www.qualiMaster.eu

QualiMaster

A configurable real-time Data Processing Infrastructure mastering autonomous Quality Adaptation

Grant Agreement No. 619525

Deliverable D3.3

Work-package	WP3: **Optimized Translation to Hardware**
Deliverable	D 3.3 : Hardware-based Data Processing Algorithms V2
Deliverable Leader	Telecommunication System Institute
Quality Assessor	H. Eichelberger
Estimation of PM spent	21
Dissemination level	Public (PU)
Delivery date in Annex I	31.7.2016
Actual delivery date	01.08.2016
Revisions	4
Status	Final
Keywords:	QualiMaster, Adaptive Pipeline, Reconfigurable Computing, FPGA Computing, Hardware, Support Vector Machines (SVM), Count Min, Exponential Histogram, Hayashi-Yoshida Correlation Estimator, Mutual Information, Transfer Entropy, design automation tool.

Disclaimer

List of Authors

Partner Acronym	Authors
TSI	E. Sotiriades, G. Chrysos, P. Malakonakis, I. Papaefstathiou, A. Dollas
MAX	G. Gaydadjiev, T. Becker, N. Voss

Table of Contents

Executive summary

We have achieved all project goals to date with the completion of all hardware modules which can be incorporated in the QualiMaster Pipeline. WP3 partners have achieved to have a set of architectures that are integrated in the QualiMaster framework that can work in several pipelines. These designs have been created with generalized design rules and patterns which are used toward the implementation of an automation design tool which reduce the design cycle time.
In this deliverable several results are presented:

- A second generation of architectures for hardware acceleration is substantially faster vs. the first generation.
- A new set of design rules is presented which are used for the design of the automation tool.
- The hardware designs which run on the reconfigurable Maxeler platform are ready to be integrated into the QualiMaster Pipeline. The integration processing part of WP5, however, it entailed substantial work on WP3-related architecture development. There was close collaboration with WP5 partners in order to address a large number of technical difficulties. The completion of the interface plays an important role in the QualiMaster pipeline. Although the interface itself does not have any exceptional characteristics in terms of performance, nonetheless, it plays a very significant role because it is the element which ensures seamless functionality among architectures which are disparate and fundamentally incompatible with one-another.
- WP3 staff from TSI visited the MAX headquarters for an extended stay (two weeks), in which the designs were ported to newer generation technologies, in order to assess portability of the designs and gather more data of how a QualiMaster Pipeline could be maintained in the long run in the future.
- Hardware modules are designed in a generic manner and several tests of them show that they and can be easily integrated to the QualiMaster platform.
- The WP3 review was successful; the reviewers' comments on issues to address were few, and the requested actions are quite manageable – we are in fact addressing all such issues and comments.
- Ever-increasing in size data sets have been used in the hardware designs and the QualiMaster pipeline itself, not only in order to address reviewer's comments that tests need to be made with realistic scenario, but also in accordance to 3rd year project goals for the QualiMaster project, in order to demonstrate the capabilities of the system.
- Related to WP3 but reported in WP7, we have published several of our results in international conferences and have submitted work under review in refereed journals, a process which will continue through the end of the project.

1. Introduction

In D 3.2 the QualiMaster partners had shown the 1st Generation of Algorithms implementations in Hardware following the design process that has been introduced in D3.1. In order to improve design performance, the design process had to be repeated towards the implementation of the 2nd Generation of Algorithms. Usually, in the 1st design cycle the designer focuses on the accurate functionality of the system. In the subsequent designs several aspects are considered which could not be taken into account in the initial design cycle, aiming at performance improvement of the design. Such aspects are Algorithm characteristics and how they can be efficiently mapped to the target technology, which can lead to an improved solution and on many instances even to a new architecture (design from scratch). Through the first implementation the designer completely understands the algorithm as well as the platform features, in every detail. Then he/she can use all the technology features towards improved performance.
In the QualiMaster designs, in which the algorithms and the QualiMaster pipeline platform are completely new for the designers, several changes have been done to implement the 2nd Generation of Algorithms, yielding substantially improved results.

1.1 Improved Algorithms and designs

For all of the designs, which have been implemented in D3.2, a 2nd Generation architecture has been designed, except for the Support Vector Machine (SVM) training algorithm, in which initial results proved to be lackluster due to Platform design restriction and I/O bounds.
The Hayashi - Yoshida correlation estimator has been improved and tested on different platforms, the new design offering larger capacity for real time calculations. The new design was integrated with the QualiMaster pipeline. By comparison to the initial design, the 2nd Generation offers 36% larger system capacity.
Count Min Sketch and its Exponential Histogram extension have been integrated in a new module by using the generic interface that has been developed for the QualiMaster pipeline. The new designs are 2 times faster vs. the first one and 13 times faster vs. the official, single thread software release (there is no official multithread or multicore release).
Two new algorithms that have been introduced to process social network and financial data, the Mutual Information estimator and the Transfer Entropy estimator were both redesigned and tested in new platforms, offering performance improvements of 2x on average.
In the design of the 2ndGeneration, new platforms where tested in addition to the architectural improvements, described above. The state of the art, for Maxeler, are the Maia platform and the ISCA platform with direct Ethernet connection, both of which have been also used to test the algorithms. These tests show that the hardware modules are designed in a generic manner and can be easily integrated to the QualiMaster platform.
Finally a set of design rules has been encoded as a guideline for the automation tool implementation. These rules are main considerations that a designer has to implement an efficient hardware module. In that sense these rules are not in strict mathematical form to implement but guidelines that an automation tool has to follow.

1.2 Interaction with other WPs

In this deliverable, as was described above, the QualiMaster partners were in close cooperation with WP5. All the designed modules had to be integrated in the complete system and work with the QualiMaster pipelines. Part of the design procedure was to create a generic interface between hardware modules and the QualiMaster platform over the Maxeler Server – Storm interface which has been described in the previous deliverables.
This interface consists of several modules which receive data out of order, process them in order and retransmit them to the nodes in which they are needed. This module can replace the respective software or work in parallel with it, in order to receive higher throughput.
Furthermore, the Hayashi Yoshida algorithm has used for evaluation and demos as part of the WP6.

Addressing D3.3 objectives

This section focuses on the presentation of the objectives for the deliverable D3.3, as they were described in the QualiMaster Description of Work (DoW). Table 1shows our actions on the D3.3 objectives and where in this deliverable more details can be found.

Tasks	**Objective**	**Specific actions undertaken according to D3.3 objectives**	**Sections where more details can be found**
T3.1	Identify and analyse algorithms that will be accelerated through hardware	Actions for this Task have been taken at D3.1 and 3.2	-
T3.2	Develop an initial translation of the proposed stream processing algorithms on reconfigurable technology, offering special-purpose hardware-based accelerators	Actions for this Task have been taken at D3.1 and 3.2	-
	Provide technical restrictions and the related tradeoffs of reconfigurable hardware with respect to the initial translation of the proposed algorithmic tasks	Continue from the D3.2 all the technical restrictions and tradeoffs are presented. Focusing in the integration of the created modules to the QualiMaster platform. Most effort has been made to study the I/O issues and resource utilization in order to achieve performance boosting. I/O issues include internal module interface (e.g. special-purpose processor with memory) and system-level interfaces as communication in the QualiMaster pipeline	Section 2.1 Integration with QualiMaster Pipeline Architectural tradeoffs at the description of each implementation Sections 3.1.5, 3.2.5, 3.3.5, 3.4.5
T3.3	Move from special-purpose translation towards a general mechanism for mapping stream processing algorithms on reconfigurable logic (MAXELER)	Encode a Set of Rules for design and Implement of 3 different hardware modules in different Maxeler platforms.	Sections 2.1 3.1, 3.2, 3.3, 3.4

	Employ rapid system prototyping for algorithm mapping, so that advantages/disadvantages of each architecture to be understood from actual runs	As mentioned in D3.2 "The algorithm mapping into hardware had tobe done iteratively, with working hardware designs in each case" For every one of the Hayashi Yoshida, ECM Sketch, Mutual Information and Transfer Entropy major changes have been made or even new architectures have been designed, achieving major performance improvement.	Sections 3.1, 3.2, 3.3, 3.4
	Partially reconfiguring the hardware to accommodate pipeline adaptation.	Partners have implemented a generic interface connecting algorithms mapped in hardware with the QualiMaster pipeline. This interface has been tested for several algorithms, in different modes,for large data sets.	Section 2.2
T3.4	Initial performance evaluation and testing results for the hardware-based solutions	Following the D3.2 the new architectures offer performance gains vs. Multithread software solutions. Tests were made with large real life data sets.	Sections 3.1.3, 3.2.3, 3.3.3, 3.4.3
	Create hardware tool which provides meaningful results in their own right	Start development of tools for visualization and partially automated optimization.	Section 4

Table 1: Addressing D3.3 Objectives

1.4 Addressing Reviewers' remarks

The reviewer's remarks from the 2nd year annual review were focusing on the hardware performance measurements, proposing comparisons against multithread software implementation and use of larger data sets. WP3 partners use the official distribution of software to compare with, and if possible do it with the distributor as in the ECM sketch algorithm. For the Hayashi Yoshida algorithm, comparisons are made with the corresponding multithreaded solution when mapped to distributed platform using the Storm framework. Official distribution of the Mutual Information algorithm is coded in the R-Project, which has inherently low performance. Transfer Entropy as was presented in [4] was implemented in C which proved not to be optimized. For that reason WP3 partners had to rewrite themselves the software in the C programming language, which proved to be 19 times faster than R-Project and 2.5 times faster than the original software for the Transfer Entropy. For this reason, our home-developed highly optimized software was used for performance comparisons.
Data sets that are used are real life data from stock exchange markets and were provided by Spring.

2. Algorithms and System Design

This deliverable presents the hardware-based components of the QualiMaster project and the way that these components are integrated into the final QualiMaster pipeline platform. This section describes the generic system level architecture of the hardware-based QualiMaster algorithms. Next, we present the way that the different hardware-based algorithms are combined under a common hardware-based wrapper. Last, we describe the way that the final infrastructure is integrated into the QualiMaster platform.

2.1 Encoding design rules

Design process for the implementation of Algorithm for the QualiMaster project has created a set of rules in order to build hardware modules with generic interfaces that can adapt in different platforms and achieve performance boosting.

a) Designer has to exploit all the inherent parallelism that an algorithm has. Parallelism can be in data level, processing of multiple data, or in processing level as the same data can be processed from different modules if needed or in time as different modules can process different data at the same time in the sense of hardware pipeline.
b) Designer has to implement a fast memory interface based on the platform technology as in most problems I/O and memory interconnections are the bottlenecks. Such interface has to take in advantage the highest possible memory bandwidth making batch memory reads and avoiding memory collisions. Following memory access patterns or scheduling memory access usually is the solution to such a problem.
c) Designer also has to take into account the available resources and design architectures that can fit to the available technology. If resources are not enough then a part of the processing has to be done in the general purpose processor or data has to be stored in external memory. Such solutions usually do not offer performance boosting.
d) As IO is usually considered as the bottleneck designer has to design architectures that have enough computing power to process all incoming data.

2.2 Integration with QualiMaster Pipelines

As described in D3.1 and in D3.2, the hardware algorithms are implemented as hardware libraries, which can be loaded and unloaded dynamically on a Maxeler server. The Maxeler platform is a general purpose, Linux-based server with a powerful FPGA-based coprocessor. Algorithms are mapped both on the software-based part of the server, i.e. CPU and the hardware-based part, i.e. Data Flow computing Engine DFE. The algorithms start their execution on the host processor. The hardware libraries, i.e. the part of the algorithm that has been mapped on DFEs, are loaded through a simple function call procedure. Next, the main processing steps take place. Initially, the data are passed to the DFE through specific interfaces, which are used for connecting the Maxeler CPU and the DFE devices. Then, the processing takes place into the DFE. Last, the hardware library produces the results and it sends them back to the processor. The algorithm execution continues from the point that the hardware part was called till the completion of the algorithmic workload. Figure 1 presents the way that we integrated each one of the implemented hardware-based algorithms into the final QualiMaster infrastructure.

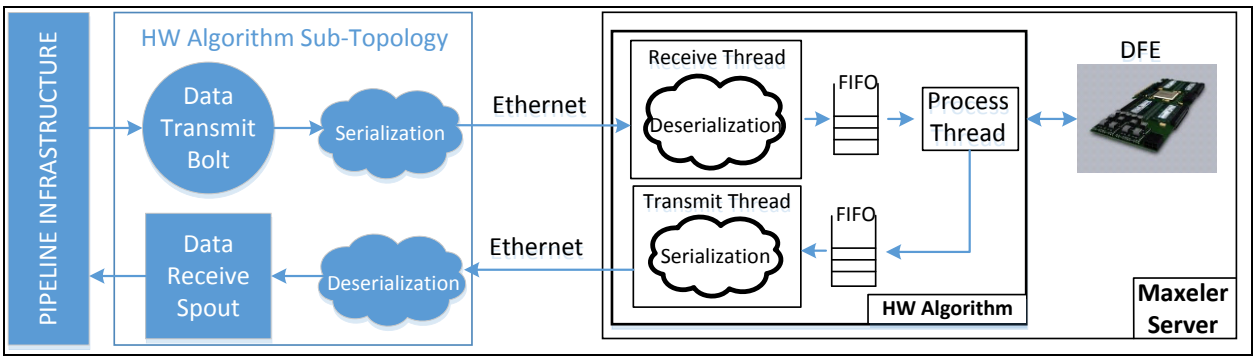

Figure 1: Integrating hardware-based algorithms into QualiMaster platform

First, we briefly describe the way that we used for mapping the algorithms on the hardware platform. Each algorithm is split into two parts: the software part, which is responsible for unpacking and packing the incoming and the outgoing data, respectively, and the hardware part, which implements the main computational part of the algorithm. As shown in Figure 1, the software part consists of three parallel running threads, i.e. the receive thread, the process thread and the transmit thread. Each one of them serves a different task and they communicate with each other through software-implemented FIFOs.

Next, we are going to describe the dataflow cycle for the hardware-based implementation they receive thread establishes a network connection with the "external" data source, i.e. data transmit bolt. Subsequently, it starts receiving data tuples in an infinite-loop. It unpacks the incoming tuples and passes the data through a FIFO to the process thread. The process thread updates the algorithm's internal data structures and makes function calls to the hardware module for data processing. In addition, the process thread collects the results from the reconfigurable part of the Maxeler server and passes them through another FIFO structure to the transmit thread. The transmit thread receives the processed results, it packs them into a specific format and forwards them through the network link to the data receive spout. As Figure 1 shows, the communication threads, i.e. the receive and the transmit threads, use the serialization and deserialization frameworks, respectively. As described in D5.3, we needed a consistent data serialization solution between software and hardware to provide an easy way of integrating the hardware based algorithms into the QualiMaster instantiation. Thus, we used the Google Protocol Buffers (protobuf) as a serialization solution. This solution offers us a flexible and platform-neutral way to serialize structured data. Specifically, it offers efficient, flexible and extensible ways for encoding (serializing) and decoding (de-serializing) structured data while it has easy language and platform interoperability.

Another important part of the implemented system that is presented in Figure 1 is the "software" part of the hardware-based module. The software part of the implementation, i.e. the Hardware Algorithm Topology, consists of two modules, the data transmit Bolt and the data receive Spout. The Bolt receives tuples from the Pipeline infrastructure and packs them using the serialization scheme. Next, it sends the serialized data via a network link to the Maxeler server. Concurrently, the Spout receives the results from the Maxeler server and deserializes them into tuples. These tuples are sent to the rest of the pipeline infrastructure using the Storm communication links. Last, it is important to mention that the code for the software part of the hardware-based module, i.e. the Hardware Algorithm Topology, is automatically produced by the QualiMaster tool chain.

Concluding, as described in D3.2 and in D5.3, the main drawback of the proposed hardware-based streaming framework is the restrictions on the data transmission rate due to some internal problems of Storm framework. Our main attempt till the end of the QualiMaster project will focus on increasing the data transmission rate between the Storm-based modules and the hardware-based mapping on the Maxeler server. The solution that we aim to try is to increase the number of the TCP ports, which are used for the communication between the hardware algorithm topology and the mapped hardware algorithm on the Maxeler server. We believe that this solution will offer much higher throughput rate and it will lead to even higher processing rates.

2.3 System Level Architecture

This section describes the integration of the above hardware-based system into the QualiMaster pipeline. Also, we present the way that the hardware based algorithms can be mapped and run in parallel on the Maxeler server.

As described above, a hardware-based algorithm can be loaded and unloaded dynamically on the Maxeler platform. We implemented a framework that communicates with the adaptation layer of the QualiMaster infrastructure and it supports all these procedures. Specifically, we created a software-based interface, which communicates with the adaptation layer through TCP Ethernet links. The adaptation layer is responsible for sending through this interface all the needed information about loading/unloading the corresponding hardware-based algorithms on the Maxeler server. The interface from the hardware-side sends all the needed information, e.g. the open TCP ports, which the pipeline infrastructure needs in order to create the "connection" between the HW algorithm topology and the Maxeler platform. In addition, the interface module on the Maxeler server is responsible for activating/deactivating the corresponding hardware-based algorithm and initializing all the parallel running threads, which were described in the previous section, for each algorithm. Figure 2 describes in a schematic way all the needed interconnections that are used for activating and deactivating a hardware algorithm.

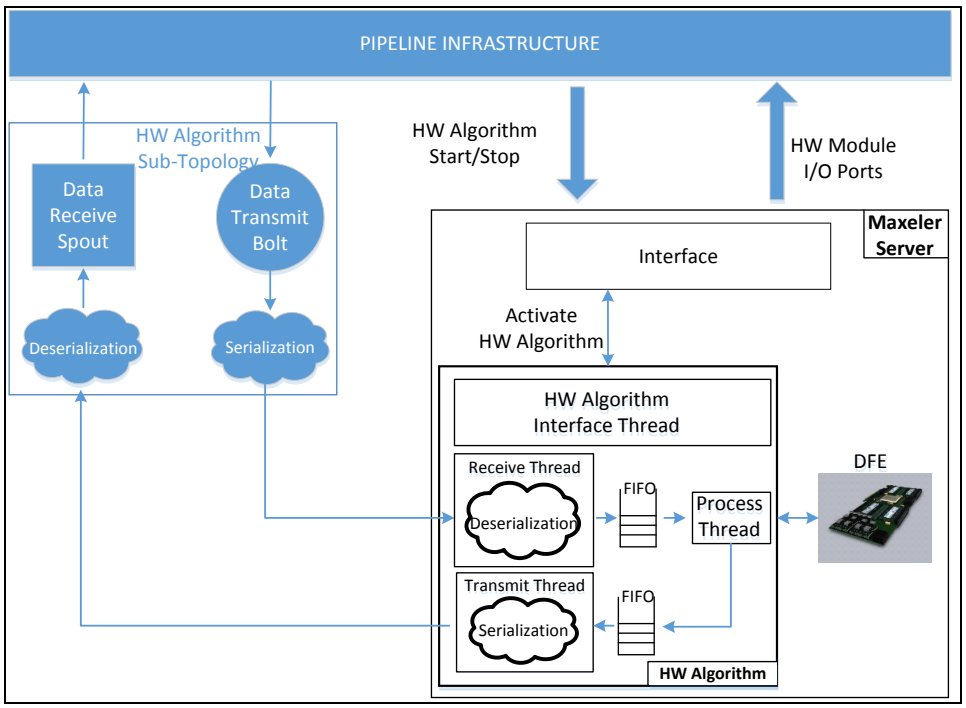

Figure 2: Steps for activating/deactivating a hardware-based algorithm

As each Maxeler platform consists of 4 DFEs (one DFE corresponds to one FPGA device, which is used as a dataflow engine), thus, it can host up to 4 different hardware-based modules. These hardware-based modules can map the same or different algorithms. Taking into account the above, we implemented a software-based framework that supports the concurrent loading of more than a single hardware-based module. Figure 3 shows the way that we managed to run in parallel more than a single hardware-based algorithm. Firstly, we extended the interface between the Maxeler server and the pipeline infrastructure. Specifically, in our initial implementation the interface accepted simple commands from the QualiMaster Infrastructure Adaptation Layer about loading and unloading algorithms on the hardware platform. In this newer version, the Adaptation Layer passes to the interface module the URL of the Maven hardware artifact, where the hardware-based implementation of an algorithm is stored. Next, the interface downloads the hardware-based implementation on the local Maxeler server and if there are available (unused) DFEs, then, the new hardware module is loaded to the available DFE slot. In case that there are no available DFEs, then an inactive hardware-based module needs to be replaced. It is important to note that the Adaptation Layer knows about

the number of allocated and free DFEs through the monitoring protocol (seeD3.2 and D5.3), so that replacement of hardware-based modules happens on purpose. In order to set a hardware-based running algorithm inactive, the adaptation layer needs to pass the corresponding "stop" command for the specific algorithm to the interface module. In case of loading a new algorithm, the interface answers to the load request with a set of TCP ports that will be used by the corresponding Hardware Algorithm Sub-topology to be connected to the Maxeler server. Each hardware mapping uses a different pair of TCP ports for receiving data and sending data from/to the pipeline infrastructure. The complete system was tested and we validated its correct functionality under real life experiments.

Last, the performance of the final hardware-based system is based on various parameters of the final QualiMaster infrastructure. Hence, the final performance evaluation for the hardware integration will be presented in the upcoming Deliverables of the WP5, i.e. D5.4.

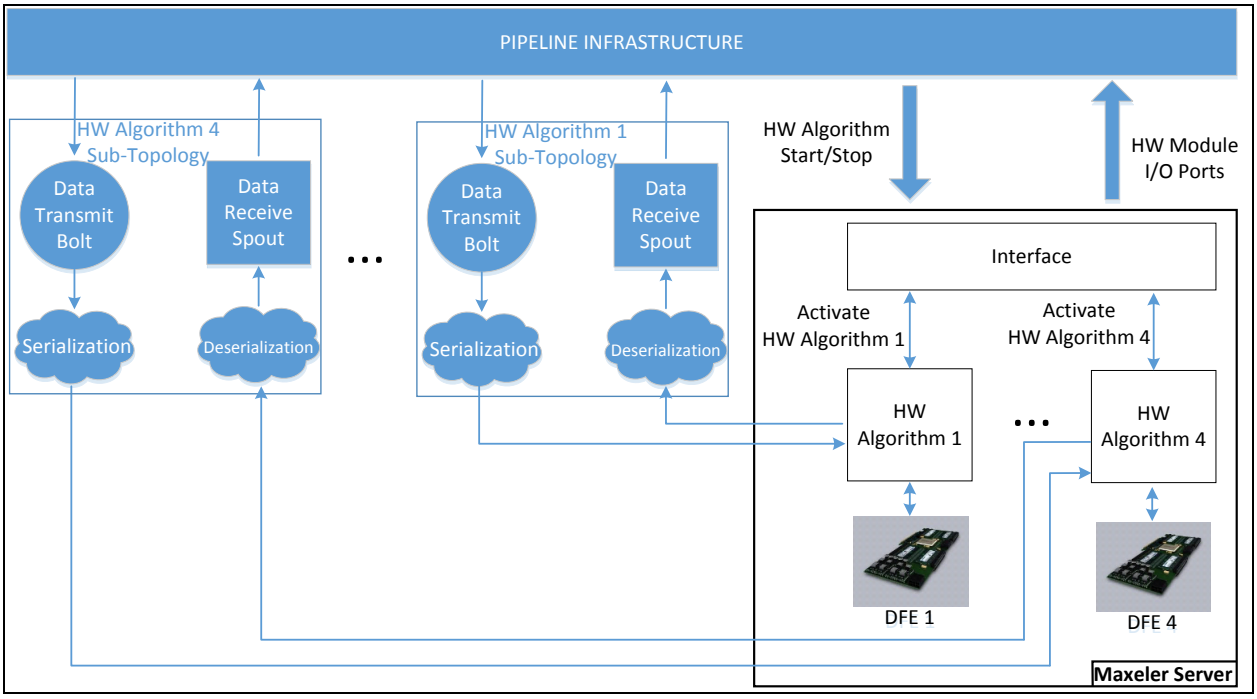

Figure 3: Loading in parallel more than a single hardware-based algorithm

3. 2nd Generation Design Architectures

First, this section introduces the new modern Maxeler cards that were used for mapping the proposed hardware-based architectures of the QualiMaster project algorithms. Next, we present the 2nd generation architectures for the hardware-based algorithms and their differences vs. the 1st generation proposed architectures. It is important to mention that the mapping and the performance evaluations, which are presented below in this section, took place after the close collaboration between the TSI and the Maxeler Technology company during the visit of two TSI fellows to Maxeler offices in London.
It is important to mention that we followed the same "pattern" for each one of the described algorithms in this section. Specifically, we, first, present the initial versions of the mapped algorithms as they were described in the previous deliverables. Second, we move to the new architectural modifications and the ways that we used for verifying these modifications. Next, we present the performance evaluation of the proposed architecture vs. the best available software solutions and the previous hardware-based implementations. Lastly, we sum up with the advantages and the disadvantages of each one of the proposed hardware-based architectures.

3.1 Hardware Platforms

As referred in D3.2, our initial hardware-based platform was a Maxeler MPC-C (Vectis) platform with four Xilinx Virtex 6 FPGA devices, 24 CPUs @1.6 MHz and 64 GBs RAM. One of our main concerns was to move towards more modern platforms with higher capabilities and evaluate the performance of the proposed hardware-based architectures. Hence, we mapped our proposed implementations on the modern cards, i.e. Maia and ISCA cards, of Maxeler technologies.
Maxeler MPC-X (Maia) platform consists of 8 Altera Stratix V FPGA devices and 48 GBs RAM. The Maia card has the same maximum PCIe bandwidth of 2GB/s and the same 24 GBs internal memory (LMEM) as the Vectis platform. The increased resources, provided by the Altera FPGA, allows the implementation of up to 6 parallel memory controllers, which improve the LMEM bandwidth vs. the throughput achieved by the Vectis card.
The Maxeler MPC-N (ISCA) platform provides a 10 Gbit Ethernet link connected directly to the FPGA. The FPGA devices on the ISCA cardsare the same as the Maia ones, i.e. Altera Stratix V, while it has 12GB internal RAM (LMEM). The main advantage of the ISCA cards is that they offer the lowest possible latencies for data transfer from/to the DFE while supporting full line-rate processing.

3.2 ECM-sketches

One of the main goals of the QualiMaster project is the processing of continuous high-volume streams of data in real time. As described in Deliverable D3.1 and D3.2, there are data structures, which could be used for building sketch synopses of the QualiMaster pipeline input streams in order to provide approximate answers with quality guarantees minimizing the storage space and the update time. The ECM sketch [1] is such a compact structure that combines a state-of-the-art sketching technique for data stream summarization, i.e. the Count-Min data structure, with deterministic sliding window synopses, i.e. the Exponential Histograms. Below we present the two proposed architectures for the hardware-based ECM sketch data structure.

3.2.1 ECM-sketch algorithm

First, we are going to briefly describe the ECM-sketch algorithm. The ECM-sketch data structure combines the Count-Min sketch structure [2], which is used for conventional streams, with a state-of-the-art tool for sliding-window statistics, i.e. the Exponential Histograms [3].The input of the ECM-sketch data structure is a number of data streams. The output of the ECM-sketch algorithm is a sliding window sketch synopsis that can provide provable, guaranteed error performance for queries, and can be employed to address a broad range of problems**.** As mentioned above, the ECM-sketch, Figure 4, combines the functionalities of Count-Min sketches and exponential histograms. Specifically, the corestructure for the ECM-sketch algorithm is a modified Count-Min sketch. As Count-Min sketches alone cannot handle the sliding window requirement, thus, ECM-sketches replace the Count-Min counters with sliding window structures, i.e. Exponential

Histograms. Each counter is implemented as an exponential histogram, covering the last N time units, or the last N arrivals, depending on whether we need time-based or count-based sliding windows. In more details, adding an item x to the ECM structure is similar to the case of the standard Count-Min sketches.First, the EHs that need to be updated according to the corresponding to the d hash functions are detected. For each one of the EH, we register the arrival of the item, and remove all expired information, i.e. the buckets of the exponential histogram that have no overlap with the sliding window range.

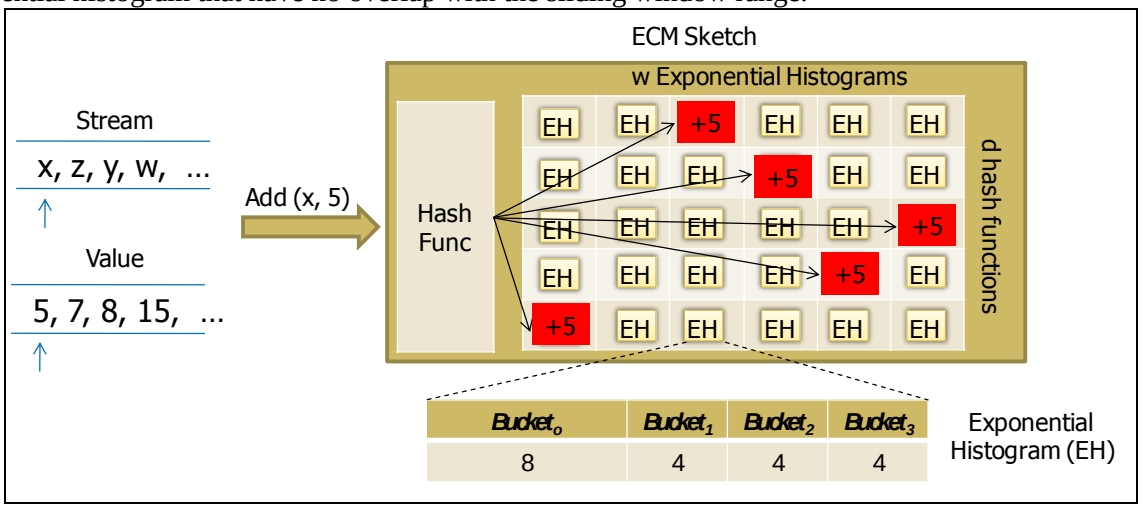

Figure 4: Example of adding a new element (x, 5) on an ECM sketch data structure.

3.2.2 1st HW-based ECM sketch Architecture

This section describes briefly the 1st hardware-based architecture of the ECM sketch data structure that was presented in Deliverable D3.2. The proposed system, Figure 5, was divided into two functional parts: the hash function and a module that combines the Count-Min module with the functionality of Exponential Histogram structure. In this first implementation the hash function is implemented in software while the ECM-sketch data structure on reconfigurable hardware.

The system takes as input streaming tuples with format (*Value, Timestamp*). Each time a new element arrives, the *Value* enters the software-based Hash function and a new tuple with format (*EH_ID, Timestamp*) comes out. The *EH_ID* value is used for assigning the input element to a single Exponential Histogram data structure at each row of the ECM sketch. Next, the new created tuple is passed to the main ECM data structure for updating it.

As shown in Figure 4, only one Exponential Histogram data structure from those that belong to the same row of an ECM-sketch is updated at each new tuple arrival. Thus, we grouped each row of the ECM sketch data structure into a single compact hardware-based Exponential Histogram data structure. This hardware-based structure implemented the basic functionality of an EH data structure. Specifically, it consisted of pipelined buckets, i.e. bucket level 0, bucket level 1, and each one of them consisted of connected elements, i.e. the internal bucket elements. These internal bucket elements were connected to a memory module, i.e. W counters, which kept the stored information for the same position of all the W Exponential Histograms for a single ECM-sketch row. This proposed architecture offered a high level fine-grained parallelization. Each mapped bucket level could process independently a different update request at each clock cycle due to the pipeline technique. For example, each time a new element arrived at the first bucket of the data structure, the update process took place to that level. Concurrently, the second bucket level module could update another EH, or even the second level of the same EH, based on a merge request that arrived from the previous bucket level during the previous clock cycle. This process could take place concurrently in all the mapped buckets of the ECM data structure. The same procedure took place in case of querying the ECM sketch data structure. Taking into consideration all the above, we concluded that the complexity of updating or querying the ECM sketch with the proposed ECM hardware-based architecture was O(1). It is important to mention that the proposed architecture could easily exploit the coarse grained parallelization of the problem, too, by processing each row of the EH independently.

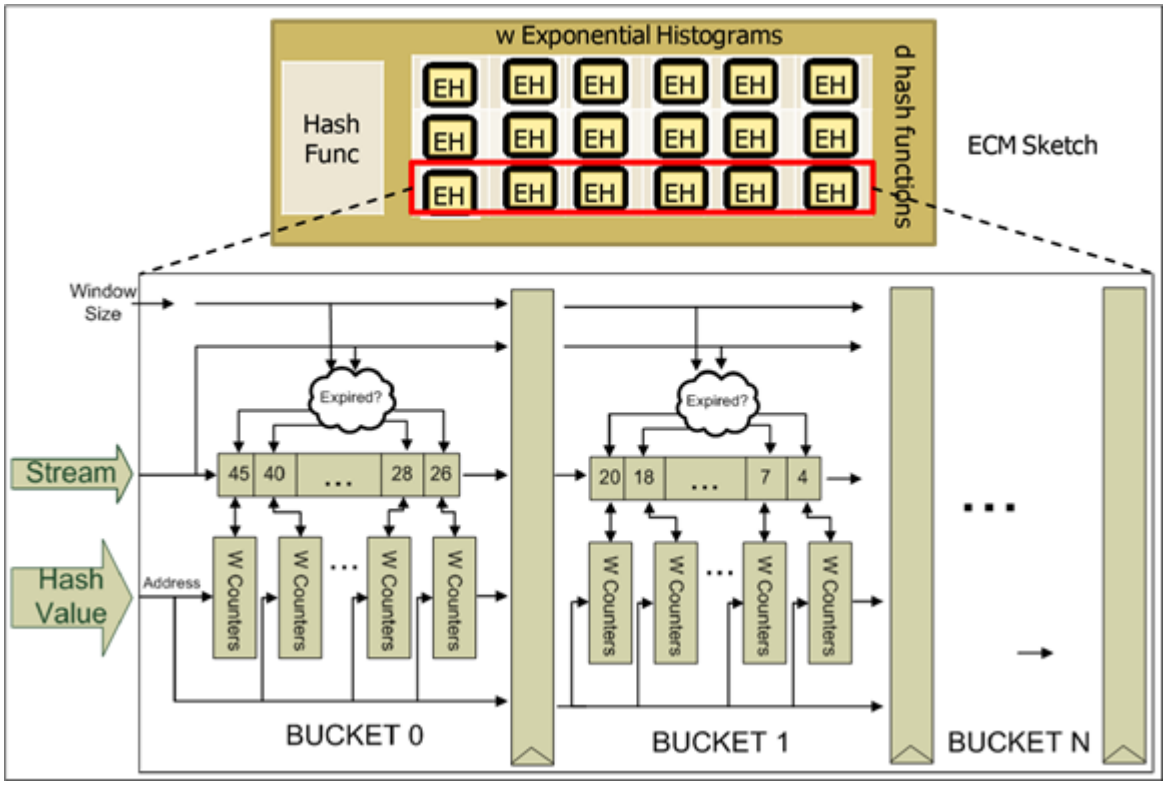

Figure 5: Initial hardware-based architecture for mapping ECM on reconfigurable platform

3.2.3 Architectural Modifications

In the second version of the hardware-based ECM architecture, we tried to minimize some of the disadvantages of the first proposed version. As presented in D3.2, the main drawback of the first hardware-based ECM architecture was the high resource utilization and the unstable workload balancing, which lead to inactive bucket modules for most of the processing time. This newly proposed architecture, Figure 6, makes a better use of the FPGA resources and it optimizes the way that the available resources are used. In more details, first, this proposed architecture increases the parallelization degree by implementing independent bucket modules that can independently update the EH data structures of an ECM sketch data structure. Second, this proposed architecture maps each EH of the ECM sketch on a single bucket module, which can be used for updating either pending or new incoming merges from the first bucket module. This solution offers a better workload balance for the mapped modules, as each bucket module can process the pending merges, thus it does not stay for long time inactive. All the above modifications lead to better performance results, as we will show in the next section.

Next, we are going to describe our newly proposed hardware-based architecture of the ECM sketch implementation. The proposed architecture is divided into two architectural levels, as Figure 6shows. The first part, i.e. Bucket 0, maps the bucket 0 of the ECM data structure. This bucket operates in the same way as the bucket modules of the first architecture. Specifically, this bucket is used for updates of the newly arrived tuples. This module keeps the information from the first bucket of the mapped EHs of a single ECM sketch row. Thus, each time a new tuple comes in, the data from the corresponding EH are loaded to the internal processing modules and they are updated. If the update process leads to a merge, then the information for the new merged tuple is passed to the second part of the proposed architecture through a demux module.

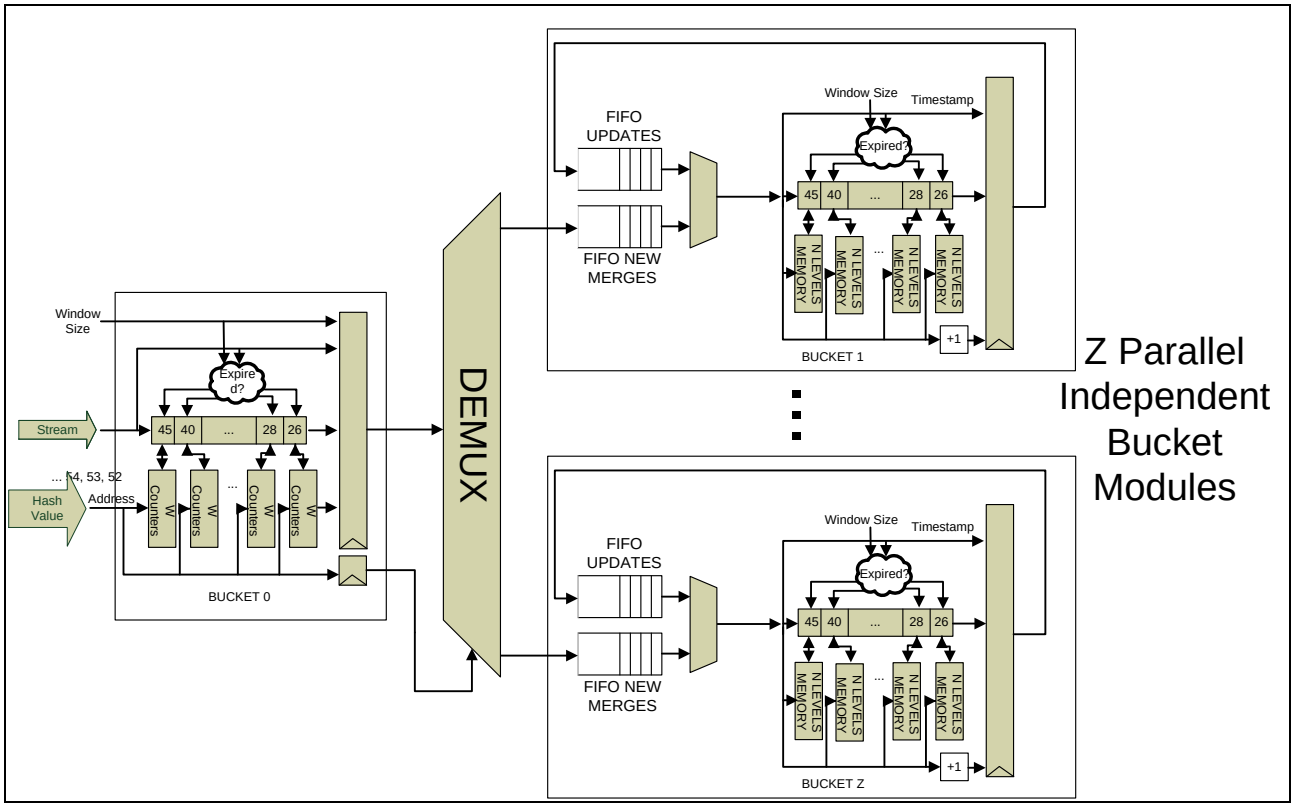

Figure 6: Second hardware-based architecture for mapping a single line of ECM sketch on reconfigurable platform

The second part of the proposed architecture consists of many parallel and independent modules, which map an EH of a single ECM sketch line. These modules consist of two different FIFOs and a bucket module. At this point, it is important to mention that the bucket modules of this second part are based on a different architecture from those of the first part. Specifically, each bucket module keeps internally the information of a complete EH with the processing element of the first architecture. In more details, each internal processing element is connected to a BRAM module, which has N positions, i.e. where N is the maximum number of the EH levels. Hence, each time a new update at a level needs to take place, e.g. update at level H, firstly, the values from the H position are loaded to all the processing elements. Next the update takes place and, lastly, the new values are stored again at the position H. Thus, this proposed bucket module can serve all the possible updates that could take place into a single EH. This novelty offered better resource use just by rearranging the way that data are stored and used by the bucket modules. Also, as described above there are two different FIFOs, which are used for the two different types of updates that can take place in this proposed architecture. The first FIFO, i.e. new merge FIFO, keeps the updates that arrive from the update module from the first part. The second FIFO, i.e. update FIFO, stores the pending merges that are created from the other levels of the specific EH. Thus, if the update process from a new element at a specific level of EH causes a merge, then the next level update will be stored to another FIFO, i.e. update FIFO. The proposed architecture offers great low level parallelism as it can process a new update at each clock cycle (either a new or a pending one from previous merges).

3.2.4 Verification

We tested our system with various input datasets and different configurations, i.e. the number and the size of buckets. We queried our system for the estimation of different timestamps and in different times and the results were verified vs. the results of the official Java code of the ECM sketch [1].

3.2.5 Performance Evaluation

This section presents the performance evaluation of both the first and the second hardware-based ECM sketch architectures. Also, we compare the performance of the hardware-based implementations vs. the official software solution.

As described in D3.2 and in the previous section, the 1st and the 2nd hardware-based architecture of the ECM sketch can be easily extended into parallel independent modules that map different lines of the ECM sketch into the available DFE devices of a Maxeler server. Such solutions increase the coarse grain parallelization that hardware can offer. On the other hand, our first experimental evaluation presents the performance achieved by hardware and software solutions when a single row of the ECM sketch is mapped on a single DFE device. Hence, we show only the advantages of the fine grained parallelization that reconfigurable logic can offer. The performance advantages when the complete ECM sketch data structure is mapped on reconfigurable logic, will be presented in our final report in Deliverable D3.4.

Both the software and the hardware implementations used the same parameter values in order to build a single row ECM sketch data structure. First, we needed to define the size of the ECM-sketch parameters, i.e. w and d. These parameters are calculated by two factors: ε and δ, where δ is the probability for answering a query over the ECM sketch within an error factor ε. The typical values for both ε and δ are usually in the range of [0.05, 0.2]. Also, each EH data structure consists of increasing sized non-overlapping buckets with sizes k+1 for the first one and $k/2 + 1$ for the rest of them, where k is bounded by the $1/\varepsilon$ value. We used the values $\delta = 0.4$, $\varepsilon = 0.05$ and window size = 2000000 for the evaluation of all the software and hardware implementations. At this point, it is important to mention that both our proposed architectures are fully parameterizable, which means that we can change the dimensions of the mapped ECM sketch according to the needs of the user application.

Dataset	#Events	SW Implementation	1st HW-based ECM		2st HW-based ECM		
		Update Rate (#Elements/sec)	Update Rate (#Elements/ sec)	Update Rate Increase vs. SW	Update Rate (#Elements/ sec)	Update Rate Increase vs. SW	Update Rate Increase vs. 1st HW Arch.
Random_1	10^8	10602205	75707894	7.1 x	145056187	13.7 x	1.9 x
Random_2	10^8	10770059	73426011	6.8 x	147247661	13.7 x	2.0 x
SNMP	$3.1*10^7$	11385588	69209585	6.1 x	140958243	12.4 x	2.0 x
IPS	10^8	10214505	76431383	7.5 x	147877804	14.5 x	1.9 x
WC	10^8	12163970	75565332	6.2 x	147006750	12.1 x	1.9 x

Table 2 Performance results for updating the software and hardware implementations of ECM data structure

The implemented systems were evaluated with a set of extensive experiments, using large real world and synthetic datasets. Specifically, we used two random datasets and three real life datasets: the world-cup'98 (WC) dataset, the Crawdad SNMP Fall 03/04 (SNMP) dataset and the IPS dataset. The WC data set consists of all HTTP requests that were directed within a period of 92 days to the web-servers hosting the official world-cup 1998 website. The SNMP data set contains a total of 134 million records collected from the wireless network of Dartmouth college during the fall of 2003/2003. The IPS dataset consists of Internet

traces collected by passive monitors according to the popularity of specific URL pages. The hardware-based ECM architectures were both mapped on a Maxeler Vectis server using a single FPGA device. On the other hand, the software solution was evaluated on a server with a Xeon processor @2.67 GHz with 50 GB RAM.
Table 2 presents the performance results as far as the processing streaming throughput rates achieved and the increase in update rates per second that hardware implementations vs. official single thread software solution can offer. We evaluated our proposed hardware solution vs. the best optimized solution of the software implementation. It is important to highlight that the performance for the software-based implementation of the ECM sketch in our tests was based on a single thread implementation, as we implemented an ECM data structure with a single row. Hence, such test case would lead to inefficient performance solution when mapped using multi-thread frameworks due to the fact that such tests do not offer coarse grain parallelization that software solutions can take advantage of. The performance comparison between the final hardware-based implementation and a multi-thread solution of a complete multi-row ECM-sketch data structure will be presented in the final evaluation in D3.4.
As Table 2 shows, the 1st version of the hardware based ECM sketch data structure can offer 7 times faster stream processing throughput than the official software solution. On the other hand, the 2nd version of the hardware-based ECM sketch can offer about 13 times faster stream processing than software and it is up to 2 times faster than the previous hardware solution. The results verify the high performance that the hardware-based architectures can offer. Also, the results indicate the advantages of the second hardware-based solution vs. the firstly proposed one.

3.2.6 Architectural tradeoffs

This section presents the advantages and the disadvantages of each one of the two previously presented hardware-based ECM sketch architectures.
The first hardware-based ECM architecture showed some important advantages that reconfigurable computing can offer.

- Our proposed architecture took advantage of the algorithmic steps and proposed a compact solution, where all the EHs of a single ECM-sketch row could use the same resources for the updating or querying process.
- In addition, the update and the query complexity that it offers is stable at O(1) as each update or query can be done in a single clock cycle vs. software solution, where the update complexity is amortized O(1).
- Last, the proposed architecture offered really high fine grained parallelization due to the capability of processing up to N independent updates at each clock cycle, i.e. where N is the number of ECM sketch bucket levels.

On the other hand, although this solution offered good resource reusability, due to the algorithmic nature many buckets stayed inactive for most of the processing time, which lead to bad workload balance of the resources. Hence, we moved to our 2ndhardware-based solution architecture that we described above.
The second proposed architecture has two main advantages vs. the previously presented one.

- Firstly, our second hardware-based solution offers a better resource utilization as it maps many parallel bucket modules, which can process ECM sketch updates in a completely independent and parallel way.
- Also, the workload balancing is optimized as the resources for the bucket modules are used continuously either for the newly arriving updates or the pending updates.
- Last, this second architecture combines the advantages of the previous proposed architecture offering a high performance implementation of the ECM-sketch data structure

The main drawback of this second proposed hardware-based architecture is the high resource utilization and the parallelization degree that can be achieved in a single DFE. Hence, in our final proposed solution we are going to use a multi-DFE platform for covering all the different test cases. Concluding, it is important to mention that both the proposed solutions exploit the fine grained parallelization that the hardware can offer by processing a different update/query at each clock cycle. Also, the proposed architectures can exploit the coarse grained parallelization, which, of course, can be easily exploited by multi-threaded software solutions, by processing each row of the ECM-sketch independently**.**

Last, we are going to describe some of the future plans for the ECM sketch hardware-based architecture that are going to present in D3.4 at the end of the QualiMaster project. First, we will try to increase the coarse-grain parallelism and the throughput rate of the complete system by mapping the processing of the hash function, which takes place in software, on hardware resources. Such solution will offer better performance results, as a single tuple transmission could be used for updating concurrently all the mapped rows of the hardware-based ECM-sketch. Second, we will focus on expanding the ECM sketch on the multi-FPGA platform for supporting the processing of the complete multi-row ECM-sketch software-based solution. This solution will show the performance advantages that hardware can offer in system level vs. the best available multi-threaded software solution.

3.3 Hayashi-Yoshida Correlation

The QualiMaster project focuses on the analysis of streaming financial data and the implementation of methods for event detection. The correlation estimator is a basic tool for financial risk modeling and it can be applied directly on series of stock markets transactions. As described in Deliverable D3.2, we implemented a first hardware-based version of the well-known Hayashi-Yoshida correlation algorithm on reconfigurable logic, which is briefly presented below. Our first proposed architecture was mapped on a Maxeler MPC-C (Vectis) node, while the performance evaluation of this system was presented in D3.2. This deliverable presents two new mappings of the proposed HY architecture on two more modern Maxeler platforms, i.e. the Maxeler MPC-X (Maia) node and the Maxeler MPC-N (ISCA) node. We, also, present some performance evaluation results from the three different hardware based systems and their comparison with the parallel multi-threaded software-based solution of the algorithm.

3.3.1 HW-based Hayashi-Yoshida Correlation Architecture

This section describes the proposed hardware-based implementation of the HY algorithm. As described in D3.2, the proposed architecture is a software-hardware co-design implementation. The software part receives the streaming input data, it updates the internal data structures and it streams out the results, while all the computations take place on hardware part.

In more detail, the software part consists of three parallel running threads, as shown in Figure 7. First, the Receive thread establishes the connection with the data source. As described in Section 2, the data arrive from a Storm Bolt node in a streaming way. Next, the Receive thread starts an infinite loop, where the newly arrived data are parsed and they are passed through a dedicated FIFO to the Process thread. This thread updates the internal algorithm's data structures and at each timestamp, i.e. 1 second, it function calls the hardware implementation, which is mapped on a DFE. Next, the processing results are streamed out from the Process thread to the Transmit thread via another FIFO. Last, the Transmit thread packs the results and it outputs them to the receiving result sink.

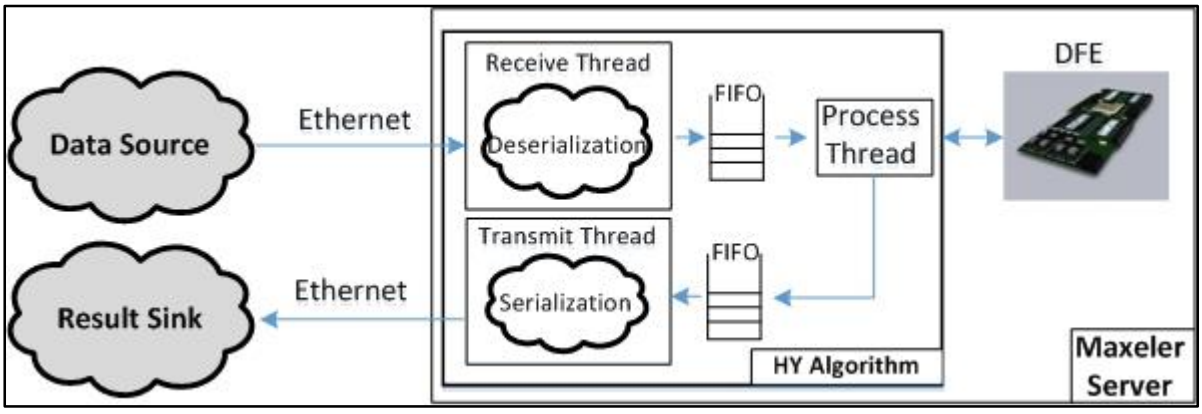

Figure 7: Second hardware-based architecture for mapping a single line of ECM sketch on reconfigurable platform

On the other hand, the main computational part of the algorithm is mapped on reconfigurable logic. Figure 8 presents the proposed architecture, which was analytically described in D3.2. The hardware-based HY

Coefficient Estimator module calculates the coefficients in an additive way. It uses two smaller modules that calculate independently and concurrently the correlation coefficients at the start and at the end of the processing sliding time window. Next, the correlation coefficient from the starting point of the processing window is subtracted from the correlation coefficient of the expiring point of the processing window. The result of the subtraction is added to the correlation coefficient from the previous timestamp. The final result is the new correlation coefficient for the pair of the processing stock markets. Last, the new correlation value is sent to the global shared memory and from there it is streamed to the Transmit thread. The proposed hardware implementation was mapped on a Maxeler MPC-C (Vectis) node.

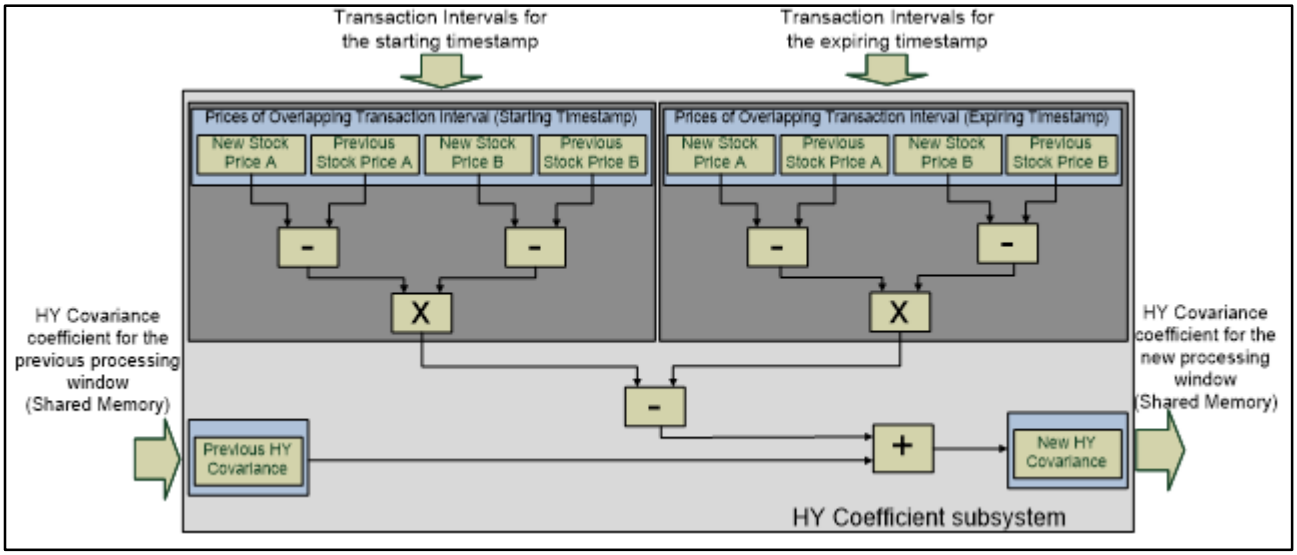

Figure 8: Hardware architecture of mapping the Hayashi-Yoshida Correlation algorithm

3.3.2 Architecture Modifications

As one of the QualiMaster project goals is to process high speed data streams, we moved towards the implementation of the 2nd generation systems that could offer much better performance results. These systems use the same reconfigurable architecture for the Hayashi-Yoshida algorithm, as presented in Figure 8, but they attempt to take advantage of some important characteristics that new and modern Maxeler platforms can offer. Firstly, we moved towards the use of a Maxeler MPC-X server focusing on its technology for faster data transfer between the CPU and the DFEs. Next, we used a Maxeler MPC-N node that offers high speed interconnection between the DFEs and outside world as it fully supports network interfaces with ultra-low network protocols and low-latency PCI-Express data transfer from/to the CPUs.

a) 1st modification

In this 1st architectural modification, we mapped our initially proposed architecture of the HY correlation algorithm on a Maxeler MPC-X (Maia) node. This node offers the new Remote Direct Memory Access (RDMA) technology, which provides direct transfers from CPU node memory to dataflow engines without inefficient memory copies. In addition, this platform offers infiniband connection between the CPU and the DFE module. The proposed architecture was mapped and evaluated showing a performance increase on computational part of the system around 20% vs. the previous proposed system.

b) 2nd modification

Next, we mapped the hardware Hayashi-Yoshida implementation on a modern Maxeler MPC-N (ISCA) node. This work focused on taking advantage of the ultra-low latency line-rate processing of multiple 10 Gbit data streams that a Maxeler MPC-N node can offer. As the MPC-N node has a different architecture from those of MPC-C and MPC-X servers, we needed to make some important modifications on the mapped architecture. First, we tried to take advantage of the high rate data transfer UDP links that the MPC-N server offers. This solution required a hardware-based architecture that would receive data from two different and independent "sources", i.e. the DFE's internal LMEM memory and the UDP link. Unfortunately, we did not manage to synchronize the data transfers from the UDP network links with those from the LMEM memory. Thus, we chose to abolish the use of the LMEM and move towards a solution where only the 10 Gbit UDP

network data streams were used for transferring data and results from/to the DFE. As Figure 9 shows, all the needed information from the CPU is now passed to the DFE through a 10 Gbit UDP link. This solution changed the work mapped on the Process thread, which now needed to pack and unpack the data into/from UDP packets. These packets are passed to the DFE device, where they are unpacked and processed. Last, the results return to the CPU, again, through UDP links.

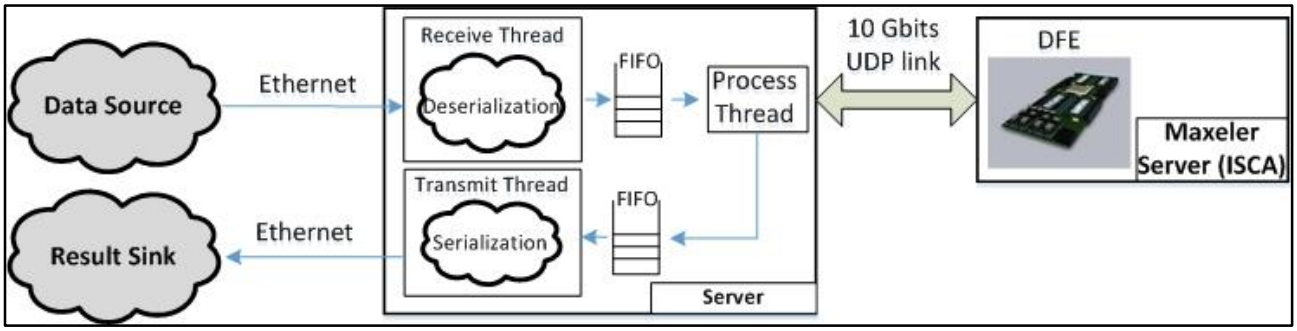

Figure 9: HW-based Hayashi-Yoshida Correlation implementation on a Maxeler MPC-N (ISCA) node

3.3.3 Verification

We tested our system with various input datasets, i.e. number of market stocks. The results of the hardware-based system were verified vs. a reference software-based implementation and the corresponding multi-thread implementation that was mapped on a TSI cluster.

3.3.4 Performance Evaluation

This section presents the performance evaluation of all the presented new hardware-based systems that map the Hayashi-Yoshida correlation algorithm.

First, we present the performance achieved by the two new hardware-based Hayashi-Yoshida implementations vs. the initial hardware-based proposed system. As referred in D3.2, the initial hardware-based implementation of the Hayashi-Yoshida algorithm was mapped on a Maxeler MPC-C platform with four Virtex 6 FPGA devices, 24 CPUs @1.6 MHz and 64 GBs RAM. On the other hand, the first newly proposed system was mapped on a Maxeler MPC-X (Maia) platform with 8 Altera Stratix V FPGA devices and 48 GBs RAM, while the second generation system was mapped on a Maxeler MPC-N platform with 4 FPGA devices and 40 Gbit Ethernet connections. For our performance evaluation all the implementations used a single FPGA device with low resource utilization as described in Deliverable D3.2.

The implemented systems were tested with synthetic and real life input datasets. As the proposed implementations follow the streaming formulation, the size of the processing time window is irrelevant to the performance of the implemented systems, thus we used time window size with 1 hour, i.e. 3600 seconds and advance 1 sec, i.e. the smallest possible advance time. Then, we evaluated the main execution time, i.e., the processing and the I/O time, for the reconfigurable part of the hybrid platforms for various input size datasets.

# Stock Markets	Processing time per timestamp (timestamp = 1 second)		
	Maxeler MPC-C (Vectis) System Evaluation (msec)	Maxeler MPC-X (Maia) System Evaluation (msec)	Maxeler MPC-N (ISCA) System Evaluation (msec)
100	1.9	0.5	0.67
500	2.3	1.5	8.5
1000	9.0	6.0	34.0
5000	224.0	165.0	874.0

Table 3: Performance evaluation of hardware-based Hayashi-Yoshida implementation mapped on different platforms

As Table 3 shows, the mapping of the architecture on a platform with high bandwidth links between the DFE module and the external memory can offer a higher processing rate up to 1.5x vs. our initial hardware based version. In addition, the performance results from the Maxeler MPC-N indicate that passing all the data through network and abolish the LMEM will not offer any performance advantages to the final system. Concluding, it is obvious that the Maia reconfigurable system using a single FPGA device and a single CPU node for I/O issues can compute the correlation metric for all the pairs of over 5000 stock markets in less than 1 second including the hardware I/O time overhead. In more details, according to the presented real life performance results the Maia system can process up to 6000 stock market players/sec and it seems to offer much higher processing power than the distributed Storm-based solution, which according to the presented results in D5.3 and D3.2 a cluster with 9 nodes, i.e. Dual-core AMD CPU @ 2.1 GHz and 8 GB RAM, can process up to less than 1000 stock market players per second.

3.3.5 Architectural Tradeoffs

This section sums up the advantages and the disadvantages of the proposed hardware-based architectures. As shown above, the mapping of the Hayashi-Yoshida algorithm on a Maxeler Maia platform offers the best achieved performance. This is due to the internal structure of the Maia platform between the LMEM and the reconfigurable, which uses a larger number of memory controllers than other platforms.

On the other hand, the mapping of the proposed architecture on a Maxeler MPC-N (ISCA) platform seems to decrease the overall system performance. This is mainly due to the fact that we did not manage to synchronize the LMEM module with the UDP links, thus we used only the UDP links for transferring data in and the results out from the DFE. This procedure created a great overhead as a huge amount of data needs to be transferred over network links.

Concluding, we analyze the performance of the Hayashi-Yoshida system when we attempted to run it over the full QualiMaster pipeline. As described and in deliverable D3.2 and D5.3, the main drawback between the performance achieved from the hardware parts when they are connected to the pipeline is the I/O restrictions, which take place between the mapped Storm components, i.e. Bolt and Spout, and external data sources when they are connected via TCP links.. Until the end of the project we are going to focus mainly on this problem and we will work on increasing the data transmission rates from and to the hardware platform using multiple parallel data I/O links.

3.4 Mutual Information

As mentioned in Deliverable 3.2, Mutual information I(X;Y) computes the amount of information a random variable includes about another random variable, or in terms of entropy it is the decrease of uncertainty in a

random variable due to existing knowledge about the other. More formally, for a pair of discrete random variables X, Y with joint probability function p(x,y) and marginal probability functions p(x) and p(y) respectively, the mutual information I(X;Y) is the relative entropy between the joint distribution and the product distribution:

$$I(X;Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)(y)} \quad (1)$$

Some of the information presented in the following sections for Mutual Information, and more specifically on the sections that present the hardware architectures, are also presented on the Transfer Entropy section for completeness reasons. There are many similarities on the base of the two metrics that lead to very similar architectures.

3.4.1 Summary of the initial version

Architecture

The architecture of the initial version of MI is presented thoroughly in the Deliverable 3.2. In short, the MI processing is divided in two parts, the probability density function (PDF) estimation and the calculation of Mutual Information from the function shown above (1). The PDF estimation, based on histograms, is calculated on software, as it requires random accesses, which is would be very slow if done on the DDR of the FPGA. The most efficient approach is for the PDF to be calculated in software, The MI system architecture is shown on Figure 1.

The PDFs, p(x,y), p(x), p(y), are streamed into hardware for the calculation of Mutual Information. The length of the streams depends on the number of Bins that is R^2. Each stream of the p(x) and p(y) is streamed R times, with R being the number of Bins, in order to match the size of p(x, y). In order to stream the data into the DFEs in the correct order either the values in p(x) or p(y) have to be copied R times each, while the other is streamed R times. The processing is fully pipelined meaning that each clock cycle the appropriate values of p(x,y), p(x) and p(y) have to arrive in the cores at the same time for the correct results to be produced.

The MI calculation utilizes 3 of the 8 streams available on the Maxeler System. In order to further accelerate the application, by utilizing the parallelization, the PDFs were divided by 2, providing 6 streams, 5 streams more precisely. Basically, the stream that is streamed R times is now streamed R/2 times. Also 2 hardware cores are responsible of calculating the partial MI results. This allowed double bandwidth utilization and thus even better performance. Also, even with two cores the FPGA resource Utilization remains at about 10% of the total available resources.

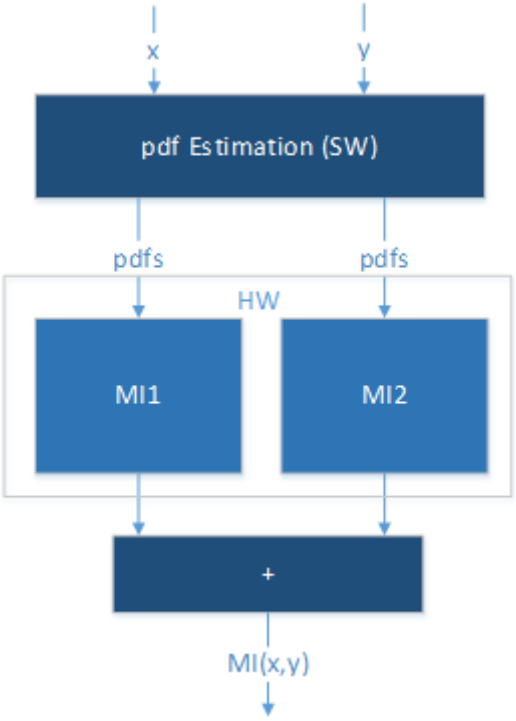

Figure 10: Mutual Information basic System Architecture

The basic hardware architecture for each core is more thoroughly presented on D3.2. The architecture for the calculation of MI (equation (1)) is presented on Figure 10.

Figure 11: Mutual Information Calculation Core Architecture (Equation (1))

The Sum module is responsible of accumulating the partial results. The floating point adder that accumulates the results needs 13 clock cycles in order to produce the result, but is fully pipelined. The feedback is done by using a 13 slot buffer. This means that the first result is reported 13 clock cycles after the inputs are inserted in the adder. In order to produce the correct result from hardware one value for each of the p(x,y), p(x) and p(y), should be streamed every 13 clock cycles, which would make the implementation very slow, exactly 13 times slower than the hardware implementation without such a restriction. In order to avoid this drawback data are streamed every clock cycle and are accumulated and stored in a FIFO. Basically, the first slot of the FIFO contains the results for the 1^{st}+14^{th}+27^{th} and so on iterations, the second the accumulation of the 2^{nd} +15^{th}+28^{th} and so on iterations. This applies in all the 13 slots of the feedback buffer.
At the end of the calculation, the last 13 results are streamed to the CPU where they are accumulated in order to produce the correct results. Finally, the Sum of the partial results of the two cores is done on software as mentioned above. Basically, 26 partial results, 13 from each core, are accumulated to produce the final MI.

Performance Results

In this section the performance results of the initial version of the MI implementation on hardware are presented. The platform where the experiments were run is the same MPC-C series Maxeler System, with two 6-core Intel Xeon @ 3.2 GHz with 50GB RAM, and 4 DFEs (XCV6475T FPGAs) connected via PCI with the CPU, with a 2GB/s bandwidth. The maximum bandwidth can be achieved by using the 8 streams available for each DFE. Each stream has a maximum bandwidth of 250 MB/s.
The results of the comparison of the hardware and software implementations are presented on the following Tables. In Table 4 the comparison of the software execution time with the hardware execution time with 2 MI calculation cores is presented. The experiments presented here and in the next section were done using 100.000 length time series for two random variables. The random variables represent stocks, while the time series are their values over time. The software used in these experiments is the equivalent single thread

implementation of the algorithm. It is written in C and the PDF estimation part is also used on the hardware implementation in the host code segment.
As mentioned above, there are 8 streams available for data streaming between the CPU and the DFE. By using 2 processing cores the processing power of the hardware is doubled. This improvement allowed an increase on performance of up to 9.4x for large number of Bins. Equation (1) allows the parallelization of the algorithm. The maximum available bandwidth for this initial architecture is the limiting factor as the 2 core implementation utilizes only 10% of the available FPGA resources while using 5 of the 8 available streams.

Num of Bins	SW(sec)	HW2(sec)	SpeedUp
100	0.002	0.036	0.05
500	0.025	0.037	0.68
1000	0.095	0.047	2
2000	0.4	0.077	5.2
5000	2.5	0.31	8
10000	10.3	1.1	9.4

Table 4: SW vs. HW MI calculation time with 2 cores

3.4.2 Architecture Modifications

MI Implementation version 2
In the next sections the second version of our implementation for MI is presented. The basic architecture difference between the 2 hardware versions is that the second has 3 MI calculation cores instead of 2. Another basic difference is that the data of the 1D PDFs do not have to be copied R times. Also some more optimizations were applied in order to optimize the memory accesses, which were also applied to the software version that also provided an increase in performance. Initially the optimizations were done on hardware, and then they were integrated also in software, providing the equivalent performance increase. A more efficient sequence of the math functions also provided an increase in performance to the software implementation.

Version 2 Architecture
The top-level hardware architecture for the MI computation is presented in Figure 11. The new addition is that now there are 3 cores responsible of calculating the MI, instead of two that the previous version had. Our solution consists of three stages. The first stage, represented by the 'PDF Estimation' module estimates the PDF values. In the second stage, the MI computation is performed on the hardware platform. The MI computation is partitioned into three parallel computational cores shown as 'MI' modules. The final step accumulates the partial results produced by the 3 MI cores and accumulates them.

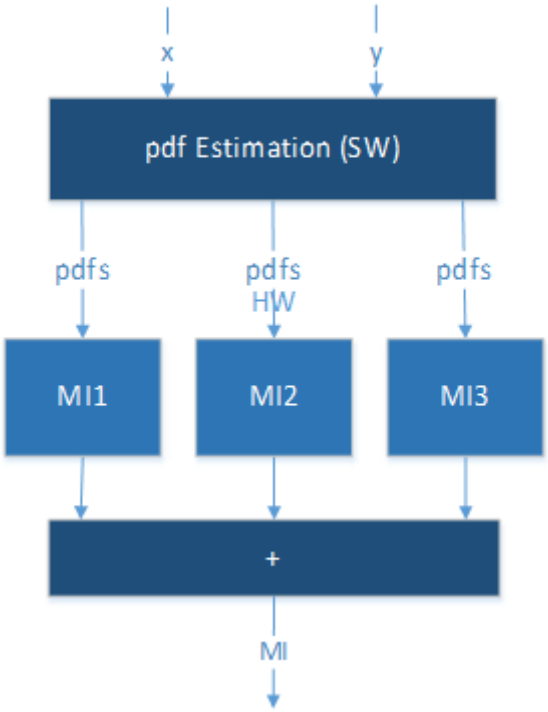

Figure 12:Mutual Information basic System Architecture Version 2

Further efficiency was achieved by dividing the PDF sequences p(x) and p(x,y) by 3, thus producing sub-streams of sizes R/3 and $R^2/3$ respectively. Each $R^2/3$ sub-stream is streamed R/3 times into the FPGA for alignment purposes. Thus, our proposed solution reaches almost the optimal bandwidth utilization of 7 out of 8 streams, with respect to the physical restrictions of the system and the nature of the input streams based on the computational measurement.
The use of 3 parallel processing hardware components leads to a significantly better performance. However, even with the deployment of 3 cores, the occupied FPGA resources were less than 12% of the total available resources. This implies that more processing units could be mapped to the hardware platform for more parallelization. However, this is restricted by the fact that the system offers a specific number of PCI streams that define how many cores can be fed at a time.

Multi-FPGA Architecture
The Maxeler MPC – C series offers a multi-FPGA platform comprising 4 FPGAs which, if utilized efficiently, allow further improvements in terms of system performance. We exploited the additional opportunities for parallelism by devising a multi-FPGA solution for the MI and TE computations. The corresponding implementation is presented in Figure 12. It presumes the existence of special-purpose task threads. The ‘receive’ and ‘transmit’ threads are responsible for receiving the input streams and transmitting the results. In our setting the input streams are financial time-series that represent stock values. The application calculates the pair wise MI or TE measurements between all the time series. In the case of MI there are N*(N-1)/2 time-series combinations due to the symmetric property of MI, while in the case of TE the number of pair wise combinations is N*(N-1).

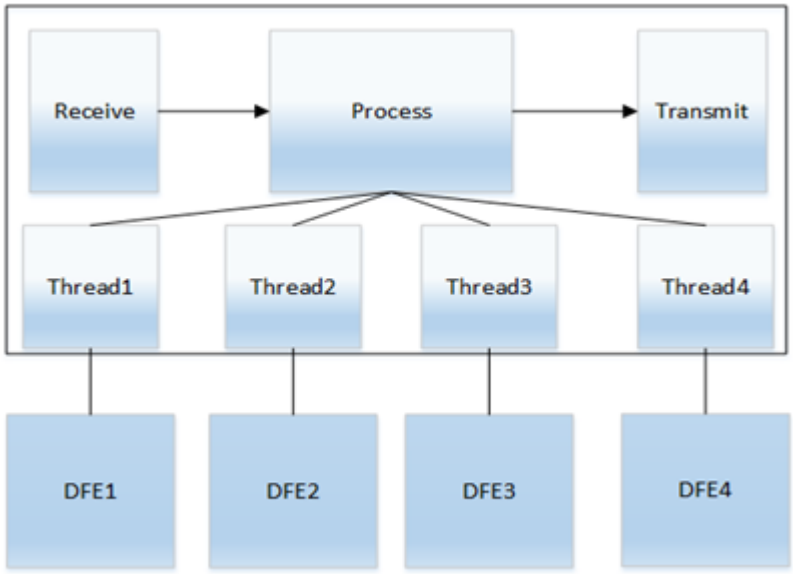

Figure 13: Multi-FPGA Architecture

The ‘process’ thread is responsible for distributing the work to the independent threads, shown as ‘Thread1 – Thread4’ in Figure 12. Each independent thread receives the data of 2 random variables, performs any required preprocessing and calls the corresponding FPGA. More specifically, the ‘process’ thread initiates one independent thread for each FPGA. These POSIX threads, usually referred to as Pthreads, are used to parallelize the 4 FPGA hardware calls, thus allowing up to 4 FPGAs to work in parallel.

Version 2 vs Version 1

In order to optimize the initial version implementation some changes were made on the hardware implementation. The most effective changes were on the memory utilization. Basically, there is no need for the duplication of the p(x) and p(y) R times. The other optimization is that the p(y) is streamed every R cycles, allowing further bandwidth to be utilized by the rest of the PDFs. This allowed a performance increase of about 2x. For small number of bins it can reach to up to 5x, mostly because of the efficient use of the cache of the processor and very little initialization time needs, which for such small processing times take a significant amount of the total runtime.

Num of Bins	HWv1(sec)	HWv2(sec)	SpeedUp
100	0.036	0.007	5.1
500	0.037	0.008	4.6
1000	0.047	0.015	3.1
2000	0.077	0.032	2.4
5000	0.31	0.14	2.2
10000	1.1	0.52	2.1

Table 5: HW version 2 vs. Initial Version hardware Implementation

On the software side similar optimizations were considered. More specifically the calculation of MI was changed in order to take full advantage of the cache memory. The PDFs can fit into the cache memory of the new high end processors. Also, the sequence of the calculations was changed. All such small optimizations allowed the acceleration of the software by 2x, while making it about 19x faster than the equivalent implementation of the R project, and more specifically the entropy package [5].

Num of Bins	SWv1(sec)	SWv2(sec)	SpeedUp
100	**0.002**	**0.002**	**1**
500	**0.025**	**0.012**	**2.1**
1000	**0.095**	**0.032**	**3**
2000	**0.4**	**0.11**	**3.6**
5000	**2.5**	**0.65**	**3.8**
10000	**10.3**	**2.58**	**4**

Table 6: SW version 2 vs. Initial Version software

Performance results

In our experimental setting the random variables are stocks, and their respective time-series show their values over time. The datasets were retrieved from financial market databases. In addition, the reference software implementation for both, MI and TE, used for performance comparison purposes is equivalent to the single thread implementation of the respective hardware solution. The software code was written in C and was highly optimized. The MI software implementation, our code is ~19x faster than the respective MI implementation included in the entropy package offered by the R [5] as mentioned above.

The performance of the single-FPGA with 3 MI cores is compared to the reference software implementation, in Table 7. For a small number of histogram bins the software approach is 3x faster than the hardware approach. This is attributed to the overhead induced by the hardware initialization process that consists of the hardware function call and the stream initialization phase. The time required for the completion of the initialization step ranges from 5 to 30 ms. Nevertheless, as the number of bins increases, the hardware performance significantly improves, since the initialization time becomes a small fraction of the actual processing time.

Num of Bins	SWv2 (sec)	HW 3 cores(sec)	SpeedUp
100	0.002	0.007	0.3
500	0.012	0.008	1.5
1000	0.032	0.015	2.1
2000	0.11	0.032	3.4
5000	0.65	0.14	4.6
10000	2.58	0.52	5

Table 7: SW vs. Single-FPGA HW MI execution time

The single-FPGA approach yields up to 5x performance acceleration. Regarding resource utilization 7 PCI streams of the DFE were used and only 11% of the available FPGA resources. Using 3parallel processing cores instead of one leads to a performance increase approximately linear to the number of cores. Thus, by using 3 cores instead of 1 we achieved ~3x increased performance.

The performance of the multi-FPGA approach with 3 MI cores per each FPGA is compared to the reference software implementation, in Table 8. The results show that the multi-FPGA implementation of MI that involves 4 FPGAs reaches 15.2x acceleration compared to the optimized software and is 3.2x faster than the corresponding single-FPGA system.

Num of Bins	SWv2 (sec)	HW 4DFEs (sec)	SpeedUp
100	0.03	0.03	1
500	0.16	0.04	4
1000	0.5	0.08	6.3
2000	1.73	0.17	10.2
5000	10.3	0.76	13.6
10000	41	2.7	15.2

Table 8: SW vs. Multi-FPGA HW MI Execution time for 16 MI calls

Ethernet I/O implementation

In our visit at London the first thing that was considered was to port our implementation to the latest Maia cards. The process was very easy, as the implementation is generic and can be easily ported to newer technologies. In the case of MI the results were the same with respect to performance.

The next attempt was to implement the MI algorithm using the network based card ISCA. The data are sent via a 10G Ethernet using UDP. Data structures were used to provide the appropriate data required for the calculation of MI. More specifically 32 cores were fit to the FPGA and each packet includes all the appropriate data needed by the 32 cores. The 32 MI calculation cores utilize more than 90% of the available resources of the FPGA. The performance results are shown on the following Table. The results showed that the MI algorithm implementation is not efficient using the ISCA card, as the bandwidth is even less than the PCI bandwidth. As a result this attempt led to speed down and thus was rejected.

Num of Bins	SWv2(sec)	HW32core(sec)	SpeedUp
100	0.002	0.005	0.4
500	0.012	0.008	1.5
1000	0.032	0.04	0.8
2000	0.11	0.15	0.7
5000	0.65	1.1	0.6
10000	2.58	3.5	0.7

Table 9: MI implementation performance on ISCA network based card

Final architecture

On our investigations on how to improve the performance of our implementations we concluded that the usage of the LMEM is a way to introduce optimization and thus increase the performance. During the visit at Maxeler we were introduced on the way to iterate over data written in the LMEM. This allowed the implementation of the final architecture for the Mutual Information.

Final Architecture- Utilization of LMEM

The final architecture for MI is shown on the following Figure. Basically, the usage of the LMEM in order to take advantage of its bandwidth allows an increase in performance. This architecture is implemented on the next generation Maxeler platform, the Maia.

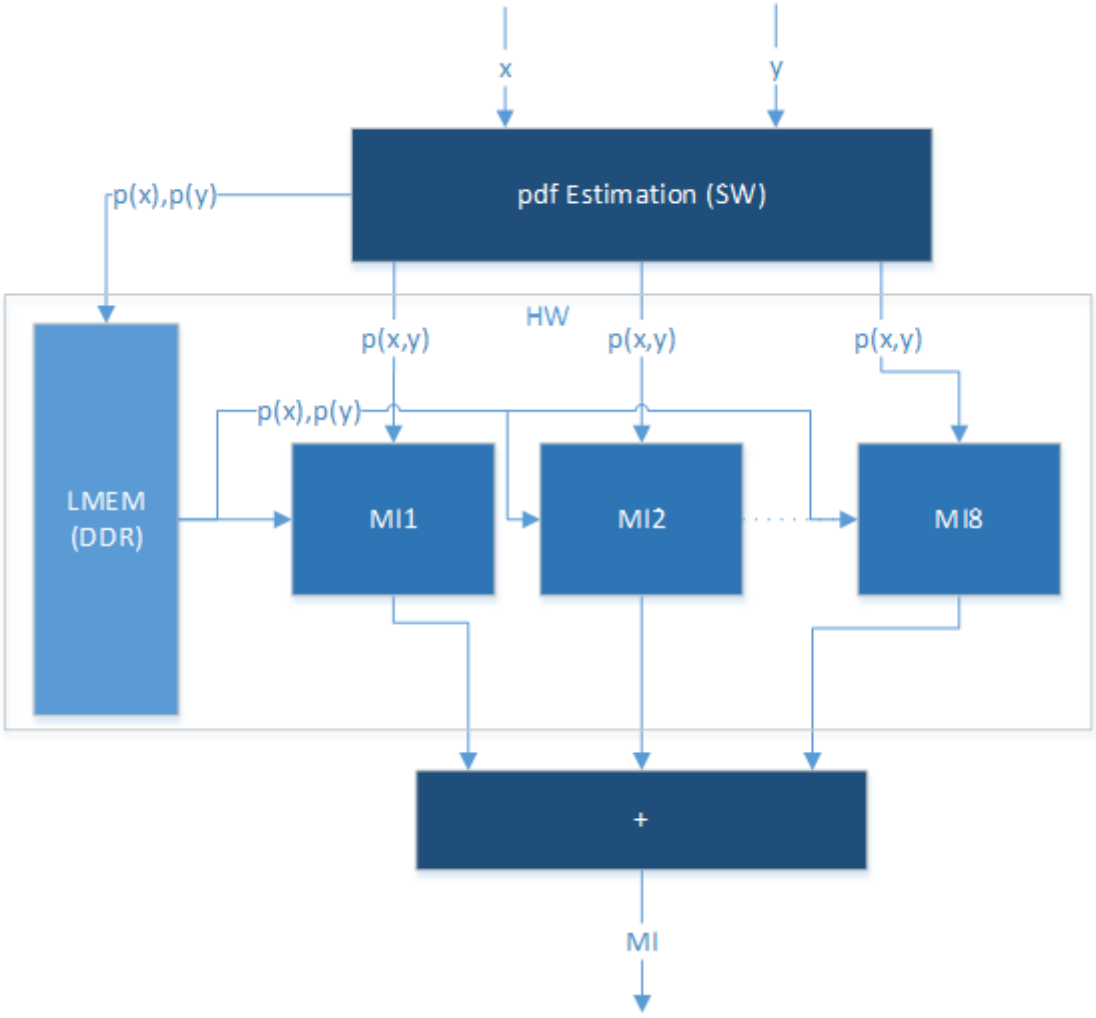

Figure 14: Final MI Architecture- LMEM Utilization

The difference between the previous architecture (version 2) is that the p(x) and p(y) PDFs are streamed from LMEM (DDR Memory) instead from PCIe. The core of the algorithm remains the same (Figure 10). This allows the whole PCIe bandwidth to be utilized by the p(x,y) PDF. So we can use all the 8 streams from the pc to the DFE for p(x,y).

First the p(x) and p(y) are written to the LMEM of the DFE from the host. This includes a call to the DFE to write the LMEM with the provided data. Then the p(x,y) is divided into 8 streams and is streamed to the DFE via the 8 available PCIe streams. This allows 8 MI calculation cores to be placed into the DFE that work in parallel. The 8 partial results are then accumulated in hardware and the result is streamed to the host.

This significantly increased the resource utilization as shown on the following Table. In more details, the Look Up Tables (LUTs) are the core structure for mapping the reconfigurable architecture in an FPGA device. The BRAMs are the memory modules that reside internally to the reconfigurable logic devices while the DSPs modules are components for implementing the main mathematical operations on reconfigurable hardware. The BRAM usage is mainly due to the Memory Controllers as we do not use any memory in our implementation.

	Percentage
LUTs	49%
BRAMs	47%
DSPs	24%

Table 10: Resource utilization

Performance Improvements

As far as the results with respect to performance the final architecture is up to 1.6 times faster than the previous architecture. The expected results would be more than 2x as the cores are more than doubled. The reason is that the overhead for writing to LMEM the p(x) and p(y) is included. In the case of MI this takes a significant portion of the total execution time. Actually for small numbers of bins the execution time is about the same between the 3 core and the 8core implementation. With the increase of the num of bins the performance increase is more visible as the write LMEM calls become a smaller portion of the total execution time.

Num of Bins	HWv23core(sec)	HW8core(sec)	SpeedUp
2*96	0.006	0.007	0.9
6*96	0.008	0.007	1.15
10*96	0.012	0.011	1.1
20*96	0.027	0.02	1.35
52*96	0.14	0.09	1.6
104*96	0.5	0.35	1.4

Table 11: Final Architecture vs. Version 2 Architecture

The number of bins has to be multiple of 2*96 as LMEM data have to be multiple of 384 bytes, which is the burst size for LMEM. The reason is that the p(y) variable is divided by 8 and as such it results in smaller than 384 bytes for small number of bins.

Padding is used if different size of bins is needed but it has a performance penalty. The best solution is for the number of Bins to be a multiple of 2* 96. We present here the results with this restriction as padding causes a very slight decrease in performance. We suggest if this architecture is used, that the num of bins is multiple of 2*96, if the applications allow it, in order to avoid the overhead introduced by padding.

For completion we present the increase in performance vs. the version 2 optimized MI software implementation on the following Table. As a result the final architecture is up to 7.7 times faster than the

equivalent software implementation. The maximum performance in stock/s for 10*96 bins, of the hardware system derived by the following results, is 13.5, as it can calculate ~91 MI values per second.

Num of Bins	SWv2(sec)	HW8core(sec)	SpeedUp
2*96	0.004	0.007	0.6
6*96	0.015	0.007	2.1
10*96	0.032	0.011	2.9
20*96	0.11	0.02	5.5
52*96	0.68	0.09	7.6
104*96	2.7	0.35	7.7

Table 12: Final Architecture vs. Version 2 Software

Multi - DFE

The multi DFE implementation for the final architecture is under development at the moment that the deliverable is being written. As a result we cannot provide any actual runtime results for a multi DFE implementation. At the moment we can only make a projection of the performance with the use of multiple DFEs based on the multi-DFE implementation of Version 2 (section 3.3.2.1.4). On the Vectis card the 4 DFE implementation is about 3 times faster than the single DFE. The expected performance of the multi-DFE implementation (4 DFE) is expected to be 3x better reaching 40.5 stocks/s. On the Maia card we expect to have about the same or even better results. The reason is that the initialization steps are faster on the Maia card than the Vectis. The initialization (4 DFE programming, stream initiation e.t.c.) is the reason that the actual increase in performance is not linear.

3.4.3 Verification

The verification of our initial software implementation was done by Maxeler Technologies as mentioned on deliverable 3.2. All the next implementations were verified using the same datasets. The initial software is run and the results are compared with each new implementation. If the results are not exactly the same the implementation is reconsidered and debugged to the point it has the expected behavior. Actually even the partial results are checked, and not only the final MI result. This allows the verification of the core architecture, as well as that the data are processed in the correct order.

3.4.4 Architecture cycles tradeoffs overview

In this section we will describe in brief the generations of our implementations for MI from the initial version till the final architecture that provided the best results with respect to performance.

The initial implementation was based on the first software implementation of the MI algorithm. This architecture provided a SpeedUp of up to 9 vs the initial software implementation. This performance increase was not significant, while the architecture used only 10% of the available resources with 2 calculation cores. The conclusion was that we should make an architecture that utilizes more efficiently the available

bandwidth. This leads to the Version 2 implementation. This implementation required less memory, while the bandwidth to the DFE was better divided between the PDFs. The Version 2 architecture was 2x faster than the initial version, partly due to 3 calculation cores and partly due to more efficient bandwidth utilization. The Version 2 implementation led to the implementation of the Version 2 software, using similar techniques, which led to better cache memory usage and provided a 2x faster software. The Version 2 software implementation is actually 19x faster than the equivalent implementation of the R project. The V2 hardware implementation is up to 5x faster than the V2 software, while the multi DFE implementation with 4 DFEs is 15x faster.

The next two implementations were done at the visit at Maxeler technologies. The first is the implementation on the ISCA network based card which actually did not provide any acceleration and was rejected. The second is the final architecture presented on the previous sections. This implementation utilizes the LMEM (DRAM) available on the Maia cards. The 1D PDFs are stored on the LMEM and are streamed from there to the DFE, while the 2D PDF is streamed through the PCIe. This allowed the utilization of the LMEM bandwidth in parallel with the PCIe. The final architecture is 1.6x faster than Version 2 architecture and 7.7x faster than the Version 2 software implementation. The final architecture has 8 calculation cores allowing much higher parallelization.

Further evaluation on the performance of the hardware implementation integrated on the QualiMaster Pipelinewill be included in the upcoming deliverables D3.4 and D5.4.

3.5 Transfer Entropy

Transfer Entropy is thoroughly presented in Deliverable 3.2. Actually, Transfer entropy is a non-parametric statistic measuring the amount of directed (time-asymmetric) transfer of information between two random processes. Transfer entropy from a process X to another process Y is the amount of uncertainty reduced in future values of Y by knowing the past values of X given past values of Y. The transfer entropy from X to Y can be written as:

$$T_{X\to Y} = \sum_{y_{n+1},y_n,x_n} p(y_{n+1}, y_n, x_n) log \frac{p(y_{n+1},y_n,x_n)p(y_n)}{p(y_{n+1},y_n)(y_n,x_n)} \quad (2)$$

Transfer Entropy is able to distinguish effectively driving and responding elements and to detect asymmetry in the interaction of subsystems.

Some of the information presented in the following sections for Transfer Entropy, and more specifically on the sections that present the hardware architectures, were also presented on the Mutual Information sections above for completeness reasons. This is done in order to allow someone studying only the algorithm of interest.

3.5.1 Summary of the initial version

Architecture

The system architecture for Transfer Entropy shown in Figure 14 is very similar to the architecture presented in 3.3.1 for MI. The TE calculation also starts with the PDF estimation, which is done on software, then the PDFs are streamed to the DFE where TE is calculated.

The PDFs p(x), p(x,y), p(x_{n+1},x) and p(x_{n+1},x,y) are streamed to the DFE. The length of the streams is R^3, with R being the number of bins, and R^3 is the size of p(x_{n+1},x,y). In order for the rest PDFs/streams to match the stream size of p(x_{n+1}, x, y), p(x) is copied R times and streamed R times, p(x, y) is streamed R times and p(x_{n+1}, x) is copied R times. These array copies increase drastically the memory requirements of the applications.

These streams utilize 4 of the 8 streams available on the Maxeler System. In order to further accelerate the application the PDFs were divided by 2, providing 8 streams, actually 6 streams are used as 2 streams remain the same for both cores, p(x) and p(x,y), which are streamed R/2 times. Also 2 hardware cores are responsible of calculating the partial TE results. This allowed more bandwidth utilization and thus even better performance. Also, even with two cores the FPGA resource Utilization remains at about 10% of the total available resources, indicating that the limiting factor remains the PCIe bandwidth, just like in MI. The results from the hardware cores are streamed and accumulated on software to produce the final TE result.

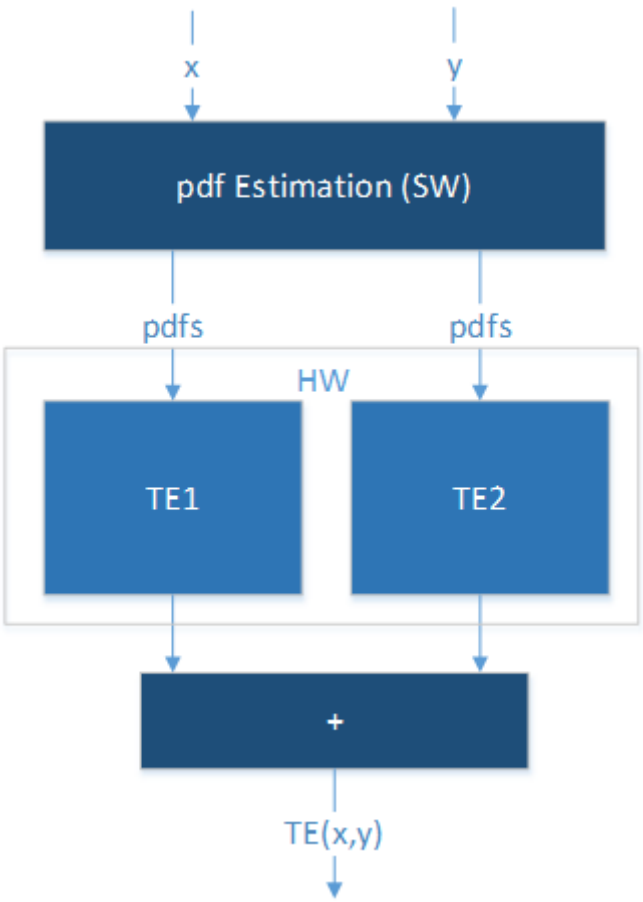

Figure 15: Transfer Entropy basic System Architecture

The basic hardware architecture for the calculation of TE (equation (2)) is presented on Figure 15. This architecture represents each one of the TE1 and TE2 cores shown in Figure 1. For a more thorough presentation of the basic core architecture refer to deliverable 3.2.

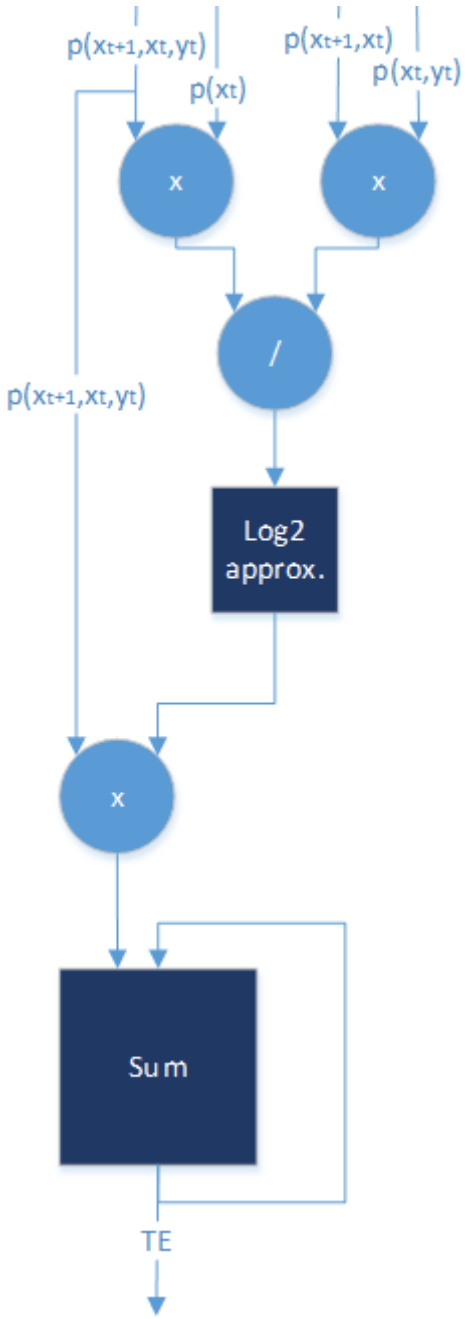

Figure 16: Transfer Entropy Calculation Core Architecture (Equation (2))

Performance Results

In this section the performance results of the initial version of the TE implementation on hardware are presented. The platform where the experiments were run is the same MPC-C series Maxeler System presented in the previous section for MI, with two 6core Intel Xeon @ 3.2 GHz with 50GB RAM, and 4 DFEs (XCV6475T FPGAs) connected via PCIe with the CPU, with a 2GB/s bandwidth. The maximum bandwidth can be achieved by using the 8 streams available for each DFE. Each stream has a maximum bandwidth of 250MB/s.

The results of the comparison of the hardware and software implementations are presented on the following Tables. In Table 1 the comparison of the software execution time with the hardware execution time with 2 TE calculation core is shown. The experiments presented here were done using 100.000 length time series for two random variables. The random variables represent stocks, while the time series are their values over time. The software used in these experiments is the equivalent single thread implementation of the algorithm.

It is written in C and the PDF estimation part is also used on the hardware implementation in the host code segment. Also it is optimized as it is 50% faster than the software presented in [4]. The processing time results show that even for small amounts of Bins the hardware implementation is faster than the software, unlike MI. The 100 Bins for TE (3D PDF) is like the 1000 bins for MI (2D PDF) with respect to the streamed data size, where MI is faster than software. As the number of bins increases, the hardware performance increases, as the initialization time is a small fraction of the calculation time.
There are 8 streams available for data streaming between the CPU and the DFE in the Maxeler MPC-C series system. In order to utilize more of the available bandwidth the PDFs were divided by two, which allows the use of 8 streams. Also by using 2 processing cores the processing power of the hardware is doubled. This improvement allowed an increase on performance of up to 5.4x for large number of Bins. Equation (2) just like in Mutual Information allows the parallelization of the Transfer Entropy algorithm. The maximum available bandwidth is the limiting factor as the 2 core implementation which utilizes less than 10% of the available FPGA resources while actually using 6 of the 8 available streams.

Num of Bins	SW(sec)	HW2(sec)	SpeedUp
100	0.074	0.046	1.6
200	0.53	0.11	4.8
500	6.1	1.1	5.5
800	23.3	4.5	5.2
1000	45.2	8.5	5.3
1200	77.8	14.5	5.4

Table 13:SW vs. HW TE calculation time with 2 cores

3.5.2 Architecture Modifications

TE Implementation version 2

In the next sessions the second version of our implementation for TE is presented. The same optimizations that were used in MI are also implemented for TE. Initially the optimizations were done on hardware, and then they were integrated also in software, providing the equivalent performance increase. The basic architecture difference between the 2 hardware versions is that the second has 3 TE calculation cores instead of 2. Another basic difference is that the data of the 1D PDFs don't have to be copied R^2 times and the 2D R times, in order to match the $p(x_{n+1},x,y)$. This reduced the memory needs of the application for the hardware implementation by a factor of 4. Also some more optimizations were applied in order to optimize the memory accesses, which were also applied to the software version that also provided an increase in performance. Also a better sequence of the math functions provided an increase in performance to the software implementation.

Version 2 Architecture

The top-level system architecture that performs the Transfer Entropy (TE) computation is shown in Figure 16. Similarly to MI, TE computation begins with estimating the required PDF values of equation (2). This operation is performed on the CPU, for the same reasons as in MI. Then, the PDF estimations p(x), p(x,y), $p(x_{n+1},x)$ and $p(x_{n+1},x,y)$ are streamed to the FPGA. The length of each stream has to be equal to R^3. The number R^3 derives from the histogram of the PDF $p(x_{n+1},x,y)$ which is represented by a 3-dimensional array of size R^3, since it records the occurrence of every triplet (X_{n+1},X,Y). In order to align the different sized PDF sequences, we streamed the PDF $p(x_{n+1},x,y)$ of size R^3 once, the PDF p(x) of length R repeatedly every R clock cycles and this was performed R times, the PDF p(x,y) of size R^2 repeatedly every R clock cycles and the PDF $p(x_{n+1},x)$ of size R^2, R times. Given that the system is fully pipelined the final output is produced after approximately R^3 clock cycles, since it is the time required for the entire PDF $p(x_{n+1},x,y)$ estimation to become accessible by the FPGA. Further efficiency was achieved by dividing sequences $p(x_{n+1},x,y)$ and p(x,y) by 3, thus producing sub-streams of sizes $R^3/3$ and $R^2/3$ respectively. Each $R^2/3$ sub-stream is streamed R/3 times to guarantee alignment. In addition to optimal bandwidth utilization, 3 TE hardware cores were mapped to the FPGA for parallel processing to take place. However, these processing cores only consumed a small percentage of the total available resources, a percentage less than 13%, as the parallel processing is restricted by the number of the available PCI streams that can be used to send data to the DFE. The results produced by the hardware cores are streamed out of the FPGA and are accumulated on the CPU to produce the final TE outcome.

The inner architecture of the TE computational hardware core remains the same as the initial version 1 (Figure 2) and it represents each of the TE modules illustrated in Figure 3. The basic core of the architecture is highly similar to the computational core of MI. The main difference between the two modules is the size of the streaming PDFs, which in the case of MI is R^2 and in the case of TE is R^3. The architecture is fully pipelined, thus allowing an iteration of the sum loop to be calculated every clock cycle. The four PDFs are streamed into the FPGA pipeline, one value of each PDF every clock cycle, are processed in the pipeline and the results are accumulated in the 'Sum' module.

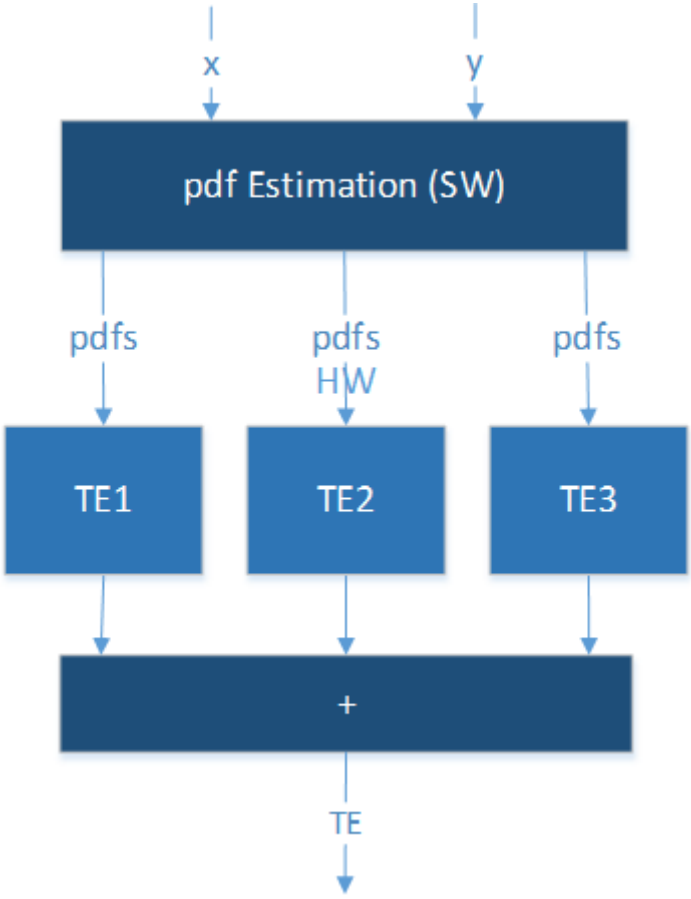

Figure 17:Transfer Entropy basic System Architecture Version 2

Multi-FPGA Architecture

The Maxeler MPC – C series offers a multi-FPGA platform comprising 4 FPGAs which, if utilized efficiently, allow further improvements in terms of system performance. We exploited the additional opportunities for parallelism by devising a multi-FPGA solution for the TE computations, using the same system architecture as in MI. The corresponding implementation is presented in Fig. 17. It presumes the existence of special-purpose task threads. The 'receive' and 'transmit' threads are responsible for receiving the input streams and transmitting the results. In our setting the input streams are financial time-series that represent stock values. The application calculates the pair wise TE measurements between all the time series. In the case of TE the number of pair wise combinations is N*(N-1).

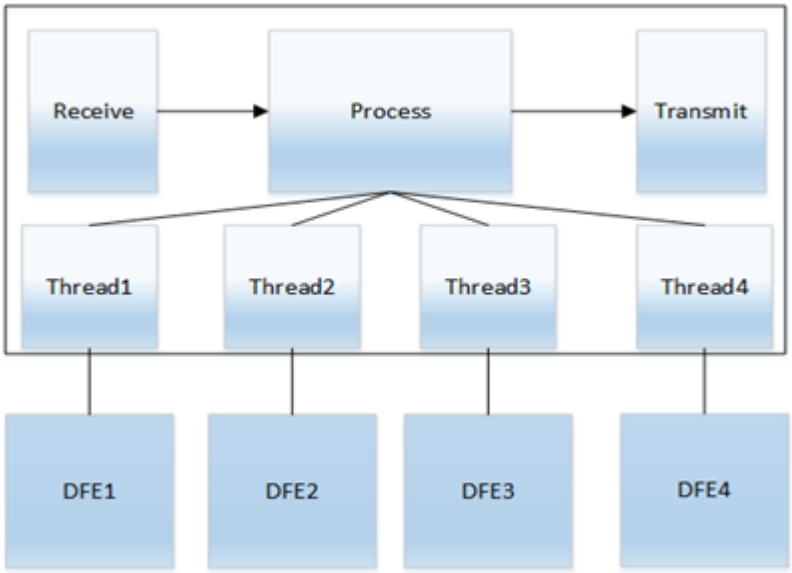

Figure 18: Multi-FPGA Architecture

The 'process' thread is responsible for distributing the work to the independent threads, shown as 'Thread1 – Thread4' in Figure 17. Each independent thread receives a pair of random variables (stocks) and their values overtime, calculates the PDFs and streams them to the DFE, which then calculates their respective TE value. More specifically, the 'process' thread initiates one independent thread for each FPGA. These Pthreads, are used to parallelize the 4 DFE hardware calls, thus allowing up to 4 FPGAs to work in parallel.

Version 2 vs Version 1

In order to optimize the initial version implementation of the TE algorithm, some changes were made on the hardware implementation, as mentioned in the previous sections. The most effective changes were on the memory utilization. Basically there is no need for the duplication of the $p(x)$ R^2 times and $p(x,y)$ and $p(x_{n+1},x)$ R times in order to match the size of $p(x_{n+1},x,y)$. Also $p(x)$ is streamed every R cycles and this is done R times. Finally $p(x_{n+1},x)$ is streamed R times and $p(x,y)$ is streamed every R cycles. The results are presented on the following Table.

Num of Bins	HWv1(sec)	HWv2(sec)	SpeedUp
100	0.046	0.02	2.3
200	0.11	0.05	2.2
500	1.1	0.6	1.8
800	4.5	2.4	1.9
1000	8.5	4.7	1.8
1200	14.5	7.7	1.9

Table 14:SW vs. HW TE calculation time with 2 cores

On the software side similar optimizations were considered. More specifically the calculation of TE was changed in order to take better advantage of the cache memory. Also, the sequence of the calculations was changed. All such small optimizations allowed the acceleration of the software by 2x, while making it about 2,5x faster than the equivalent implementation of the software implementation presented in [4]. The performance increase is presented on the following Table.

Num of Bins	SWv1(sec)	SWv2(sec)	SpeedUp
100	**0.074**	**0.05**	**1.5**
200	**0.53**	**0.28**	**1.9**
500	**6.1**	**3.51**	**1.7**
800	**23.3**	**12.8**	**1.8**
1000	**45.2**	**24.5**	**1.8**
1200	**77.8**	**40.9**	**1.9**

Table 15:SW version 2 vs. Initial Version software

Performance results

In our experimental setting the random variables are stocks, and their respective time-series show their values over time. The datasets were retrieved from financial market databases. In addition, the reference software implementation for TE, used for performance comparison purposes is equivalent to the single thread implementation of the respective hardware solution. The software code was written in C and was

highly optimized. In the case of the TE software implementation, our code is ~2.5x faster than the software implementation presented in [4].
The performance of the single-FPGA with 3 TE cores is compared to the reference software implementation, in Table 4. The results show that even for relatively small numbers of bins the hardware implementation is superior to the reference software, unlike MI. The acceleration achieved by the single-FPGA TE approach in the case of 100 bins is close to the performance of MI when considering 1000 bins. As the number of bins increases, the hardware performance increases, since again the initialization time becomes a small fraction of the processing time.

Num of Bins	SWv2 (sec)	HW 3 cores(sec)	SpeedUp
100	0.05	0.02	2.5
200	0.28	0.05	5.6
500	3.51	0.6	5.9
800	12.8	2.4	5.3
1000	24.5	4.7	5.2
1200	40.9	7.7	5.3

Table 16: SW vs. Single-FPGA HW TE execution time

Using 3 TE parallel processing cores yields a 5.3x speedup compared to the reference software. By using 3 parallel cores the processing power of the hardware system is tripled. Similarly to the case of MI the increase in performance is almost linear to the number of cores. Regarding resource utilization 8 PCI streams of the DFE were used, i.e. the maximum number of available PCI streams and only 13% of the available FPGA resources. The maximum available bandwidth is the restricting factor since the hardware solution utilizes a relatively small percentage of the available resources.
The performance of the multi-FPGA approach with 3 TE cores per each FPGA is compared to the reference software implementation, in Table 17.

Num of Bins	SWv2 (sec)	HW 4DFEs (sec)	SpeedUp
100	0.72	0.16	4.5
200	4.27	0.46	9.3
500	54.5	3.93	13.9
800	202	12.8	15.8
1000	390	23.4	16.7
1200	636	36.7	17.3

Table 17: SW vs. Multi-FPGA HW TE Execution time for 16 MI calls

Results in Table 17 show that the multi-FPGA implementation of TE that utilizes 4 FPGAs reaches 17.3x acceleration compared to the optimized software and is 3.4x faster than the corresponding single-FPGA system.
The only implementation this work could be safely compared to is the one presented in [4] as the experiments are carried out on the same platform. Our multi-FPGA approach is ~17x with 4 DFEs faster than our reference software implementation, while the solution presented in [4] yields speedups around 112x with 1DFE compared to their software, which is 2.5x slower than our software implementation. This happens due to the optimization techniques implemented in [4]. However, these techniques are data dependent. For example, they use bit-width narrowing to achieve better bandwidth utilization. Yet, for different datasets this technique may not be viable. Even though the work presented in [4] has the scope of addressing any dataset while maintaining the required accuracy, applying techniques like the ones in their implementation generally reduces flexibility and makes the solution less applicable to real-world problems.

Ethernet I/O implementation
In our visit at London the first thing that was considered was to port our implementation to the latest Maia cards. The process was very easy, as the implementation is generic and can be easily ported to newer technologies. In the case of TE as in MI the results were the same with respect to performance. The reason is that the limitation was the PCIe bandwidth, which is 2GB/s just like on the Vectis cards.
The next attempt was to implement the TE algorithm using the network based card ISCA. The data are sent via a 10G Ethernet using UDP. Data structures were used to provide the appropriate data required for the calculation of TE. More specifically 36 cores were fit to the FPGA and each packet includes the all data needed by the 36 cores. The 36 TE calculation cores utilize more than 90% of the available resources of the FPGA. The performance results are shown on the following Table. The results showed that the TE algorithm implementation is not efficient using the ISCA card, as the bandwidth is even less than the PCIe bandwidth. As a result this attempt led to speed down and thus was rejected.

Num of Bins	SWv2 (sec)	HW 4DFEs (sec)	SpeedUp
100	0.05	0.05	1
200	0.28	0.25	1.1
500	3.51	3.1	1.1
800	12.8	15.8	0.8
100	0.05	0.05	1
1200	0.28	0.25	1.1

Table 18: TE implementation performance on ISCA network based card

3.5.3 Final architecture

Similar to Mutual Information we used LMEM in order to increase the performance of the Transfer Entropy architecture. The details of the final architecture for TE are presented on the following sections.

Final Architecture- Utilization of LMEM

The final architecture for TE is shown on the following Figure. Basically, the usage of the LMEM, in order to take advantage of its bandwidth, allows an increase in performance just like in the Mutual Information case. The increase is even more visible in TE. Even for small number of bins, as the execution time is much larger and thus the write LMEM function calls take a very small portion of the execution time This architecture is implemented on the next generation Maxeler platform, the Maia.

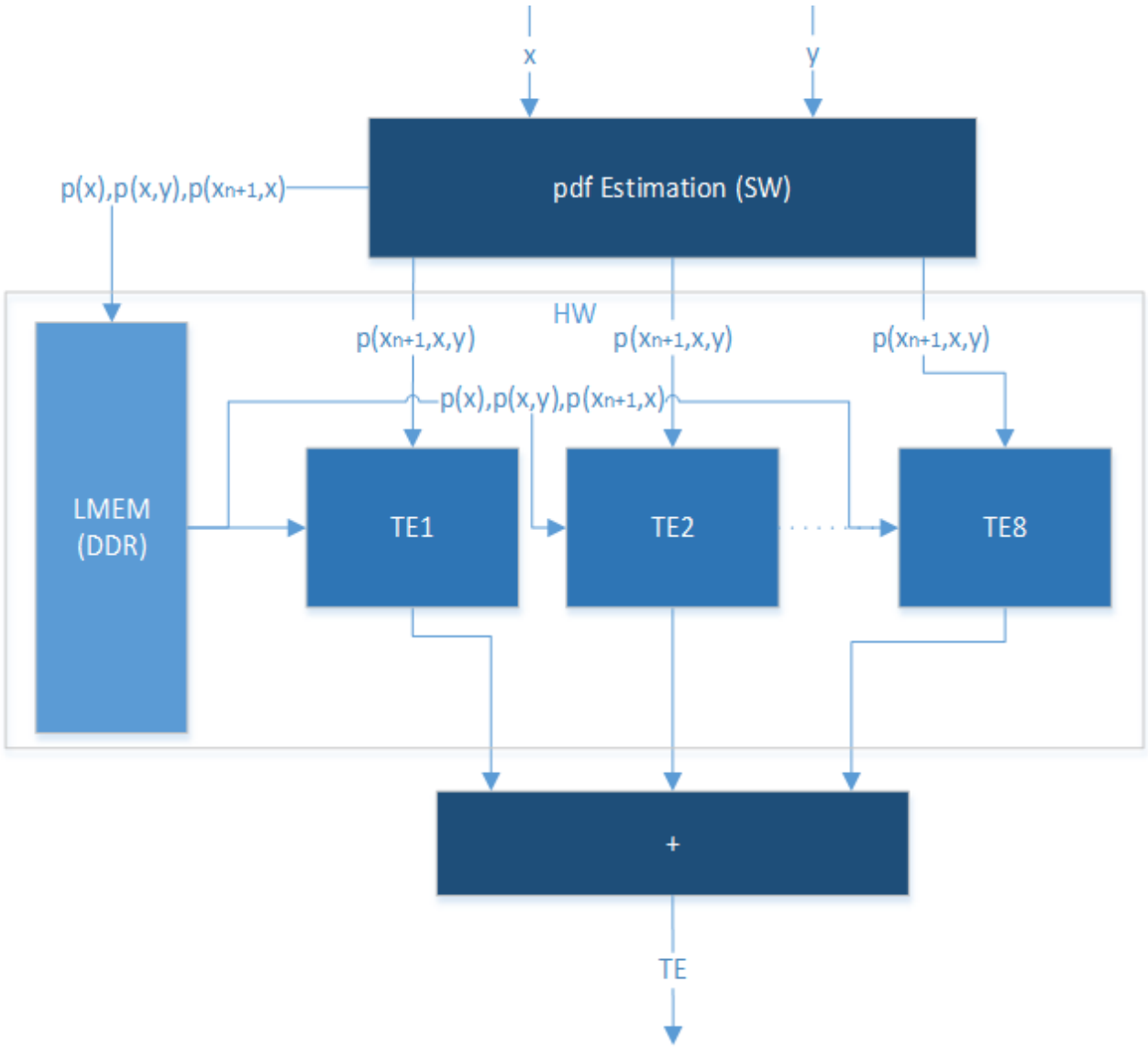

Figure 19: Final MI Architecture- LMEM Utilization

The difference between the previous architecture is that the p(x), p(x,y) and $p(x_{n+1},x)$ PDFs are streamed from LMEM instead from PCIe. The core of the algorithm remains the same (Figure 15). This allows the whole PCIe bandwidth to be utilized by the $p(x_{n+1},x,y)$ PDF. So we can use all the 8 streams from the CPU to the DFE for $p(x_{n+1},x,y)$.

First the p(x), p(x,y) and $p(x_{n+1},x)$ are written to the LMEM of the DFE from the host. This includes a call to the DFE to write the LMEM with the provided data. Then the $p(x_{n+1},x,y)$ is divided into 8 streams and is streamed to the DFE via the 8 available PCIe streams. Also p(x,y) is divided into 8 streams in order to have the correct data sequence for the calculation. Actually 8 streams are initiated from different addresses of p(x,y) from LMEM. This allows 8 TE calculation cores to be placed into the DFE that work in parallel. The 8 partial results are then accumulated in hardware and the result is streamed to the host.

This significantly increased the resource utilization as shown on the following Table. The BRAM usage is mainly due to the Memory Controllers as we do not use any memory in our implementation.

	Percentage
LUTs	49%
BRAMs	48%
DSPs	23%

Table 19: Resource utilization

3.5.4 Verification

The verification of our initial software implementation was done by Maxeler Technologies as mentioned on deliverable 3.2. All the next implementations were verified using the same datasets. The initial software is run and the results are compared with each new implementation. If the results are not exactly the same the implementation is reconsidered and debugged to the point it has the expected behavior. Actually even the partial results are checked, and not only the final TE result. This allows the verification of the core architecture, which does the calculations of equation (2), as well as that the data are processed in the correct order.

3.5.5 Performance Improvements

As far as the results with respect to performance the final architecture is up to 1.8 times faster than the previous architecture. The expected results would be more than 2x as the cores are more than doubled. The reason is that the overhead for writing to LMEM the p(x), $p(x_{n+1},x)$ and p(x,y) is included on the total runtime.. Actually for small numbers of bins the execution time is about the same between the 3 core and the 8core implementation. With the increase of the num of bins the performance increase is more visible as the write LMEM calls become a smaller portion of the total execution time. Another factor that increases the overhead is the initialization of the 10 streams from the LMEM.

Num of Bins	HWv.2 (3cores) (sec)	HW (8cores) (sec)	SpeedUp
96	0.01	0.01	1
2*96	0.043	0.025	1.7
5*96	0.52	0.33	1.6
8*96	2.1	1.2	1.8
10*96	4	2.4	1.7
12*96	6.8	4.1	1.7

Table 20: Final Architecture vs. Version 2 Architecture

The number of bins has to be multiple of 96 as LMEM data have to be multiple of 384 bytes, which is the burst size. The difference from MI is that the PDFs are a lot larger in the TE case, so the division of the p(x,y) does not lead to streams with size smaller than 384 bytes. Padding is used if different size of bins is needed but it has a performance penalty. The best solution is for the number of Bins to be a multiple of 96. We present here the results with this restriction, as padding causes a very slight decrease in performance. We suggest if this architecture is used, that the num of bins is multiple of 96, if the applications allow it, in order to avoid the overhead introduced by padding.
For completion we present the increase in performance vs. The version 2 software on the following Table. As a result the final architecture is up to 10 times faster than the equivalent software implementation.

Num of Bins	SWv2(sec)	HW8core(sec)	SpeedUp
96	0.04	0.01	4
2*96	0.25	0.025	10
5*96	3.2	0.33	9.7
8*96	11.6	1.2	9.7
10*96	22	2.4	9.2
12*96	37	4.1	9

Table 21: Final Architecture vs. Version 2 Software

The above results show that the hardware system with 1 DFE can calculate up to 40 TE values every second, for 2*9 bins. This means that the hardware implementation can calculate the pairwise TE between 6 different stocks every second.

Multi - DFE

Just like with the Mutual Information case the multi DFE implementation for the final architecture is under development at the moment that the deliverable is being written. As a result we cannot provide any actual runtime results for a multi DFE implementation. At the moment we can only make a projection of the performance with the use of multiple DFEs based on the multi-DFE implementation of Version 2. On the Vectis card the 4 DFE implementation is about 3.4 times faster than the single DFE. The performance in stocks/s for the 4 DFEs is expected to be ~ 21 stocks/s. On the Maia card we expect to have about the same or even better results. The reason is that the initialization steps are faster on the Maia card than the Vectis. The initialization (4 DFE programming, stream initiation etc.) is the reason that the actual performance is not increased by 4.

3.5.6 Architecture cycles tradeoffs overview

In this section we will describe in brief the generations of our implementations for TE from the initial version till the final architecture that provided the best results with respect to performance.

The initial implementation was based on the first software implementation of theTE algorithm. This architecture provided a SpeedUp of up to 5.5 vs the initial software implementation. This performance increase was not significant, while the architecture used only 12% of the available resources with 2 TE calculation cores. The conclusion was that we should make an architecture that utilizes more efficiently the available PCIe bandwidth. This leads to the Version 2 implementation. This implementation required less memory, as we do not need to make the 1D and 2D PDFs as big as the 3D PDF, while the bandwidth to the DFE was better divided between the PDFs. The Version 2 architecture was about 2x faster than the initial version, partly due to the 3rd core integration and partly due to bandwidth and memory optimizations. The Version 2 implementation lead to the implementation of the Version 2 software, using similar techniques, which led to better cache memory usage and provided a 1.8x faster software. The Version 2 software implementation is actually 2.5x faster than the equivalent implementation presented on [4] and is considered optimized. The V2 hardware implementation is up to 5.9x faster than the V2 software, while the multi DFE implementation with 4 DFEs is 17x faster.

The next two implementations were done at the visit at Maxeler technologies. The first is the implementation on the ISCA network based card which actually did not provide any acceleration and was rejected. The second is the final architecture presented on the previous sections. This implementation utilizes the LMEM (DRAM) available on the Maia cards. The 1D PDF as well as the 2D PDFs are stored on the LMEM and are streamed from there to the DFE, while the 3D PDF is streamed through the PCIe. This allowed the utilization of the LMEM bandwidth in parallel with the PCIe. The final architecture is 1.8x faster than Version 2 architecture and 9x faster than the Version 2 software implementation. The main reason is the parallelization by 8, as the final architecture integrates 8 TE calculation cores, instead of 3 of V2. The effects of the final architecture are more significant for TE than MI because the initialization of LMEM is a significant amount of the total execution time for MI.

As far as the performance on the QualiMaster Pipeline further evaluation will be included in upcoming deliverables D3.4 and D5.4.

4. Outlook for the design automation tools (MAX)

The goal of the design automation tool is to support the designer in systematically translating algorithms into a hardware implementation as well as to investigating partially automated translation methods. In this context, it needs to be pointed out that it is very challenging to automatically turn a sequential software implementation into optimized hardware parallel implementations, especially in a general, non-domain-specific approach. Hence, the goal here is not to attempt the fully automatic translation of sequential legacy code into hardware. Instead, it is more practical to support the designer in creating hardware implementations by designing on a high level, increase productivity and automate time-consuming optimization steps. Maxeler already provides a high-level design environment where hardware implementations are created by describing a dataflow model of the application. This dataflow model is developed in MaxJ, a high-level meta-language based on Java. It allows the application developer to focus algorithm implementation and optimization across several layers of abstraction without having to deal with low-level implementation details in Verilog and VHDL. Developing in MaxJ is already far more productive that translating an algorithm into hardware by following a conventional low-level approach. Examples of low-level analysis and implementations were given in Deliverable D3.1. However, MaxJ exposes some optimization options to the designer that may not be immediately obvious to a traditional software developer. For example, the designer has full control over the numerical representation of the data (including custom number formats) or where to store and buffer data. This is simply due to the fact that the underlying compute substrate is reconfigurable and more flexible than a conventional CPU. As a result, more optimization options are available and they cannot be fully automated without any application knowledge. While it is important to have these optimization options available, it is key make them easily available to the designer and support them with partial automation and visualization in order to increase designer productivity. Currently, the tool development pursues two specifics areas for design optimization and increased productivity: a) FIFO visualization and b) kernel merging. These two aspects were chosen because the can be time-consuming and difficult to perform by hand and therefore benefit from tool support.

FIFO generation is an inherent feature of the graph scheduling that is performed by MaxCompiler. Developers writing MaxJ code do not have to explicitly schedule operations in the graph or balance delays through operators with different latencies. If two parallel operations have different latencies, the compiler will automatically insert a First-In First Out (FIFO) buffer to balance theses different latencies. This automatic scheduling of the dataflow graph is one of the key advantages in writing high-level MaxJ code over manual low-level hardware design. In practice the compiler will balance different latencies by automatically inserting FIFOs into the graph. When generating the hardware design, the compiler will use some of the reconfigurable chip resources such as flip-flops or small on-chip RAMs to implement the FIFOs. However, these resources, in particular the on-chip RAM, is also valuable as a resource for on-chip buffering. Hence, it is beneficial to optimize the FIFO usage since it can improve overall performance of the design. FIFO optimization currently requires the designer to manually inspect the generated dataflow graph and link the present FIFOs to parts of the original application and modify it accordingly. This can be a time-consuming progress. Therefore, it is now targeted by a new optimization tool. This tool generates a new visual representation that links FIFO resource usage to specific lines of application source code that are responsible for the FIFO creation. This visual representation vastly improves the designer productivity for this optimization step by automatically linking generated hardware resources to lines of source code. An example is shown in Figure 20. It shows the complexity of a graph in real applications. Here every node links resource usage to lines of code. Without this visual representation, FIFO optimization would be very difficult.

The second aspect that is targeted by the optimization tool is kernel merging. As explained above, MaxCompiler automatically generated dataflow graphs based on the application developer's MaxJ description. A compute intensive part of an application mapped to a dataflow graph is called a kernel and often multiple kernels combined into an accelerator design which is offloaded to reconfigurable hardware. Since kernels coexist in the same hardware design they use the same pool of chip resources, i.e. adding more kernels means that the chip will fill up at some point. A new optimization approach developed here is kernel

merging. The key concept of kernel merging is to facilitate the reuse of hardware blocks between multiple independent computations, where the result of those computations is not needed at the same time, which are to be implemented on the same chip usually as separate hardware modules laid out in space rather than time. A simple example would be the computation of $(x + y)^2$ and $(x + y) * z$ which result in different dataflow graphs. However, both graphs have in common one adder and one multiplier which are interconnected in different ways. It is therefore possible to implement both computations by sharing the same operators and inserting control logic to switch between the two functionalities. This transformation is illustrated in Figure 21 and 22. Merging the graphs reduces the hardware requirements significantly since only one addition and one multiplication have to be implemented. While this seems to be trivial for this simple example, the search for optimal solutions is far more complex when graphs with thousands of nodes are considered. We are currently developing a tool to automatically merge multiple dataflow subgraphs in order to create a single structure implementing all distinct computations with minimal area overhead. The tool combines several optimizations and kernel merging employing a greedy heuristic, which can lead to significant resource reductions in the hardware design size compared to a naive hardware design.
The work on the tool highlighting the impact of scheduling on hardware resource usage is finished. The tool merging dataflow graphs is still in development. While the core functionality is finished current efforts try to improve fixed point arithmetic support and add more optimizations. The final result and evaluation on the tool will be presented in the final deliverable D3.4.

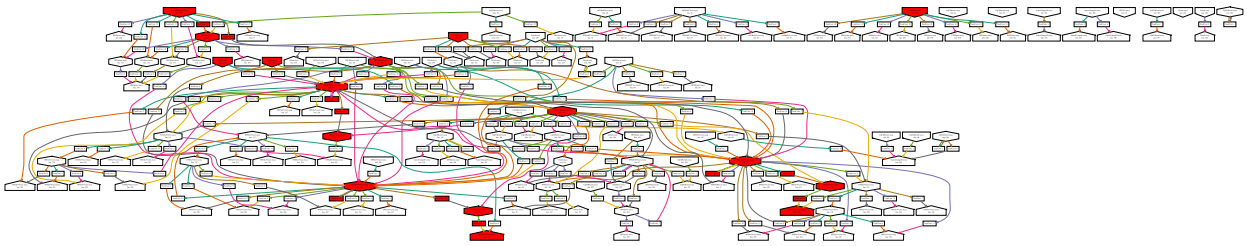

Figure 20: Dataflow graph highlighting FIFOs which use a considerable amount of memory resources.

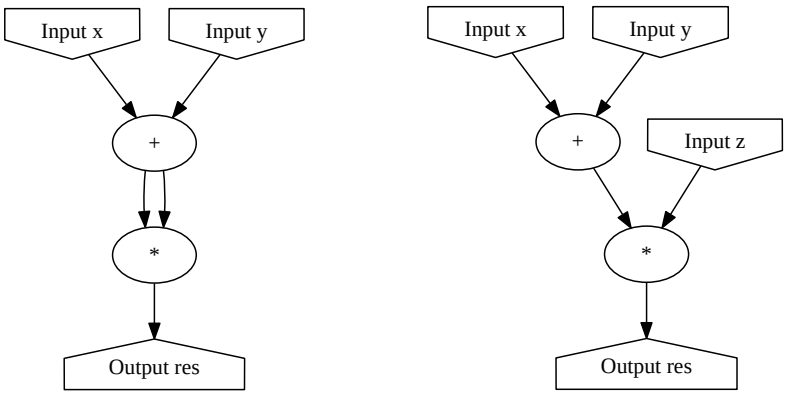

Figure 21: Dataflow graphs than can be merged.

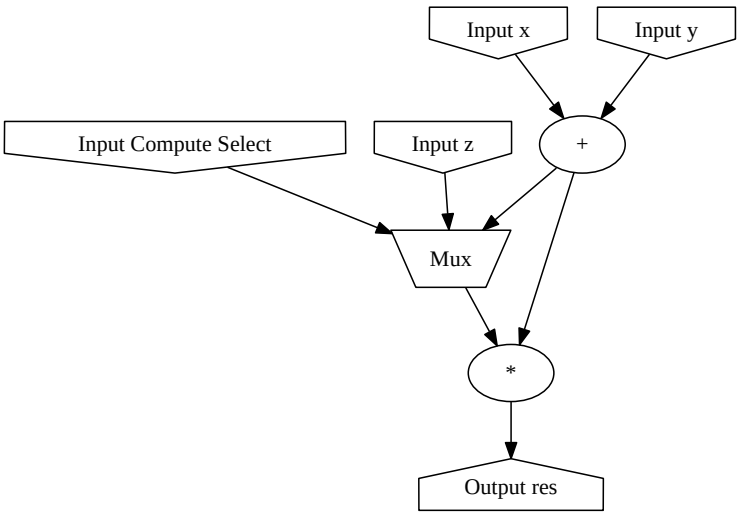

Figure 22: Merged dataflow graphs shown in Figure 19.

5. Conclusions

Deliverable D3.3 describes the second design cycle for the hardware modules implementing four algorithms for the QualiMaster pipelines. This deliverable continues and extents work that have been done in D3.2. In D3.2 several designs have been implemented focusing on the precise functionality of these algorithms comparing with the official software. In D3.3 improved or new architectures have been designed, focusing on performance boosting and to be integrated on the generic interface that have been implemented for the QualiMaster Pipeline. Taking a further step, four basic rules for hardware module design were encoded as a guideline for a high level CAD tool.

This interface gets data from other nodes of Storm that implement QualiMaster pipeline, put them in order according to their time stamp, send them to the hardware module, and when it receives them processed, transmit them back to the right node to continue in the QualiMaster Pipeline. This interface can be used as a node performing this process in parallel with software, depending on the Pipeline setup. This interface seems trivial as it follows a typical communication scheme, but technically is very challenging as it integrates a number of technologies and protocols (Storm, Maxeler, QualiMaster Pipeline) which are not inherently compatible.

For the four algorithms that have been redesigned, performance proved to be better than the first version. Performance improvement was from 36% to 2 times faster than the previous generation which is a factor of multiplication to the impressive results against software that were presented in D3.2. In addition 2nd generation designs were ported to different Maxeler platforms exploiting different platform features.

The rules that have been encoded are some high level guidelines which always a designer has in mind. A tool implementation has these rules as a goal, but first has to express them in mathematical form. For that reason this set of rules is not expressed strictly but as a general guideline that can provide hardware modules with proper performance.

Concluding the design cycle for second generation of hardware modules, offers implementations with performance boosting, fully functional in the QualiMaster pipeline, and portable to different implementation platforms and guidelines towards an automatic design tool.

References

[1] Papapetrou, O., Garofalakis, M., &Deligiannakis, A. (2012). Sketch-based querying of distributed sliding-window data streams. Proceedings of the VLDB Endowment, 5(10), 992-1003.

[2] Cormode, G., & Muthukrishnan, S. (2004, April). An improved data stream summary: The count-min sketch and its applications. In Latin American Symposium on Theoretical Informatics (pp. 29-38). Springer Berlin Heidelberg.

[3] Datar, M., Gionis, A., Indyk, P., & Motwani, R. (2002). Maintaining stream statistics over sliding windows. SIAM journal on computing, 31(6), 1794-1813.

[4] Shao, S., Guo, C., Luk, W., & Weston, S. (2014, December). Accelerating transfer entropy computation. In IEEE*Field-Programmable Technology (FPT), 2014 International Conference on* (pp. 60-67).

[5] https://cran.r-project.org/web/packages/entropy/index.html