Efficient Event Processing through Reconfigurable Hardware for Algorithmic Trading

Mohammad Sadoghi, Martin Labrecque, Harsh Singh, Warren Shum, Hans-Arno Jacobsen
University of Toronto

ABSTRACT

In this demo, we present *fpga-ToPSS* (Toronto Publish/Subscribe System Family), an efficient event processing platform for high-frequency and low-latency algorithmic trading. Our event processing platform is built over reconfigurable hardware—FPGAs—to achieve line-rate processing. Furthermore, our event processing engine supports Boolean expression matching with an expressive predicate language that models complex financial strategies to autonomously buy and sell stocks based on real-time financial data.

1. INTRODUCTION

Algorithmic trading is a computer-based approach to execute buy and sell orders on financial instruments such as securities (e.g., stocks, bonds, and options.) Financial brokers exercise investment strategies using autonomous high-frequency algorithmic trading fueled by real-time market events (e.g., stock & news feeds.) Algorithmic trading is dominating financial markets and now accounts for over 70% of all trading in equities [5]. Therefore, as the computer-based trading race among major brokerage firms continues, it is crucial to optimize execution of buy or sell orders at the microsecond level in response to market events, such as corporate news, recent stock price patterns, and fluctuations in currency exchange rates, because every microsecond translates into opportunities and ultimately profit [5]. Consider a classical arbitrage strategy with an estimated annual profit of $21 billion according to TABB Group [6]: Barrick Gold stock (TSE:ABX) is trading at $\$40.04$ a share in Toronto while (NYSE:ABX) is trading at $\$40.05$ in New York. Therefore, to take advantage, a high-frequency algorithmically driven strategy must quickly respond by buying in Toronto and selling in New York before the price gap closes [3]. Every 1-millisecond reduction in response-time is estimated to generate the staggering amount of over $100 million a year [10].

Algorithmic trading is naturally modeled by an event processing platform in which financial news and market data are captured as events such as: [*stock* = ABX, $TSX_{ask} = 40.04$, $NYSE_{ask} = 40.05$]; while investment strategies are formulated by financial institutions and brokers in the form of subscriptions such as: [*stock* = ABX, $TSX_{ask} \neq NYSE_{ask}$] or [*stock* = ABX, $TSX_{ask} \leq 40.04$].

Proceedings of the VLDB Endowment, Vol. 3, No. 2

Therefore, a scalable event processing platform must efficiently find all investment strategies (subscriptions) that match incoming market events at a high rate, up to a million market events per second [2]. Therefore, to achieve the desired event processing throughput, we propose a novel FPGA-based solution to significantly speed up algorithm trading computations, namely, market event parsing and market event matching against strategies. We experiment with three novel approaches exploring the trade-off between flexibility in programming versus performance gained in data processing with FPGAs (Field Programmable Gate Arrays.)

Another key realization in algorithmic trading is that generally investment strategies are developed using extensive data analysis. Most important, in high-frequency trading in which orders are fulfilled in order of microseconds, a miscalculated and a rushed strategy could result in the loss of millions of dollars within seconds. Therefore, the typical high-frequency trading strategy tends to be consistent over time, and even the highly competitive micro-level investment strategy has an average shelf life on the order of days [6]. As a result, many investment strategies are relatively long-lived and are not rapidly updated. Thus, in addition to using hardware acceleration to improve matching and parsing, we can encode parts or all of the strategies into custom hardware blocks for superior performance and seamlessly re-synthesize, on the order of minutes, relevant components as strategies change.

Why Use FPGAs? Meeting the demand of data processing at current network bandwidths, which has a tendency to double in approximately 9 to 10 months (Gilders Law), is becoming increasingly challenging. This up-trend of network bandwidth is quite evident in the trading network capacity (1-million+ messages per second) [2] which is required to support the large fraction of trading involving algorithmic techniques (over 70%) on today's markets [5]. On the other hand, supply of affordable line-rate processing solutions is becoming scarce as the trend of up-scaling of transistor densities and higher clock frequencies in commodity CPUs as predicted by Moore's Law is now flattening due to the physical limitations of current semiconductor fabrication technology. Thus, there is a keen interest in the research community and companies alike [Celoxica, Exegy, RedLine] to rely on FPGA-based computing solutions for applications where deterministic, multi-gigabit processing throughput and low-latency in forwarding of mission critical data is required. The true success of FPGAs is rooted in three distinctive features: hardware parallelism, hardware reconfigurability, and substantially higher throughput packet processing.

Parallelism & Reconfigurability The ability of an FPGA to be re-configured on-demand into a custom hardware circuit with a high degree of parallelism is key to its advantage over CPUs for data and event processing solutions. Using a powerful multi-core CPU system does not necessarily increase processing rate (Amdahl's

Law) as it increases inter-processor signaling and message passing overhead, often requiring complicated concurrency management techniques at the program and OS level. On the other hand, FPGAs allow us to get around these limitations due to their intrinsic highly inter-connected architecture and the ability to create custom logic on the fly to perform parallel tasks [11, 13]. In our design, we exploit parallelism, owing to the nature of the matching algorithm (Sec. 3), by creating multiple matching units which work in parallel with multi-giga bit throughput rates (Sec. 5), and we utilize reconfigurability by seamlessly adapting relevant components as investment strategies evolve (Sec. 5).

Packet Processing Another benefit of an FPGA-based solution is that there are multiple high bandwidth (giga-bit) I/O pins that allow these devices to be inserted into the high data-rate streams without added latency on the outgoing traffic. In modern server systems, however, there is the additional OS layer latency overhead in moving data between input and output ports (Sec. 7).

2. RELATED WORK

FPGA An FPGA is a semiconductor device with programmable lookup-tables (*LUTs*) that are used to implement truth tables for logic circuits with a small number of inputs (on the order of 4 to 6 typically). FPGAs may also contain memory in the form of flip-flops and block RAMs (BRAMs), which are small memories (on the order of a few kilobits), that together provide a small storage capacity but a large bandwidth for circuits in the FPGA. Thousands of these building blocks are connected with a programmable interconnect to implement larger-scale circuits.

Past work has shown that FPGAs are a viable solution for building custom accelerated components [13]. For instance, [11] demonstrates a design for accelerated XML processing. Work in [12] shows an FPGA solution for processing market feed data while our work concentrates on an event processing platform to accelerate the matching computation of strategies in algorithmic trading. Lastly, [14] presents a framework to use FPGAs for data stream processing as co-processors in many-core architectures (including CPUs) while our entire architecture is built directly on FPGAs.

Matching The matching is one of the main computation intensive components of event processing which has been well studied over the past decade (e.g., [1, 4]). In general, the matching algorithms are classified as (1) counting-based [4], and (2) tree-based [1]. The counting algorithm is based on the observation that investment strategies tend to share many common predicates; thus, the counting method minimizes the number of predicate evaluations by constructing an inverted index over all unique predicates. Similarly, the tree-based methods are designed to reduce predicate evaluations; in addition, they recursively cut through space and eliminate strategies on the first encounter with an unsatisfiable predicate. The counting- and tree-based approaches can be further classified as either key-based (in which for each strategy a set of predicates are chosen as identifiers [4]), or as non-key based [1]. In general, the key-based methods reduce memory access, improve memory locality, and increase parallelism, which are essentials for a hardware implementation. The most prominent key-based matching algorithm is Propagation [4].

3. EVENT PROCESSING MODEL

Subscription Language & Semantics The matching algorithm takes as input an event (e.g., market event, stock feed) and a set of subscriptions (e.g., investment strategies) and returns matching subscriptions. The event is modeled as a value assignment to attributes and the subscription is modeled as a Boolean expression (i.e., as conjunction of Boolean predicates.) Each Boolean predicate is a triple of either [attribute$_i$, operator, values] or [attribute$_i$, operator, attribute$_j$]. Formally, the matching problem is defined as follows: *given a market event e and a set of financial strategies, find all strategies s_i satisfied by e.*

Matching Algorithm The Propagation algorithm is a state-of-the-art key-based counting method that operates as follows [4]. First, each strategy is assigned a key (a set of predicates) based on which a multi-attribute hashing scheme uniquely assigns strategies into a set of disjoint clusters. Second, keys are selected from a candidate pool using a novel cost-based optimization tuned by the workload distribution to minimize the matching cost [4]. The Propagation data structure has three main strengths which makes it an ideal candidate for a hardware-based implementation: (1) strategies are distributed into a set of disjoint clusters which enables highly parallelizable event matching through many specialized hardware matching units, (2) within each cluster, strategies are stored as contiguous blocks of memory which enables fast sequential access and improves memory locality, and (3) the strategies are arranged according to their number of predicates which enables prefetching and reduces memory accesses and cache misses [4].

4. DESIGN OVERVIEW

Commodity servers are not quite capable of processing market event data at line rates. The alternative for financial institutions is to acquire and maintain high cost purpose-built network appliances. In contrast, our design uses an FPGA, to significantly speed up algorithm trading computations involving market event parsing and market event and strategy matching. FPGAs offer a cost effective algorithmic trading solutions, since custom hardware can be altered and scaled to adapt to the prevailing load and throughput demands. Hardware reconfigurability allows FPGAs to house *soft processors*—processors composed of programmable logic. A soft processor has several advantages: it is easier to program than (e.g., using `C` as opposed to `Verilog` which requires specialized knowledge and hardware development tools), it is portable to different FPGAs, it can be customized, and it can be used to communicate with other components and accelerators in the design. In this project, the FPGA resides on a NetFPGA [9] network interface card and communicates through DMA on a PCI interface to a host computer. FPGAs have programmable I/O pins that in our case provide a direct connection to memory banks and to the network interfaces, which in a typical server, are only accessible through a network interface card. In this section, we describe the three designs that we are implementing. Not described is a PC-based version, which serves as baseline for our demo and experiments. It uses exactly the same C-based matching code as the first approach.

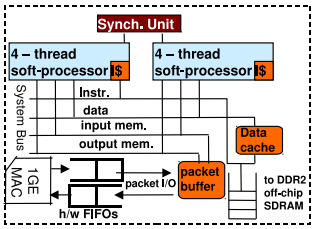

Figure 1: Soft-processor(s)-based implementation

Tuning for Flexibility Our first approach is the Soft-processor(s)-based solution (cf. Fig. 1), which runs on a custom soft processor that is implemented on the NetFPGA platform. This solution also runs the same C-based strategy matching code that is run on the PC-based version (our baseline); thus, this design is the easiest to evolve as message formats and protocols change. In order to maximize throughput of our event processing application, we chose NetThreads [7] as the baseline soft processor platform for the FPGA. NetThreads has two single-issue, in-order, 5-stage, 4-way multi-threaded processors (cf. Fig. 1), shown to deliver more

throughput than simpler soft processors [8]. In a single core, instructions from four hardware threads are issued in a round-robin fashion to hide stalls in the processor pipeline and execute computations even when waiting for memory. Such a soft processor system is particularly well-suited for event processing: The soft processors suffer no operating system overhead compared to conventional computers, they can receive and process packets in parallel with minimal computation cost, and they have access to a high-resolution system clock (much higher than a PC) to manage timeouts and scheduling operations. One benefit of not having an operating system in NetThreads is that packets appear as character buffers in a low latency memory and are available immediately after being fully received by the processor system (rather than being copied to a user-space application). Also, editing the source and destination IP addresses only requires changing a few memory locations, rather than having to comply with the operating system's internal routing mechanisms. Because a simpler soft processor usually executes one instruction per cycle, it suffers from a raw performance drawback compared to custom logic circuits on FPGAs. A custom circuit can execute many operations in parallel.

Tuning for Performance Our second approach (cf. Fig. 2), is a purely hardware solution: custom hardware components perform necessary steps involving market event parsing and matching of market event data against strategies. This method provides the highest performance, but also involves a higher level of complexity in integrating custom heterogeneous accelerators in which both, the performance-critical portion of the event processing algorithm and trading strategies reside (encoded) within the design of the matching unit logic thereby completely eliminating all on and off-chip memory access latencies. Moreover, discretization of financial markets means that the minimum price variation of stock quote prices is discretized ($0.01 in the US) such that prices can be represented as scaled integer comparisons in hardware. This allows for further optimization of hardware resources to accommodate more matching units, that would otherwise be required to support floating point comparisons, which is particularly cumbersome in FPGAs.

The Hybrid Approach Due to increased complexity in the hardware-only approach to support dynamic strategies and variable event data formats, we implemented a third scheme (cf. Fig. 3), which is a hybrid of the previous two. Since FPGAs are normally programmed in a low-level hardware-description language, it would be complex to support a flexible communication protocol. Instead, we instantiate a soft processor to implement the packet handling in software. After parsing incoming market event data packets, the soft processor offloads the bulk of the strategy matching to a dedicated custom hardware matching unit. Unlike the strategy-encoded matching units used in the hardware-only solution, these matching units use small low-latency on-chip memories, *Block RAMs* available on FPGAs, that can be stitched together to form larger dedicated blocks of memory. The FPGA on the NetFPGA platform [9] has 232 18kbit Block RAMs which are utilized to cache a small subset of static strategies that do not change very often. Having an on-chip strategy data cache allows market event matching to be initiated even before the off-chip strategy data can be accessed. However, our matching algorithm leverages data locality in the storage of dynamic strategies, which may be updated during run time, in contiguous array clusters thereby exploiting burst-oriented data access feature of the DDR2 (SDRAM), off-chip memory, while fetching the strategy data clusters.

5. IMPLEMENTATION OVERVIEW

This section provides more details of the proposed two *fpga-ToPSS* architectures: the hardware-only and hybrid implementations.

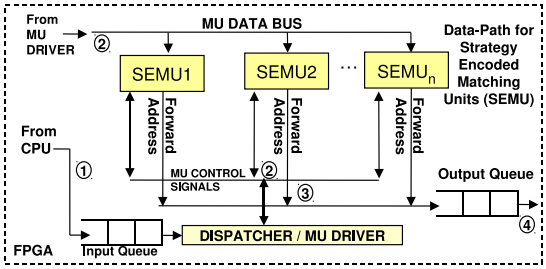

Figure 2: Hardware-only implementation

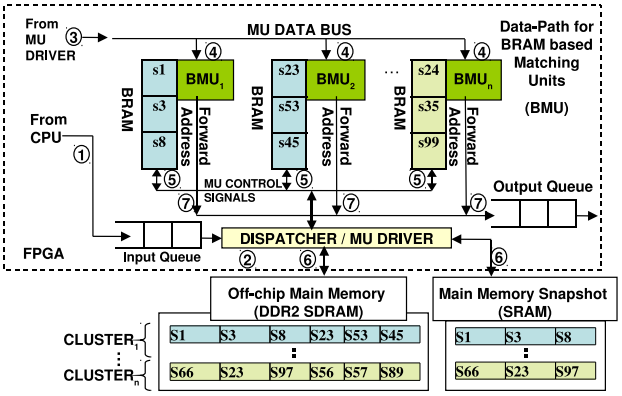

Figure 3: Hybrid implementation

Hardware-only Implementation Our *fpga-ToPSS* hardware-only implementation offers the highest possible rate at which incoming market events can be matched against investment strategies (subscriptions) that have been encoded in the STRATEGY ENCODED MATCHING UNIT (SEMU) logic on the FPGA. This method avoids the latency of both on and off-chip memory access, but significantly constraints the size of the strategy base that can be supported. A diagram of this design is shown in Fig. 2. This setup is massively parallelized and offers strategy matching at extremely high rates (i.e. one clock cycle).

The stepwise operation of the hardware-only implementation is depicted in Fig. 2. In this design, the soft processor only serves to transfer (1) the received market event data packets from the network interface input buffer to the input queue of the our hardware-only system. Custom hardware submodule, the DISPATCHER unit, parses (2) the incoming market events and feeds the current market event data to all the matching units while generating all the necessary control signals to run these units synchronously. Each unit is able to match all encoded strategies against the current market (publication) event in one clock cycle. However, subsequent clock cycles are spent in tallying the matches and preparing the final responses (e.g. forward address look-up or consolidating system wide match counts) that is eventually pushed (3) into the output queue. The soft processor then transfers (4) the final result from the output queue to the network interface to be sent to the intended host(s).

Hybrid Implementation With our *fpga-ToPSS* hybrid implementation, we employ a more generalized design that enables the matching units to support a dynamic and a larger strategy data set than can be supported in the hardware-only implementation. Unlike the SEMUs (strategies encoded within hardware), the BRAM-based Matching Units (BMUs) allow static strategies to be stored on the on-chip dedicated low latency Block Rams (BRAMs); thus making the design less hardware resource intensive compared to the hardware-only implementation. We adopt two approaches to reduce the impact of off-chip memory data access latency on the

overall system throughput. Firstly, we take advantage of the high degree of data locality inherent in Propagation's data structure which helps to minimize random access latency. Secondly, fast (single cycle latency) but smaller capacity BRAMs are dedicated for each matching unit to store static strategies, which helps mask the initial handshaking setup delay associated with off-chip main memory access, i.e., the market event matching can begin against static strategies as soon as the event arrives; in the meantime the system prepares to setup data access from the off-chip DDR2 main memory.

The stepwise operation of the hybrid system is depicted in Fig. 3. Upon arrival of a market event, the soft processor transfers (1) the data packet to the input queue of the hybrid system. A custom hardware submodule, the DISPATCHER unit, extracts strategy predicates-value pairs, which are input to hash functions to generate cluster addresses. Cluster addresses are used to look-up the memory locations (2) of the relevant strategy clusters residing both in BMU BRAMs and in off-chip main memory. The DISPATCHER then feeds the market event (3) and previously computed cluster addresses (4) on the MU DATA BUS (common to all BMUs). Next, the DISPATCHER unit activates all parallel BMUs to initiate matching (5) using on-chip static strategies stored in each BMU, while simultaneously queuing up read requests for the off-chip main memory. The transfer of dynamic strategy data between the BMUs is pipelined to avoid stalling the matching units due to lack of data. Finally matched strategies are placed (7) into the output queue.

6. DEMO METHODOLOGY

This section describes our demo setup including the hardware used to implement our FPGA-based algorithmic trading solution and the measurement infrastructure.

Demo Platform Our FPGA based solutions are instantiated on the NetFPGA 2.1 [9] platform, operating at 125MHz and have access to four 1GigE Media Access Controllers (MACs) via high-speed hardware FIFO queues (cf. Fig. 1) allowing a theoretical 8Gbps of concurrently incoming and outgoing traffic capacity. In addition, a memory controller to access the 64 Mbytes of on-board off-chip DDR2 SDRAM is added. The system is synthesized to meet timing constraints with the Xilinx ISE 10.1.03 tool and targets a Virtex II Pro 50 (speed grade 7ns). Our soft processor and matching units run at the frequency of the Ethernet MACs (125MHz),

Demo & Evaluation Setup For our experiments, we use HP DL320 G5p servers (Quad Xeon 2.13GHz) equipped with an HP NC326i PCIe dual-port gigabit network card running Linux 2.6.26. As shown in Figure 4, we exercise our algorithmic trading solutions from the server executing a modified `Tcpreplay 3.4.0` that sends market event packet traces at a programmable fixed rate. Packets are timestamped and routed to either the FPGA-based solutions (Setup 1) or PC-based (Setup 2) solution. Setup 1 is configured as one of the solutions described in Sec. 4 and Setup 2 is a baseline serving as comparison only. The network propagation delays are similar for either solution. Both FPGA-based or PC-based solutions forward market events on the same wire as incoming packets which allows the Event Monitor (cf. Fig. 4) to capture both incoming and outgoing packets from both setups. The EM provides a 8ns resolution on timestamps and exclusively serves for the measurements.

Demo Workload We generate a workload of tens of thousands of subscriptions derived from investment strategies such as arbitrage and buy-and-hold. In addition, we generate market events using the Financial Information eXchange (FIX) Protocol with FAST encoding (cf. `fixprotocol.org`).

Demo Measurements We characterize the system throughput

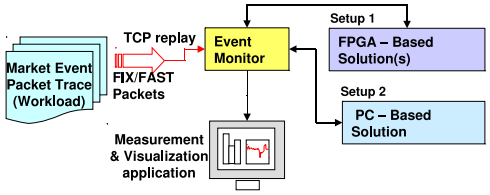

Figure 4: Demo Setup

	PC	Soft-Processor	Hybrid	Hardware-only
250	53.94	71.09	6.47	3.22
1K	60.77	199.43	7.56	N/A
10K	150.02	1,617.85	87.82	N/A
100K	2,001.29	16,422.87	1,307.34	N/A

Table 1: End-to-end System Latency (μs)

as the maximum sustainable input packet rate obtained by determining, through a bisection search, the smallest fixed packet inter-arrival time where the system drops no packets. The latency of our solutions is the interval between the time a market event packet leaves the Event Monitor output queue to the time the first forwarded version of the market event is received and is added to the output queue of the Event Monitor.

7. CONCLUSIONS

Our *fpga-ToPSS* framework is built on NetThreads's soft processors, which is optimized for throughput and latency because of the multiple threads executing in round-robin fashion that enables execution of two instructions in parallel per cycle [7]. Furthermore, our custom matching units provide the ability to match many strategies in parallel in addition to providing hardware acceleration. Lastly, by eliminating the operating system, the FPGA-based solution provides a superior end-to-end system performance. In Table 1, we demonstrate the system latency as workload (investment strategies) size changes from 250 to 100K; a similar trend was also observed for the system throughput which is omitted due to lack of space. In summary, even though our FPGA chip (125MHz Virtex II) is much slower than the latest FPGA (800MHz Virtex 6) and significantly slower than our CPU (Quad Xeon 2.13GHz), our hybrid solution outperformed the PC-based solution by upto an order of magnitude, and our hardware-only, while currently feasible only for smaller workloads due to lack of resources on the FPGA, improved the latency by nearly two orders of magnitude.

8. REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC'99*.

[2] J. Corrigan. Updated traffic projections. *OPRA, March'07*.

[3] J. Daily. There's millions in those microseconds. *The Globe & Mail, 29/1/10*.

[4] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for fast pub/sub systems. *SIGMOD'01*.

[5] K. Heires. Budgeting for latency: If I shave a microsecond, will I see a 10x profit? *Securities Industry, 1/11/10*.

[6] R. Iati. The real story of trading software espionage. *TABB Group Perspective, 10/07/09*.

[7] M. Labrecque et al. NetThreads: Programming NetFPGA with threaded software. In *NetFPGA Dev. Workshop'09*.

[8] M. Labrecque and J. G. Steffan. Improving pipelined soft processors with multithreading. In *FPL'07*.

[9] J. W. Lockwood et al. NetFPGA - an open platform for gigabit-rate network switching and routing. In *MSE'07*.

[10] R. Martin. Wall street's quest to process data at the speed of light. *Information Week, 4/21/07*.

[11] A. Mitra et al. Boosting XML filtering with a scalable FPGA-based architecture. *CIDR'09*.

[12] G. W. Morris et al. FPGA accelerated low-latency market data feed processing. *IEEE 17th HPI'09*.

[13] R. Mueller, J. Teubner, and G. Alonso. Data processing on FPGAs. *VLDB'09*.

[14] R. Mueller, J. Teubner, and G. Alonso. Streams on wires: a query compiler for FPGAs. *VLDB'09*.