An FPGA-based High-Throughput Stream Join Architecture

Charalabos Kritikakis, Grigorios Chrysos, Apostolos Dollas, Dionisios N. Pnevmatikatos
Microprocessor and Hardware Laboratory
Technical University of Crete
Chania, Greece
babis_k4@hotmail.com, gregory.chrysos@gmail.com, dollas@ece.tuc.gr, pnevmati@ece.tuc.gr

***Abstract*—Stream join is a fundamental operation that combines information from different high-speed and high-volume data streams. This paper presents an FPGA-based architecture that maps the most performance-efficient stream join algorithm, i.e. ScaleJoin, to reconfigurable logic. The system was fully implemented on a Convey HC-2ex hybrid computer and the experimental performance evaluation shows that the proposed system outperforms by up to one order of magnitude the corresponding fully optimized parallel software-based solution running on a high-end 48-core multiprocessor platform. The proposed architecture can be used as a generic template for mapping stream processing algorithms to reconfigurable logic, taking into consideration real-world challenges.**

Keywords— stream processing; ScaleJoin; join operator; FPGA architecture

I. INTRODUCTION

The data mining research area focuses on the extraction of previously unknown and potentially useful information from raw data. Modern data mining applications require real-time processing of high-volume and high-speed data streams to enhance the value of existing information resources. A fundamental operator for the data stream mining is the stream join operator. Stream join is used for correlating the information from different streams. As the stream join operator is computationally expensive, there are many works that focus on accelerating their processing workload using distributed or parallel frameworks. The ScaleJoin algorithm [1] is a new, parallel formulation of the stream join operator that uses a shared-memory framework. The algorithm offers really high performance results, outperforming any other state-of-the-art stream join implementation.

This work presents the first implementation of the most performance efficient stream join operator, i.e. ScaleJoin, on a reconfigurable platform with impressive performance results when compared to highly optimized codes running on multiprocessors. The contributions of this work are: i) this is the first hardware-based work, to the best of our knowledge, which proposes a reconfigurable architecture for the ScaleJoin stream join algorithm, ii) the proposed hardware-based architecture is scalable and generic as it can be used as template for many other problems that correlate streaming data, iii) the proposed architecture is extensible, as it takes advantage of the parallelism that reconfigurable hardware can offer and iv) the implemented system achieves at least 4x better throughput data rates vs. the fastest stream join multi-threaded solution and at least one order of magnitude higher processing rates than any other multi-core published solution.

The rest of the paper is organized as follows. Section II and III make an introduction on the stream join problem and the ScaleJoin algorithm. Section IV presents the proposed hardware-based architecture of the ScaleJoin algorithm. Section V evaluates the performance of the proposed architecture and compares its performance results with the performance of previously presented works. Section VI presents the related software- and hardware-based works on stream join operation, and Section VII draws the conclusions of this work.

II. STREAMS AND STREAM JOIN

The streams consist of flowing tuples, which are modeled as two components <v, t>, where v is a value (or a set of values depending on the application) and t is the timestamp, which defines the order in the stream sequence. The theoretical infinite size of the streams and the need for real-time processing leads to the limitation of processing on a subset of the incoming tuples, i.e. processing over sliding time-based windows.

The stream join operation takes place on streaming in-order timestamped tuples. During the join process between 2 streams, i.e. R and S, all the tuples from the R stream are "correlated"-compared with all the tuples from the S stream inside the given time-based window. Whenever the "correlation" between two tuples is "successful", a new output tuple is created, combining the attributes of both input tuples. The stream join algorithm follows the three-step procedure, which is proposed by Kang *et al.* [2]. Let W_S and W_R be the time windows, which contain the tuples from S and R streams, respectively, and a given time window size W, the three-step procedure is*: i) compare* t_R *with all* $t_S \in W_S$*, ii) add* t_R *to* W_R *and iii) remove all* $t_i \in W_R : t_i.t_s < t_R.t_s - W$.

The stream join operator has high computational cost. Given that the tuple rate is T tuples per second for both processing streams and the processing window size is W seconds, the system maintains W x T tuples, in total. Hence, T tuples have to be compared with W x T tuples every second. Thus, the total computations for the tuples of each stream are about W x T^2 and the aggregate computational cost for both streams is 2 x W x T^2.

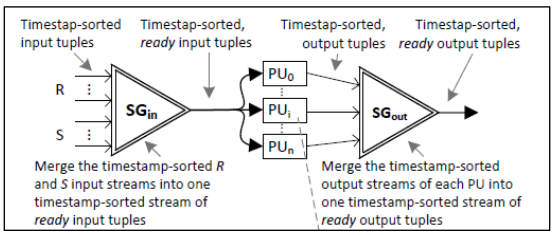

Fig. 1. Software-based Scalejoin algorithm implementation [1].

III. SCALEJOIN ALGORITHM

The ScaleJoin algorithm innovation is based on the abstract data type, namely ScaleGate. The ScaleGate can process in parallel various numbers of streams in a lock-free way. Also, it distributes the incoming tuples to the parallel threads without the need of central coordination. Last, it is, also, used for collecting the correlated output tuples from the parallel threads to the final output.

The main processing unit for the ScaleJoin algorithm software-based implementation is the Processing Unit (PU). As referred above, the ScaleJoin algorithm distributes the processing workload to n parallel PUs, which means that approximately 1/n of the overall comparisons is executed by each PU.

As shown in Fig. 1, the software implementation has 3 main modules. Firstly, the input ScaleGate module, i.e. SGin, receives the timestamp-sorted input tuples from a varying number of physical input streams and merges them into a single-timestamp sorted stream of tuples. Next, the sorted tuples are passed to the parallel processing units, where the main processing takes place. Lastly, the output tuples are sent to the SGout ScaleGate module, which sends out the results.

IV. FPGA-BASED STREAM JOIN SYSTEM

This section presents the proposed reconfigurable architecture for the ScaleJoin algorithm.

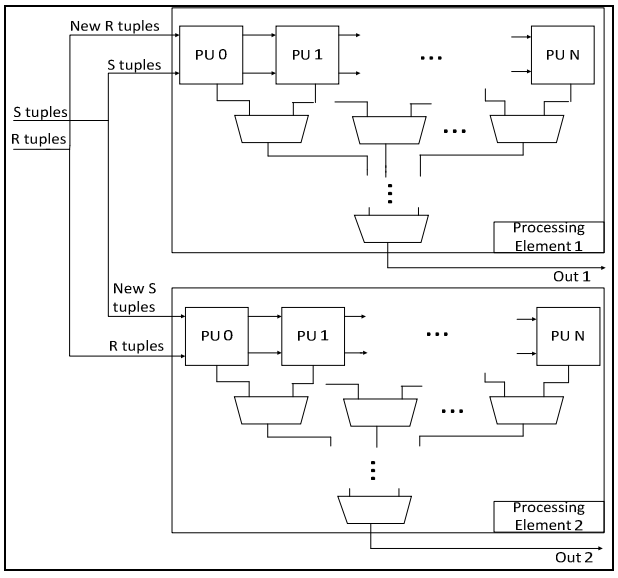

Fig. 2. Reconfigurable StreamJoin architecture

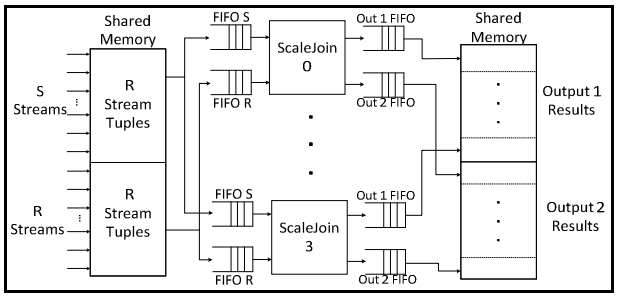

Fig. 3. FPGA-based ScaleJoin architecture

A. *ScaleJoin Module Architecture*

The ScaleJoin module consists of two processing elements (PEs) that work in parallel, as shown in Fig. 2. Each one of them correlates N newly arrived tuples of a single input stream with the all the tuples from the other stream.

The processing phase is broken into stages. Firstly, the newly arrived tuples for both streams are loaded to the corresponding processing elements (PEs). Next, the tuples from the S and the R streams, which are not outdated, are streamed to the corresponding PE. The PUs compare the two incoming tuples and if the comparison result is "successful", a new merged output tuple is created. The output information is kept into a FIFO at each PU, which is passed through a network of MUXes to the PE output. When all tuples are streamed and no other results have to be sent out, then the processing phase finishes. In case the newly arrived tuples are more than the available N PUs at each PE, the above process is repeated.

B. *Reconfigurable ScaleJoin System*

The presented reconfigurable architecture can exploit the high scalability and the performance advantages that hardware can offer, if it is mapped on a reconfigurable platform with fast data I/O links and a large number of available hardware resources. Hence, we mapped the proposed architecture on a Convey HC-2ex FPGA-based server.

Fig. 3 presents the total system architecture for the stream join processing. In our prototype platform, each one of the 4 available FPGA devices maps a ScaleJoin module, which has 256 PUs. We parallelized the problem by loading different newly arrived tuples into each one of the available ScaleJoin modules. Thus, our implemented system could process in parallel 1024 newly arrived tuples.

Initially, the newly arrived tuples are stored in shared memory by parallel threads. Next, the tuples are loaded from the RAM and they are streamed to the processing elements via FIFOs. Specifically, the Convey HC-2ex server has 16 parallel memory controllers, which were used for concurrent and parallel access to the stored data. The PUs are connected in a pipelined way, in order to make all the comparisons needed with the minimum amount of memory reads. Finally, each ScaleJoin module outputs the results into an output FIFO and then the results are stored to the global shared memory.

The proposed system can offer solution for even higher throughput rates of the incoming streams. Specifically, the high level of parallelism that hardware can offer and the high bandwidth data I/O links that our proposed platform offers, leads to the fact that the reconfigurable part can be reloaded with newly arrived tuples at the same rate-based portion of time, i.e. second. This reloading process can take place many times during the same rate-based time portion.

Concluding, this section presented the hardware-based architecture of the ScaleJoin algorithm. The parallel nature of the proposed architecture is based on three points: parallel processing of newly arrived tuples by the four available FGPAs, the parallel processing of the two independent streams by the two Processing Elements and the intrinsic parallelism that the hardware offers.

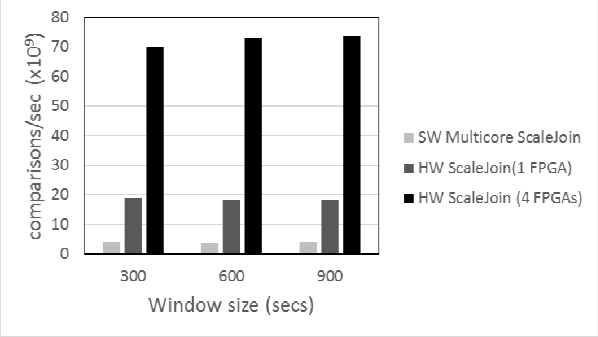

Fig. 4. Processing rate (comparisons/sec) for SW-based multicore ScaleJoin and FPGA-based solution ScaleJoin

V. System Evaluation

This section presents the performance results of the final system.

A. Theoretical Performance bounds

As analyzed in Section II, considering that the tuples from both streams arrive with a rate T tuples/sec and the time processing window has size W, then the total number of comparisons that need to take place at each second is 2 x W x T^2 [1].

B. Experimental setup

We tested and evaluated the proposed system with the same datasets that were used by previous works, i.e. ScaleJoin [1], Celljoin [8] and Handshake joins [5, 6]. Specifically, we used a C code that generated random tuples according to a uniform distribution in the interval [1−10,000]. The generated tuples were stored into Convey's RAM at each second. Next, the new tuples of R and S streams were loaded to the reconfigurable part of the ScaleJoin module, while the older tuples were streamed for processing. Last, the processor read and presented the stream join results at each second.

C. Performance Evaluation

As referred above, the reconfigurable architecture was mapped on a Convey HC-2ex server. The HC-2ex server is equipped with four Virtex 6 LX760 FPGA devices and a 4-core Intel Xeon CPU at 2.13 GHz with 24GB RAM. The resource

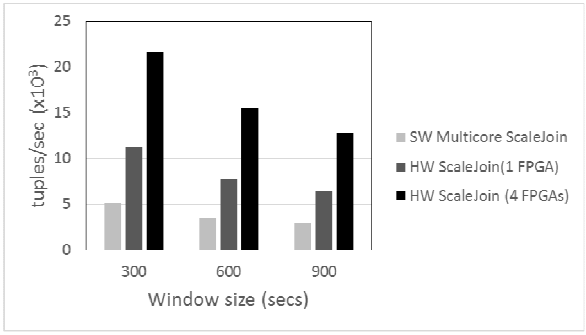

Fig. 5. Throughput (tuples/sec) for SW-based multicore ScaleJoin and FPGA-based solution ScaleJoin

utilization for the proposed architecture reaches up to 30% of the available resources (31% Slices, 15% BRAMs). The processing system is clocked at 80 MHz. On the other hand, the software-based reference system, as presented in [1], is equipped with a 2.6 GHz AMD Opteron 6230, 48 cores over 4 sockets and 64 GB RAM. Both systems' performance was measured using two metrics, i.e. the numbers of comparisons per second and the throughput rate that the proposed systems can achieve. The presented measurements are actual, experimental results from runs on the respective platforms.

Fig. 4 shows the processing capabilities of the proposed system. Specifically, we present the number of comparisons per second, which take place by the implemented systems for various input datasets. According to the performance results in [1], software-based reference system achieves approximately up to 4 billion comparisons/sec for various processing window sizes. On the other hand, the hardware-based system can offer up to 74 billion comparisons/sec. Hence, our proposed solution outperforms in terms of processing the best stream join multi-core solution by a factor of 19x.

Fig. 5 shows the throughput achieved in tuples per second for both systems. As the results indicate, the full reconfigurable system outperforms the fastest software-based multicore solution by at least a factor of 4x. Concluding, the performance results in Fig. 4 and 5 reveal the scalability of the hardware proposed solution.

D. Benchmark Performance Evaluation

This section compares the performance of the proposed stream join implementation vs. other state-of-the-art multicore solutions [1, 5, 6 and 8] under the same testing parameters, e.g. window size.

As the results in Table I show, the FPGA-based system seems to outperform any other previously proposed stream join solution by at least a factor of 2x as far as the processing rate. In addition, according to Table I, our proposed solution can outperform any other state-of-the-art multicore solution by at least one order of magnitude as far as the number of executed comparisons on streaming data including I/O time. Lastly, there are some previous works [4, 7] which map the stream join problem but they do not follow an open source benchmark to compare with. Thus, we could not compare directly the above works with ours either due to the different nature of the

TABLE I. SW MULTICORE STREAM JOIN VS. FPGA BASED STREAM JOIN ON BENCHMARK EVALUATION

Systems	Handshake system [5, 6]	ScaleJoin system [1]	CellJoin system [8]	FPGA-based ScaleJoin system
CPU Cores	40	48	9	1 CPU + 4 FPGAs
CPU type	2.2 GHz AMD Opteron	2.6 GHz AMD Opteron	1 PPE and 8 SPEs	2.13 GHz Intel Xeon
Max Throughput Rate (tuples/sec)	5125	3000	2000	12800
Max Processing Rate (Comps/sec)	1.5×10^9	4×10^9	-	74×10^9

performance results that they presented or due to the unavailability of the datasets that they used.

VI. RELATED WORK

This section presents previously software and hardware-based works that exist in literature for the stream join problem.

A. Software based implementations

The first software-based implementation on stream join problem was the Handshake algorithm [5, 6]. This algorithm can trivially be scaled up to handle large join windows, high throughput rates, and compute-intensive join predicates. Regarding processing throughput and latency, both of these implementations have significantly less processing throughput than our proposed solution, as shown above.

B. Hardware based implementations

There are many previous hardware-based works on stream join problem. To begin with, Halstead *et. al*. [3] introduces an FPGA-based implementation that uses a hash-join engine, achieving impressive performance results. Qian *et al*. presented M3Join in [4], which is a hardware solution that achieves high processing throughput rates. Authors in [5, 6] presented the reconfigurable implementations of the Handshake join algorithm using an adaptive merging network. In addition, Oge *et al*. [7] proposes a scalable and order-agnostic hardware design of sliding-window aggregation and its implementation on an FPGA. Last, Celljoin [8] is another implementation of window-based stream joins using a Cell processor. However, regarding its processing throughput both software implementations of Handshake join and ScaleJoin outperform this implementation.

VII. CONCLUSIONS

This work presented an FPGA-based system that implements a widely used stream data mining operator, i.e. stream join. To the best of our knowledge, this is the first work that maps a stream join operator on a high-end multi-FPGA system. We presented and analyzed an efficient, extensible, scalable and generic reconfigurable architecture for the stream join workload. The main characteristics of the proposed architecture are analyzed below: i) efficient as according to the performance evaluation the proposed architecture seems to outperform any other state-of-the-art published work, ii) extensible and scalable as the proposed architecture can be easily extended and offer high performance achievements and iii) generic as it can be easily expanded to tackle other stream-based workloads on reconfigurable hardware. Concluding, this work showed that FPGAs are particularly well suited for this form of computation vs. software-based multicore solutions, but fast I/O and the proper memory organization is necessary in order to fully realize these advantages.

ACKNOWLEDGMENT

This work was supported in part by the European Commission in the context of the H2020-FETHPC EXTRA project (No. 671653).

REFERENCES

[1] Gulisano, V., Nikolakopoulos, Y., Papatriantafilou, M., & Tsigas, P. ScaleJoin: a Deterministic, Disjoint--Parallel and Skew--Resilient Stream Join. In Big Data (Big Data), 2015 IEEE International Conference. IEEE.

[2] Kang, J., Naughton, J. F., & Viglas, S. D. (2003, March). Evaluating window joins over unbounded streams. In Data Engineering, 2003. Proceedings. 19th International Conference on (pp. 341-352). IEEE.

[3] Halstead, R. J., Sukhwani, B., Min, H., Thoennes, M., Dube, P., Asaad, S., & Iyer, B. (2013, April). Accelerating join operation for relational databases with FPGAs. In Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on (pp. 17-20). IEEE.

[4] Qian, J. B., Xu, H. B., DONG, Y. S., Liu, X. J., & Wang, Y. L. (2005). FPGA acceleration window joins over multiple data streams. Journal of Circuits, Systems, and Computers, 14(04), 813-830.

[5] Teubner, J., & Mueller, R. (2011, June). How soccer players would do stream joins. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 625-636). ACM.

[6] Roy, P., Teubner, J., & Gemulla, R. (2014). Low-latency handshake join.Proceedings of the VLDB Endowment, 7(9), 709-720.

[7] Oge, Y., Yoshimi, M., Miyoshi, T., Kawashima, H., Irie, H., & Yoshinaga, T. (2013, December). An Efficient and Scalable Implementation of Sliding-Window Aggregate Operator on FPGA. In Computing and Networking (CANDAR), 2013 First International Symposium on (pp. 112-121). IEEE.

[8] Gedik, B., Bordawekar, R. R., & Philip, S. Y. (2009). CellJoin: a parallel stream join operator for the cell processor. The VLDB journal, 18(2), 501-519.