related work

Stream processing on various high performance architectures has been studied in the past extensively. Many works focus on the parallelization of stream processing [7–9]. They describe how the stream processing operators should be assigned to partitions to increase parallelism. The authors in [10] describe another way of improving the performance of streaming aggregation: They propose lock-free data structures for the implementation of streaming aggregation on multicore architectures. The evaluation has been conducted on a 6-core Xeon processor and the results show improved scalability.

With respect to stream processing engines (SPEs), Aurora and Borealis [11] are among the most well known ones. Several works that focus on the evaluation of stream processing operators on specific parallel architectures can be found in the literature. For example, an evaluation on heterogeneous architectures composed of CPU and a GPU accelerator is presented in [9]. The authors of [2] evaluate streaming aggregation implementations on Core 2 Quad, Nvidia GTX GPU and on Cell Broadband Engine architectures. The aggregation model used in this work is more complex, since it focuses on timestamp-based tuple processing.

There exists several works that describe the usage of low power embedded processors to run server workloads. More specifically, many works propose the integration of low-power ARM processors in servers [3,4], or present energy-efficient clusters built with mobile processors [5].

In the area of embedded systems stream processing, several works focus on compilers that orchestrate parallelism, while they handle resource and timing constraints efficiently [12]. A programming language for stream processing in embedded systems has been proposed in [13]. These works are complementary to ours: The conclusions we drive from this work could assist the implementation of efficient compilers and development frameworks for stream programming.

Design space exploration in embedded systems is another area related with the present work. Exploration methodologies have been proposed for tuning at system architecture level [14], for customization of dynamic data structures [15] and of dynamic memory management optimization [16]. These customization approaches are complementary to the one proposed in the present work. Performance and energy consumption of streaming aggregation implementation could improve with effective customization of data structures or of the dynamic memory management of the system.

3. Streaming aggregation

In this Section we provide a description of the streaming aggregation operator and we analyze the design challenges of implementing a streaming aggregation scenario on an embedded platform.

3.1. Streaming aggregation description

Streaming aggregation is a very common operator in the area of stream processing. It is used to group a set of inbound tuples and compute aggregations on their attributes, similarly to the *group-by* SQL statement. In the context of this work, we discuss two aggregation scenarios: *multiway time-based with sliding windows* and *count-based with tumbling windows*.