Demo: fpga-ToPSS – Line-speed Event Processing on FPGAs

Mohammad Sadoghi, Harsh Singh, Hans-Arno Jacobsen
Middleware Systems Research Group
Department of Electrical and Computer Engineering
University of Toronto, Canada
mo@cs.toronto.edu,{harshvps,jacobsen}@eecg.toronto.edu

ABSTRACT

In this demo, we present *fpga-ToPSS* [1](a member of 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 mimic the buying and the selling of stocks based on real-time financial data.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Measurement, Experimentation, Performance

Keywords

FPGA, Publish/Subscribe, and Complex Event Processing

1. INTRODUCTION

Efficient event processing is an integral part of growing number of data management technologies such as algorithmic trading [8], real-time data analysis [10], intrusion detection system [3], and (complex) event processing (e.g., [1, 2, 7]).

A prominent application for event processing is algorithmic trading; a computer-based approach to execute buy and sell orders on financial instruments (e.g., securities). Financial brokers exercise investment strategies (*subscriptions*) using autonomous high-frequency algorithmic trading fueled by real-time market *events*. Algorithmic trading is dominating financial markets and now accounts for over 70% of all trading in equities [4]. 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 [6]. For instance, a classical arbitrage strategy has an estimated annual profit of over $21 billion according to TABB Group [5]. Moreover, every 1-millisecond reduction in response-time is estimated to generate the staggering amount of over $100 million a year [6]; such requirements greatly increases the burden placed on scaling the event processing platforms.

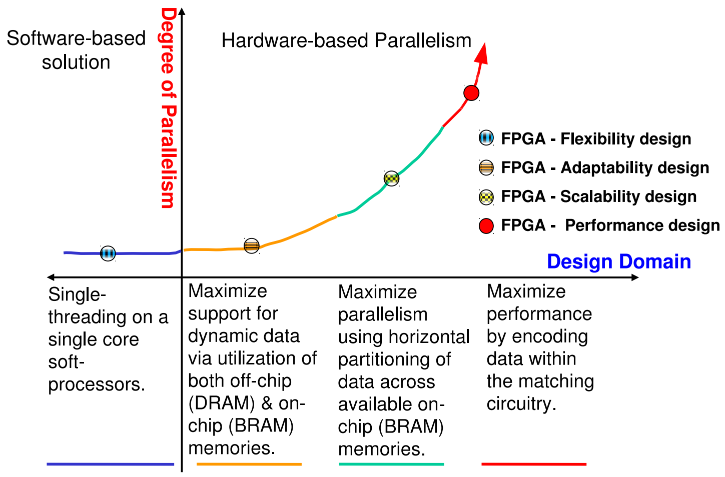

Software-based solution
Degree of Parallelism
Hardware-based Parallelism
FPGA - Flexibility design
FPGA - Adaptability design
FPGA - Scalability design
FPGA - Performance design
Design Domain
Single-threading on a single core soft-processors.
Maximize support for dynamic data via utilization of both off-chip (DRAM) & on-chip (BRAM) memories.
Maximize parallelism using horizontal partitioning of data across available on-chip (BRAM) memories.
Maximize performance by encoding data within the matching circuitry.

Figure 1: Various Degrees of Parallelism

To achieve throughput at this scale, we demonstrate our novel FPGA-based event processing designs (Field Programmable Gate Array) [9]. An FPGA is an integrated circuit designed to be reconfigurable to support custom-built applications in hardware. Potential application-level parallelism can be directly mapped to purpose-built processing units operating in parallel. Configuration is done through encoding the application in a programming language-style description language and synthesising a configuration uploaded on the FPGA chip.

Thus, in this demo, we demonstrate our solutions as a design trade-off between the degree of exploitable parallelism (cf. Fig. 1) versus the desired application-level requirements [9]. Requirements considered are: the ease of the development and deployment cycle (*flexibility*), the ability of updating a large subscription workload in real-time (*adaptability*), the power of obtaining a remarkable degree of parallelism through horizontal data partitioning (*scalability*), and, finally, the power of achieving the highest level of throughput by eliminating the use of memory and by specialized encoding of subscriptions on FGPA (*Performance*) [9].

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 commodity CPUs for data and event processing. 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 complex concurrency management techniques at the program and OS level. In contrast, FPGAs allow us to get around these limitations due to their intrinsic highly inter-connected architecture

[1]Project website: http://msrg.org/project/fpga-ToPSS.

DEBS'11, July 11–15, 2011, New York, New York, USA.
ACM 978-1-4503-0423-8/11/07.

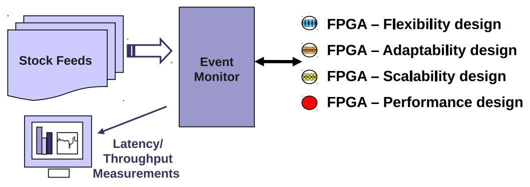

Stock Feeds
Event Monitor
FPGA – Flexibility design
FPGA – Adaptability design
FPGA – Scalability design
FPGA – Performance design
Latency/ Throughput Measurements

Figure 2: Demo Setup

and the ability to create custom logic on the fly to perform parallel tasks. In our design, we exploit parallelism, owing to the nature of the matching algorithm, by creating multiple matching units which work in parallel with multi-giga bit throughput rates, and we utilize reconfigurability by seamlessly adapting relevant components as subscriptions evolve.

2. DEMO METHODOLOGY

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

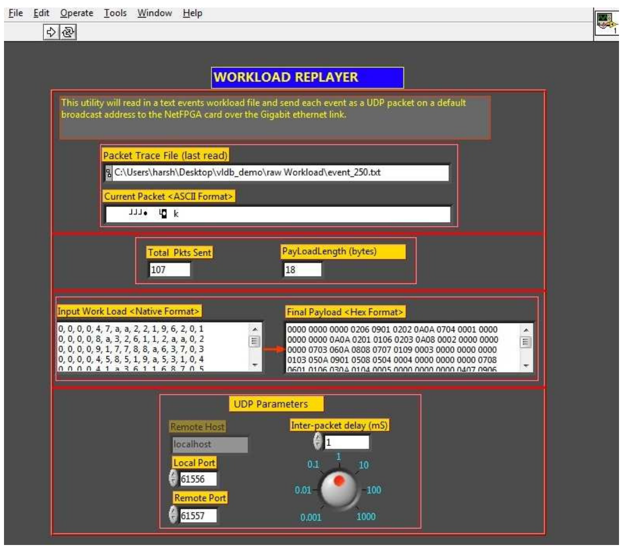
Demo Setup The input to our platform is a stream of events; events are encoded in the packets of the transport protocol that transmits event data to the FPGA board. In our implementation, we use UDP as transport over a directly connected 1 Gb/s Ethernet link (cf. Fig. 2). UDP simplifies the implementation, especially on our event processing platform side. Due to the direct, unshared Ethernet link, there are no severe transmission reliability issues to consider, except for the filling up of input/output buffers on the board. If no further events can be accepted by the board, packets are dropped and the maximal sustainable processing rate is achieved.

Our demonstration setup includes a laptop with a 1 Gb/s Ethernet interface (cf. Fig. 2). This machine drives the demo by generating the stock feeds, either based on our "event data stream generator," or from log-files of captured event traces. In practice, the online event data stream would be channeled to our platform in a similar manner so that the UDP data packets are transmitted to/from the Xilinx Virtex 5 LXT ML505 board, which is the employed FPGA board in our platform and for our demo. A USB-JTAG link is employed to program the FPGA board by a second laptop loaded with the Xilinx ISE10.1 EDK development tool suite for design synthesis and bit stream generation.

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`). The generated workload is transmitted to FPGA through our workload replayer (cf. Fig. 3(a)).

Demo Measurements We characterize the system throughput 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 when monitored for five seconds—a duration empirically found long enough to predict the absence of future packet drops at the given input rate. The latency of our solutions is the interval between the time an 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.

Demo Visualization As part of our demo, the latency information is continuously aggregated over a multitude of dimensions and presented visually in order to demonstrate the trade-offs and benefits of each solution (cf. Fig. 3(b)).


File Edit Operate Tools Window Help
WORKLOAD REPLAYER
Packet Trace File (last read)
Current Packet <ASCII Format>
Total Pkts Sent
PayLoadLength (bytes)
Input Work Load <Native Format>
Final Payload <Hex Format>
UDP Parameters
Remote Host
Inter-packet delay (mS)
Local Port
Remote Port
0.001
0.01
0.1
1
10
100
1000

This utility will read in a text events workload file and send each event as a UDP packet on a default broadcast address to the NetFPGA card over the Gigabit ethernet link.

Packet Trace File (last read): C:\Users\harsh\Desktop\vldb_demo\raw Workload\event_250.txt

Current Packet <ASCII Format>: JJJ• k

Total Pkts Sent: 107

PayLoadLength (bytes): 18

Input Work Load <Native Format>:
```
0, 0, 0, 0, 4, 7, a, a, 2, 2, 1, 9, 6, 2, 0, 1
0, 0, 0, 0, 8, a, 3, 2, 6, 1, 1, 2, a, a, 0, 2
0, 0, 0, 0, 9, 1, 7, 7, 8, 8, a, 6, 3, 7, 0, 3
0, 0, 0, 0, 4, 5, 8, 5, 1, 9, a, 5, 3, 1, 0, 4
0, 0, 0, 0, 4, 1, a, 3, 6, 1, 1, 6, 8, 7, 0, 5
```

Final Payload <Hex Format>:
```
0000 0000 0000 0206 0901 0202 0A0A 0704 0001 0000
0000 0000 0A0A 0201 0106 0203 0A08 0002 0000 0000
0000 0703 060A 0808 0707 0109 0003 0000 0000 0000
0103 050A 0901 0508 0504 0004 0000 0000 0000 0708
0601 0106 030A 0104 0005 0000 0000 0000 0407 0906
```

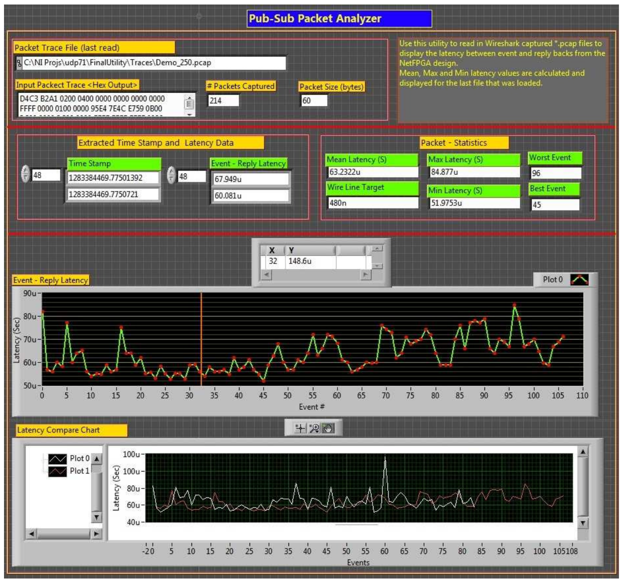
Remote Host: localhost

Inter-packet delay (mS): 1

Local Port: 61556

Remote Port: 61557

(a) Workload Replayer



Use this utility to read in Wireshark captured *.pcap files to display the latency between event and reply backs from the NetFPGA design.
Mean, Max and Min latency values are calculated and displayed for the last file that was loaded.

Packet Trace File (last read): C:\NI Projs\udp71\FinalUtility\Traces\Demo_250.pcap

Input Packect Trace <Hex Output>:
```
D4C3 B2A1 0200 0400 0000 0000 0000 0000
FFFF 0000 0100 0000 95E4 7E4C E759 0B00
```

# Packets Captured: 214

Packet Size (bytes): 60

	Time Stamp		Event - Reply Latency
48	1283384469.77501392	48	67.949u
	1283384469.7750721		60.081u

Mean Latency (S)	Max Latency (S)	Worst Event
63.2322u	84.877u	96
Wire Line Target	**Min Latency (S)**	**Best Event**
480n	51.9753u	45

X	Y
32	148.6u

(b) Event Packet Analyzer

Figure 3: *fpga-ToPSS* Interface.

3. 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] 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*.

[3] A. Farroukh, M. Sadoghi, and H.-A. Jacobsen. Towards vulnerability-based intrusion detection with event processing. In *DEBS'11*.

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

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

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

[7] M. Sadoghi and H.-A. Jacobsen. BE-Tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space. In *SIGMOD'11*.

[8] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H.-A. Jacobsen. Efficient event processing through reconfigurable hardware for algorithmic trading. In *VLDB '10*.

[9] M. Sadoghi, H. Singh, and H.-A. Jacobsen. Towards highly parallel event processing through reconfigurable hardware. In *DaMoN'11*.

[10] D. Srivastava, L. Golab, R. Greer, T. Johnson, J. Seidel, V. Shkapenyuk, O. Spatscheck, and J. Yates. Enabling real time data analysis. *PVLDB'10*.