A Nanosecond-level Hybrid Table Design for Financial Market Data Generators

Haohuan Fu*, Conghui He*, Wayne Luk†, Weijia Li*, and Guangen Yang*
*Tsinghua University, Email: {haohuan,ygw}@tsinghua.edu.cn, {hch13,liwj14,}@mails.tsinghua.edu.cn
†Imperial College London, Email: w.luk@imperial.ac.uk

***Abstract*—This paper proposes a hybrid sorted table design for minimizing electronic trading latency, with three main contributions. First, a hierarchical sorted table with two levels, a fast cache table in reconfigurable hardware storing megabytes of data items and a master table in software storing gigabytes of data items. Second, a full set of operations, including insertion, deletion, selection and sorting, for the hybrid table with latency in a few cycles. Third, an on-demand synchronization scheme between the cache table and the master table. An implementation has been developed that targets an FPGA-based network card in the environment of the China Financial Futures Exchange (CFFEX) which sustains 1-10Gb/s bandwidth with latency of 400 to 700 nanoseconds, providing an 80- to 125-fold latency reduction compared to a fully optimized CPU-based solution, and a 2.2-fold reduction over an existing FPGA-based solution.**

***Keywords*-FPGA; finance; algorithm; low latency; HPC**

I. Introduction

In recent years the automation of financial trading has driven the demand for trading applications and systems with challenging latency requirements [1]. Although the software solutions provide the flexibility to express algorithms in high-level programming models and to recompile quickly, they are becoming increasingly uncompetitive especially in financial applications due to the long and unpredictable response time mainly from the operating system network stacks and system interrupts. Nowadays, Field Programmable Gate Arrays (FPGAs) have shown to be a promising technology for high frequency trading and algorithmic trading applications [2] [3] [4], which achieve a low and constant latency for processing packets in hardware directly.

However, existing FPGA-based low-latency solutions to financial packet processing mainly focus on parsing, decompressing, filtering and forwarding the market data feeds without the need of maintaining a large volume of structured data. It is generally challenging for FPGAs in both design philosophy and memory resource to organize a large volume of packets with complicated routines, such as insertion, deletion, selection, sorting, etc. For instance, the market data generator (MDG) in CFFEX for generating market data feeds from gigabytes of order books is still based on a pure software solution, which becomes one of the bottlenecks of the low-latency trading engine. Mapping the MDG to reconfigurable hardware can potentially reduce latency, while also raise great challenges for maintaining gigabytes of order books with sophisticated routines.

FPGAs have been used in speeding up demanding routines such as sorting and database operations [5] [6] [7]. However, the data in these applications streamed to FPGAs are all from the host memory. As a result, frequent data transfers between the CPU and the FPGA greatly reduce the latency gain obtained from network processing, which goes to the opposite side of low latency.

This paper proposes a CPU-FPGA hybrid sorted table design for financial market data generators, which is able to maintain a large volume of GBs of data in nanoseconds. It is integrated into the MDG with the capability of meeting the current and future requirements regarding latency and data volumes. Our key contributions are as follows.

- A CPU-FPGA hybrid sorted table design, which consists of a cache table on the FPGA that stores MBs of most frequently used data items, and a master table at the CPU host that stores GBs of data items.
- A complete set of routines for the hybrid sorted table, including insertion, deletion, selection, etc., providing an extremely low latency at the scale of a few cycles.
- An on-demand synchronization scheme between the cache table and the master table.

The rest of the paper is organized as follows. Section II discusses the FPGA solutions to financial applications and the challenges of low-latency database-style operations. Section III presents a CPU-FPGA hybrid sorted table. Section IV proposes a complete set of database-style low-latency routines along with the hybrid sorted table. Section V discusses an on-demand synchronization strategy to exchange data between the FPGA and the CPU host. Section VI proposes a performance model and shows our experimental results for the resource usage, the latency, and the comparison with the CPU-based system in difference test cases. Finally, Section VII gives a conclusion of this study.

II. Background

A. Financial Market Data Generator

As a hub organization that enables financial transactions, one of the key functions of the financial exchanges is to provide the traders with the financial market data feeds that include changes in prices and market conditions.

The market data generator in exchanges is not only responsible for the parsing and filtering of the packets at

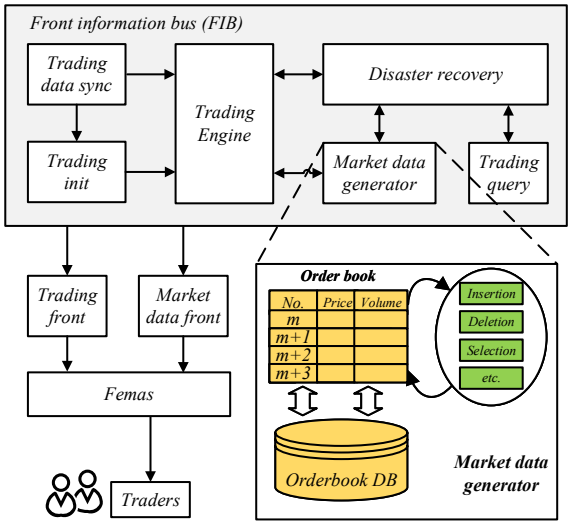

Figure 1. The architecture and the market data generator in CFFEX.

a high message rate, but also for the maintenance of GBs of data items in a structured format, which is usually called the order book. An order book is a list of buy and sell orders for a particular financial instrument, organized by price level. The order book lists the number of shares being bid or offered at each price point, or market depth. A market data generator needs to update and maintain order books in real time, which needs to support a complete set of operations such as insertion, deletion, selection, sorting, etc.

Over time the number of messages produced have drastically increased. This increase has a number of causes, such as

- Increased range of products. The number and scope of traded instruments have continually increased. In CFFEX for example, the number of instruments is expected to be growing from dozens to hundreds or even thousands in the next few years.
- High-resolution monitoring. In current configurations of exchanges, the changes in prices are often reported at the resolution of half a cent (0.005 CNY), which results in much more frequent updates due to small price movements.
- Algorithmic trading. Algorithms generate and cancel trades at a much higher rate than human traders, leading to an increased message rate to the exchange.

The pure software solutions of a market data generator are generally unable to scale with the growing message rate and meet the requirement of low latency. The conventional FPGA designs that store the order books on the host memory while employing the FPGAs to accelerate the database operation are also unable to sustain during a large burst of trading activities, mainly because the data transfer between the FPGA and the CPU via PCIe will greatly increase the latency. It is also impractical to store GBs of the order book data on the chip while supporting fast database operations. Our proposed approach is to use an FPGA-based streaming solution that employs a CPU-FPGA hybrid sorted table design to maintain order books while maximizing the incoming message rate and minimizing the latency.

B. Related Work

Field Programmable Gate Arrays (FPGAs) become increasingly popular as the high-throughput and low-latency solutions in financial industries. Pottathuparambil et al. described an FPGA-based ITCH feed handler [8] to handle a peak data rate of 420 Mbps with a deterministic latency of 2.7μs. Subramoni et al. presented a prototype of an online OPRA data feed decoder [9], achieving a latency of less than 4μs. Lockwood et al. presented an FPGA IP library with networking, I/O, memory interfaces and financial protocol parsers [2]. Another popular topic of interest is algorithmic trading techniques like the acceleration of Monte Carlo simulations [3] [4] using FPGAs. These applications demonstrate FPGAs as an ideal hardware for low-latency financial packet processing, however, they mainly focus on parsing, decompressing, filtering and forwarding the market data feeds without the need of maintaining a large volume of sorted data items.

Activefeed MPU used an XtremeData FPGA accelerator which was placed in a processor socket of the host system and communicated using HyperTransport to accelerate market data forwarding. The latency was 100 us [10]. Morris et al. presented an FPGA accelerated approach for market data feed processing, using an FPGA connected directly to the network to parse, decompress, and filter the feed, and then to push the messages to CPUs, achieving an latency of 26μs [11]. These CPU-FPGA hybrid designs frequently transfer data between the CPU host and the FPGA, which largely reduces the latency gain of the network packet processing part.

Most reconfigurable designs for rebuilding order books are commercial and their implementation details are usually not presented. The Algo-Logic System provides a full order book with a maximum processing latency of less than 230 nanoseconds on a single FPGA Platform [12]. Miles et al. utilized a ternary tree to rebuild the market data from the Nasdaq stock exchange with a latency between 180-205ns [13]. Both of them rebuild the order books with the market data feeds from the exchanges while our design is targeting building order books in the exchanges, which involves more complex operations such as publishing market data feeds. The NovaSparks announces FPGA-based order book capability for global cash equities with the latency between 800-1500ns [14], which is much higher than our design.

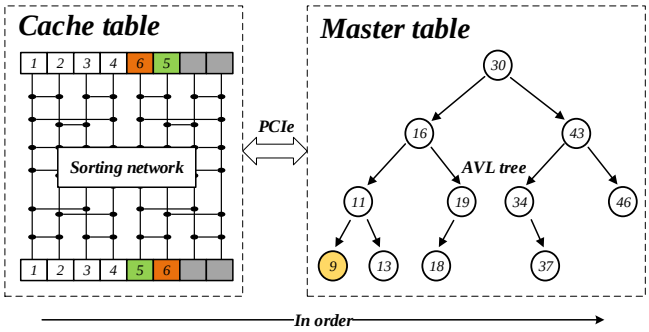

Figure 2. The data structure of the CPU-FPGA hybrid sorted table.

III. A CPU-FPGA Hybrid Sorted Table

A. Underlying Data Structure

The existing solutions to accelerate database-style operations usually store the data in the host memory, which often results in a long latency due to the data transfers [6]. We propose a CPU-FPGA hybrid sorted table, which consists of a cache table on the FPGA that stores MBs of most frequently used data items, and a master table at the CPU host that stores GBs of data items, as shown in Figure 2. The two tables work together seamlessly, with the following features.

- Both the cache table and the master table are sorted with different sorting strategies. Any low-latency sorting network algorithms such as the bitonic sorting [15] can be applied to sort the cache table. A balanced tree is usually employed to maintain the master table.
- A key of an item/order is either located at the cache table or the master table. The concatenation of the two tables forms a sorted super table that stores all data items in ascend/descend order.
- The most frequently used data items are stored in the cache table, leaving the rest to the master table.
- The two tables exchange information via PCIe and direct memory access (DMA) with a low-latency interface. We also design an efficient synchronization strategy that will be discussed in Section V.

A number of sorting algorithms on FPGAs are discussed in [15]. We employ the bitonic sort for the cache tables, which consists of $O(C_n \log(C_n)^2)$ comparators and has a delay of $O(\log(C_n)^2)$ cycles, where C_n is the size of the cache table. Usually, we set the size of the cache table in a range of 20 to 70, often resulting in a latency of a few cycles. An AVL tree [16] is applied for the master table. It has the advantage that the heights of the two child subtrees of any node differ by at most one. It takes $O(\log(M_n))$ time in both insertion and deletion where M_n is the size of the master table.

One of the flexibilities of separating the cache table and the master table is that we can use different strategies to maintain the master table according to different characteristics of real applications while still achieving a low latency with a few cycles. The CPU-FPGA hybrid hierarchical table provides us an effective way to maintain the top of the order book, which is the most frequently used data on the FPGA. Therefore, we are able to rebuild the order books in extremely low latency in a hardware accelerated way.

B. Design Methodology

The major difference between our design and previous work comes from the features of the sorted hierarchical table, which always keeps the hot data in the trading activities in the cache table. As the cache table is mapped to the FPGA, we can achieve the maximal performance and minimal latency. In this part, we discuss our considerations in designing the CPU-FPGA hybrid sorted table.

1) Why hot data is always in the cache table: In financial trading, traders are mainly interested in the top of the order book, which is the highest bid price and the lowest ask price. They signal the prevalent market and the bid/ask price needed to get an order fulfilled. If the bid on a particular instrument meets or exceeds the lowest ask price, a trade is matched and vice versa. So the top of the book is of the highest interest, followed by the second level, and the third level and so forth. As the two tables are sorted and the cache table is the head, the hot data is always in the cache table.

2) Why not keep all data in the cache table: Mapping the cache table to BRAM offers us great flexibilities and high memory bandwidth. However, the more elements we store in the BRAM, the more resources and latency we need to sort the cache table. In addition, the capacity of the BRAM is not able to hold GBs of order books of hundreds of instruments. Lastly, the traders and the match engine in the exchanges are only interested in the top of the order book, so putting them all in BRAM does not pay off.

3) Why not keep the master table in DRAM: In order to keep the cache table storing the top of the order book all the time, we need to transfer the largest/smallest elements of the master table to the cache table if necessary. Thus the elements in the master table also need to be sorted or partially sorted. Putting the master table to DRAM may meet the demand of stronger timing, however, it is extremely inefficient to insert, delete or sort elements on DRAM. On the other hand, such operations and related data structures are well developed at CPU host, so we employ the CPU to maintain the master table. The overhead comes from the data transfer latency through PCIe as well as the OS inherent overhead. So we carefully design a synchronization strategy to minimize the number of data transfers.

4) Why sorted table but not hash table or generic cache: We sort the orders mainly for two reasons. One is to provide an efficient way to access the top of the book when a trade matches or several levels of prices at the top needs to be published. The other reason is that a sorting algorithm has

Algorithm 1 Insert an order into the cache table

```
1:  procedure INSERT(k, v, C)                ▷ key, value, cache table
2:      f ← 0
3:      for c ∈ C do                         ▷ iterate cache table
4:          if c_k ≠ k then
5:              c_v ← c_v + 0
6:          else                             ▷ key is in the table
7:              c_v ← c_v + v                ▷ increase value (volume)
8:              f ← f + 1
9:      if f ≠ 0 then                        ▷ it is a new key
10:         C_{n-1} ← (k, v)                 ▷ insert new order
11:     bitonic_sort(C)                      ▷ keep table sorted
```

Algorithm 2 Delete an order from the cache table

```
1:  procedure DELETE(k, v, C)                ▷ key, value, cache table
2:      for c ∈ C do                         ▷ iterate cache table
3:          if c_k ≠ k then
4:              c_v ← c_v − 0
5:          else                             ▷ key is in the table
6:              c_v ← c_v − v                ▷ decrease value (volume)
7:          if c_v ≤ 0 then                  ▷ no volume at the price
8:              c_k ← null                   ▷ set key to invalid
9:      if size(C) ≤ N_c/4 then              ▷ not enough element
10:         C_{N/2..N} ← M_{1..N/2}          ▷ populate table
11:     bitonic_sort(C)                      ▷ keep table sorted
```

an amortized complexity and latency for different operations, which is important to the FPGA design because one FPGA always takes the longest path as its latency. Hashing is also a good method for elements lookup but it suffers from hash collision. Also, it is hard for hash table to track the top of the order book. A generic cache also suffers from the problem of tracking the top of the order book.

IV. Cycle-level Latency Routines

The underlying data structure and the algorithms are highly coupled. We first talk about the initialization process and a basic subroutine that are shared among different routines. The keys in the cache table are initialized to a specific value indicating they are invalid, denoted by the gray cell in Figure 2. They will automatically move to the end of the table after sorting.

One element is either in the cache table or in the master table, which provides us a way to filter an item to one of the table by comparing the key of the new item to M_s, where M_s is the smallest element in the master table, denoted by the yellow cell in Figure 2. If the key is less than M_s, the new item goes to the cache table, otherwise, it goes to the master table.

A. Cache Table Insertion

An item in the table is a (key, value) tuple. In the financial trading, the key and value are the price and volume of an order. If the key of the new order is larger than M_s, the new order will be inserted to the cache table following Algorithm 1. The result is to either increase the value if the corresponding key exists or add a new (key, value) tuple to the cache table if the key does not exist.

The arguments of the Algorithm 1 are the (key, value) tuple of the new order and the cache table. In line 3 to line 8, we iterate through the cache table C to determine if the key exists. If so, we increase the corresponding volume value according to the inserted new order. Otherwise, the values are unchanged. We need to insert the new item at the end of the cache table if the key of the new item does not exist. The process is described in line 9 to 10. Line 11 sorts the cache table so that the newly inserted item that is at the end of the table will automatically move to the right position. If the cache table becomes full after consecutive insertions, the second half of the table will be transferred to the master table at once so that the cache table has enough space for further insertions.

B. Cache Table Deletion

The deletion is the opposite of the insertion. While the insertion increases the corresponding value of a key, the deletion decreases it. The deletion algorithm is summarized in Algorithm 2.

From line 2 to line 8, we search the key by scanning the cache table and subtract the value from the total if the key belongs to the cache table. If the remaining amounts become zero after the subtraction, indicating that the item becomes invalid, the corresponding key should be removed from the cache table. We accomplish this by setting the key to the uninitialized value so that the key will go to the end of the table after sorting (line 7 and line 8).

If there are lots of continuous delete requests, the number of the valid items in the cache table may become smaller and smaller. Thus we need to refill a bulk of items from the master table (line 9 to 10), which is triggered when the number of valid items in the cache table falls below a user-defined threshold. We usually set the threshold to 25% of the total size of the cache table. No matter if we set a key to invalid or refill the cache table, the cache table will become sorted again (line 11).

C. Selection

The selection routine is to find an item from the table for given criteria. There are two major scenarios for a market data generator.

- Select an item with the smallest or largest key, which is a typical scenario in the financial trading for executing a trade where the smallest/largest key has the highest priority to be matched.

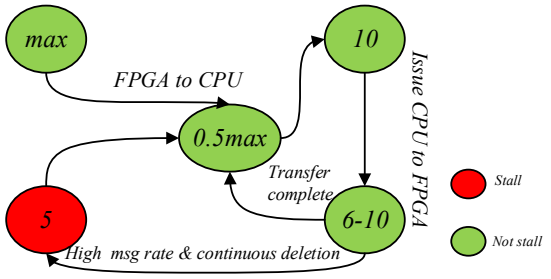

Figure 3. Data movements according to the size of the cache table.

- Select top portion of the order book, which happens twice in every second when broadcasting market data feeds.

In both scenarios, the sorted cache table guarantees that the items we need always stay at the head of the table. These operations can usually be accomplished with nearly zero latency. It is also one of the major reasons why we always keep the table sorted.

V. On-demand Synchronization

In our synchronization scheme, a data transfer is only triggered under certain user-defined conditions. Our goal is to minimize the number of transfers and maximally overlap the computation and communication.

A. FPGA to CPU

Transferring data from the FPGA to the CPU host will not stall the FPGA. Such data transfer is issued in two scenarios. One is when the cache table is full after the insertion, and the other is when the new item is filtered to the master table. In both scenarios, there is no reason for the FPGA to stall and wait for the completion of the data transfer. In case of a heavy burst of data transfer from the FPGA to the host memory, we design a large enough FIFO on the DRAM that buffers the data transferring to the CPU host so that the FPGA will not stall.

B. CPU to FPGA

A data transfer from the CPU host to the FPGA is triggered when the number of valid items in the cache table falls below a user-defined threshold, which happens in the deletion routine. Our strategy to overlap the communication and the computation is to transfer data in advance. For example, if we need to keep a minimal number of five elements in the cache table, we will issue a transfer of a bulk of data from CPU host when the number of elements falls below ten shown in Figure 3. Therefore, the logic will not stall immediately after requesting the data transfer. Figure 4 presents three communication & computation overlap scenarios.

1) When the message rate is low, the data transfer is finished before the end of the current packet processing or before the arriving of the next packet. The FPGA has enough time for communication.
2) Although the FPGA needs to process packets in a high message rate, the packets involve different operations instead of just deletions during the period of data transfer. The cache table has enough elements before the transfer is completed.
3) The FPGA needs to face a burst of packets involving only delete operations in a high message rate.

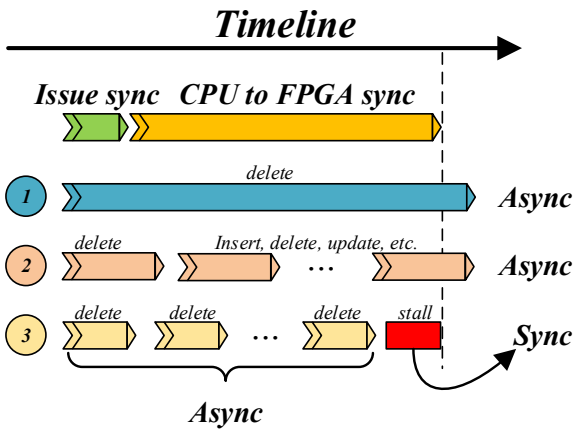

Figure 4. On-demand synchronization scheme in three different overlapping scenarios. The FPGA and the CPU are asynchronized in the first two scenarios. The FPGA only stalls for a short while with a series of deletions.

In the first two scenarios, the computation and data transfer are perfectly overlapped. However, in the last scenario, the computation and communication will first overlap when the cache table still has enough items, and the FPGA may stall for a short while before the data transfer is finished, which is denoted by the red bar in Figure 4.

In most trading activities, a data transfer is completed without the FPGA stalling because there are a lot of other actions such as insertions and selections. A strict synchronization usually occurs when there are a series of deletions, which rarely happens. In order to reduce the possibility of stalling, we can set a higher threshold to fetch data from the CPU host. For example, we can issue data transfers once the number of the valid items falls below 50% of the total size of the cache table.

In summary, the FPGA does not stall when the data moves from the cache table to the master table. For data movements from the CPU to the FPGA, we can adjust the threshold that triggers the data transfer to minimize the possibility of the FPGA stalling. In our three test cases, the possibility is less than 0.01%.

VI. Experiments

We map our design to the Maxeler MAX4 ISCA data flow engine. It is an FPGA-based network card that has two NICs supporting high-speed connections. We program it

using a high-level synthesis programming model, the MaxJ language. Our hardware design runs at 160MHz. As the frequency of the NIC module is 156MHz, running our design at 160MHz is fast enough even for processing packets at a full bandwidth (10Gb/s).

The FPGA card is then integrated into the market data generator (MDG) as shown in Figure 1. The MDG runs in the test environment of the China Financial Futures Exchange (CFFEX). The CFFEX also provides us three data sets, each containing all packets (100+ million packets and 200+ order books) of one trading day, to measure the performance of our design.

A. Performance Model

FPGAs have many difficulties for low latency measurements, and the high throughput specifically, poses a number of challenges. We propose a performance model for measuring the latency and projecting the maximum throughput. We follow Equation 1 to measure our latency.

$$L_{obs} = L_{total} - L_{const} - L_{pkg} \tag{1}$$

where L_{obs} is the observed latency of our design; L_{total} is the total measured latency of MDG; L_{const} is the constant latency without our design; L_{pkg} is the latency of parsing the package calculated by Equation 2.

$$L_{pkg} = M_{size}/(W_{data} \times F_{NIC}) \tag{2}$$

where M_{size} is the size of the message; W_{data} is the byte width of the data pth; F_{NIC} is the running frequency of NIC.

The time remained for updating the order book, $L_{order-book}$, as a function of message rate, M_{rate}, can be expressed as Equation 3 and 4.

$$L_{order-book} = L_{proc}/M_{rate} - L_{const} \tag{3}$$

$$L_{proc} = 1 - (M_{rate} \times L_{pkg}) \tag{4}$$

Therefore, the maximum message rate that our design can support theoretically can be expressed as Equation 5.

$$M_{max-rate} = L_{order-book}/L_{obs} \times M_{rate} \tag{5}$$

The above equations will be used in exploring future performance scalability in Section VI-E.

B. Latency vs. Cache Table Size

We maintain the top of the order book in the cache table on the FPGA. Table I shows the cycles we need for different routines as the size of the cache table varies. A larger cache table capacity always results in a longer latency. Another factor to the latency comes from the data transfer between the FPGA and CPU host. Figure 5 presents the number of cycles used and the number of data transfer from FPGA to CPU and vice versa according to different cache table sizes.

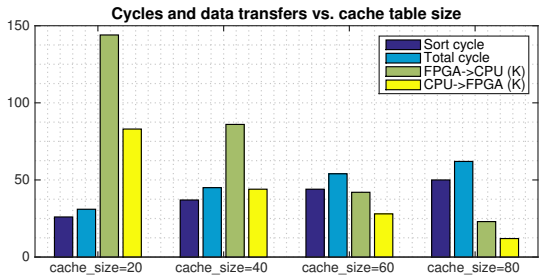

Figure 5. The number of cycles and the number of data transfers (in thousand) between the FPGA and the CPU host when the size of cache table varies. The larger the cache table is, the larger the average latency and the less number of data transfers we have.

Table I
CYCLES OF DIFFERENT ROUTINES VS. CACHE TABLE SIZE

Size	Overall	Insert	Delete	Select	Publish
20	31	27	24	20	20
40	45	38	36	31	31
40	54	45	42	37	37
80	62	51	47	43	43

The theoretical depth of bitonic sort is $O\log(N_c^2)$, so increasing the size of the cache table also increases the latency of our design. On the other hand, the number of data transfers between the FPGA and CPU host reduces as the cache table size increases. The less frequently the data transfers, the less possibility that the FPGA waits. So there is a balance between the average latency without stalling and the peak latency when the FPGA needs to wait data from the CPU host.

It is hard to derivate an optimal cache table size because the data transfer between the FPGA and the CPU highly depends on the sequence of the packets. Different actions result in different data movement behaviors. In our experience, setting the cache table size to 50-70 results in a steady latency of 400-500ns in most cases.

C. Latency vs. Message Rate

Figure 6 shows the exchange of the cache table size and the latency with a sequence of packets in different message rates. First, we can observe how the cache table size varies from the blue curve. When the size hits 50, it drops to 25 immediately because 25 elements in the cache table are moved to the CPU host (master table). Also notice that, the latency in such situation does not change because the FPGA is not necessary to wait for the completion of the data transfer.

When the size of the cache table falls below 10, the data moves from the master table to the cache table, so the cache table is filled up with 25 items. The FPGA will not stall immediately, instead, the FPGA keeps processing until there

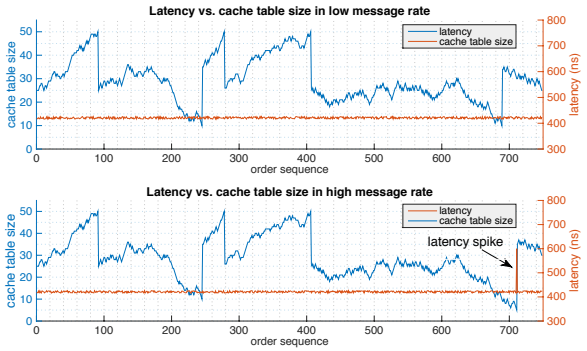

Figure 6. The behaviors of the cache table size and the latency in low message rate (1Gb/s) and high message rate (10Gb/s) when replaying the same packets. The top figure shows that our design achieves a steady and constant latency in low message rate, and the bottom figure shows that our design suffers a latency spike when moving data from the CPU host to the FPGA.

Table II
THE LATENCY (NS) AND POSSIBILITY IN DIFFERENT CACHE SIZE

Cache size	≤ 5		6-10	>10
Latency (ns)	1001-1300	701-1000	501-700	400-500
Possibility	0.1%	<2%	5%-8%	>90%

are only 5 elements in the cache table. We can observe that in the scenario of low message rate (top figure), the latency is still steady because there is enough time for the data movements. In high message rate, the latency is still steady the first time the cache table refills (at the 245th order), but we suffer a latency spike at the 711th order. We consider it reasonable because there are a sequence of delete operations in high message rate. Notice that the cache table keeps shrinking in the second data movement while it grows in the first data movement.

D. Latency Distribution and Speedup

In order to have a better understanding of the latency of the design in real cases, we summarize the latency distribution by replaying the three data sets in Figure 7 and Table II. In most situation like inserting/selecting orders to/from the order book, the latency is between 400-450ns. When the FPGA stalls, the latency usually ranges from 500-100ns. Only in less than 0.1% will the latency go up to 1000+ns. It is reasonable because the latency is highly related to the different patterns of incoming packets.

We also compare the latency of our design with the one from NovaSparks [14] and the original CPU-based solution, which is summarized in Table III. The last two columns records the speedup values of our FPGA-based solution over the CPU-based solution and the NovaSparks' solution. Our design can achieves 80 to 125 times and 1.1 to 2.2 times speedup in terms of latency over the highly optimized CPU-based solution and the NovaSparks' FPGA solution.

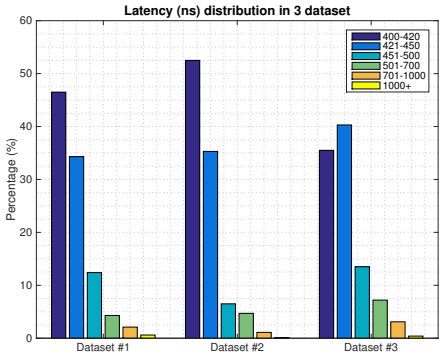

Figure 7. Latency distribution in three data sets. 97.3%, 98.8%, 96.5% of latencies are between 400-700ns in three cases respectively.

Table III
LATENCY COMPARISON WITH NOVASPARKS

CPU	NovaSparks	FPGA	SU_{cpu}	SU_{ns}
40-60us	880-1500ns	400-1300ns	80-125x	1.1-2.2x

E. Future Performance Scalability

Based on the proposed performance model, we can project the maximum throughput our design can support, which meets the requirements of future technologies.

Given the message rate $M_{rate} = 40,000msg/s$, $M_{size} = 1024B$, $L_{obs} = 450ns$, and $L_{const} = 700ns$ for the test cases in the environment of CFFEX, the maximum message rate we can support is $M_{maxrate} = 2,080,000msg/s$ according to equation 2 to 5, which is 52 times over the current message rate in CFFEX.

F. Resource Usage

The characteristic of the hybrid sorted table enables the hardware MDG to support a large number of order books. Setting cache table size to 50, Figure 8 shows how the different logic resource usages vary as the number of order books increases. The LUTs, FFs, and DSPs stay constantly when the number of order books increases because different routines are shared among different order books. Only BRAMs grow linearly. The usage of LUTs, FFs, and DSPs increases dramatically when the size of cache table increases, because the sorting is the major cause of the resource increment.

In our experiment, the logic resources on one FPGA are able to support 200+ order books, each with a size of 80.

VII. CONCLUSIONS

FPGAs have been proved to be an established technology for financial applications. However, organizing the packets into structural information with complicated routines is challenging in both designing and resources. Existing solutions usually leave the post-processing part to the CPU,

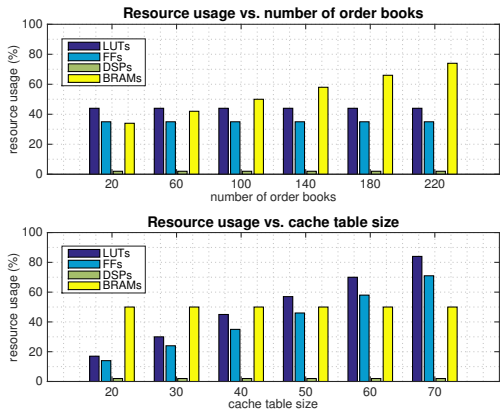

Figure 8. The top figure shows the relationship between the logic resource usage and the number of instruments. The bottom figure shows the relationship between the logic resource usage and the size of the cache table.

which largely reduces the latency gain of the network packet processing part. In contrast, this paper proposes a CPU-FPGA hybrid sorted table design that supports maintaining the storage of GB-level of order books while providing nanosecond-level latencies. We first use a hierarchical sorted table with two levels, a cache table on the FPGA storing the most frequently used data (the top of the order book) and a master table at the host memory keeping the rest. Then we design a complete set of general-purpose routines with a latency of a few cycles. The two tables communicate via an on-demand synchronization scheme. Experiments show that 98% of the packets have latencies between 400-700ns, being an 80- to 125-fold reduction compared with the fully optimized software solution, and 2.2-fold reduction over an existing FPGA-based solution.

Current and future work includes optimizing the proposed approach to minimize latency while enhancing security for the entire system, and exploring the automation of building block optimization.

ACKNOWLEDGMENT

This work was supported in part by the National Key & D Program of China (Grant No. 2016YFA0602200), the National Natural Science Foundation of China (Grant No. 4137411, 91530323), the China Postdoctoral Science Foundation (No. 2016M601031), the European Union Horizon 2020 Research and Innovation Programme under grant agreement number 671653, UK EPSRC (EP/I012036/1, EP/L00058X/1, EP/L016796/1 and EP/N031768/1), Maxeler and Intel Programmable Solutions Group.

REFERENCES

[1] S. Denholm, H. Inoue, T. Takenaka, T. Becker, and L. Wayne, "Network-Level FPGA Acceleration of Low Latency Market Data Feed Arbitration," *IEICE TRANSACTIONS on Information and Systems*, 2015.

[2] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, "A low-latency library in FPGA hardware for high-frequency trading (HFT)," in *IEEE 20th Annual Symposium on High-Performance Interconnects*, 2012.

[3] N. A. Woods and T. VanCourt, "FPGA acceleration of quasi-Monte Carlo in finance," in *International Conference on Field Programmable Logic and Applications*, 2008.

[4] S. Wray, W. Luk, and P. Pietzuch, "Exploring algorithmic trading in reconfigurable hardware." in *ASAP*, 2010.

[5] J. Casper and K. Olukotun, "Hardware acceleration of database operations," in *Proceedings of ACM/SIGDA international symposium on Field-programmable gate arrays*, 2014.

[6] B. West, R. D. Chamberlain, R. S. Indeck, and Q. Zhang, "An FPGA-based search engine for unstructured database," in *Proc. of 2nd Workshop on Application Specific Processors*, 2003.

[7] D. Koch and J. Torresen, "FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011.

[8] R. Pottathuparambil, J. Coyne, J. Allred, W. Lynch, and V. Natoli, "Low-latency FPGA based financial data feed handler," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2011.

[9] H. Subramoni, F. Petrini, V. Agarwal, and D. Pasetto, "Streaming, low-latency communication in on-line trading systems," in *International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.

[10] "Activefeed mpu: Accelerate your market data," *http://www.activfinancial.com/docs/ActivFeedMPU.pdf*, 2007.

[11] G. W. Morris, D. B. Thomas, and W. Luk, "FPGA accelerated low-latency market data feed processing," in *17th IEEE Symposium on High Performance Interconnects*, 2009.

[12] "Full order book." [Online]. Available: http://algo-logic.com/orderbook

[13] "A Full-Hardware Nasdaq Itch Ticker Plant on Solarflares AoE FPGA Board." [Online]. Available: http://www.cs.columbia.edu/ sedwards/classes/2013/4840/reports/Itch.pdf

[14] NovaSparks, "NovaSparks Announces FPGA-based Order Book Capability for Global Cash Equities." [Online]. Available: http://www.novasparks.com/news-and-events/press-releases/novasparks-announces-fpga-based-order-book-capability-for-global-cash-equities.html

[15] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of Spring Joint Computer Conference*, 1968.

[16] M. AdelsonVelskii and E. M. Landis, "An algorithm for the organization of information," DTIC Document, Tech. Rep., 1963.