Exploring Algorithmic Trading in Reconfigurable Hardware

Stephen Wray, Wayne Luk, Peter Pietzuch
Department of Computing, Imperial College London
London, United Kingdom
{*sjw06,wl,pp*}*@doc.ic.ac.uk*

Abstract—**This paper describes an algorithmic trading engine based on reconfigurable hardware, derived from a software implementation. Our approach exploits parallelism and reconfigurability of field-programmable gate array (FPGA) technology. FPGAs offer many benefits over software solutions, including a reduction in latency, while increasing overall throughput and computational density. All of which are important attributes to a successful algorithmic trading engine. Experiments show that the peak performance of our hardware architecture for algorithmic trading is 133 times faster than the corresponding software implementation. Six implementations can operate simultaneously on a Xilinx Vertex 5 xc5vlx30 FPGA on average, maximising performance and available resource usage.**

Keywords-**Algorithmic Trading; Reconfigurable Hardware;**

I. Introduction

Electronic trading enables trades to be completed in a virtual environment by computers with orders being communicated across computer networks. This revolution started in the 1970s and by 2007, 60% to 70% of the volume traded on the New York Stock Exchange (NYSE) was done so electronically [1]. In 1961, the average daily traded volume on the NYSE was four million, however by January 2001, daily volume was already topping two billion trades a day [2].

The increase in electronic trading had a large effect on how the equity markets reacted and systems designed to operate in these environments had to possess certain attributes to be successful:

- Orders now take a fraction of the time they previously took to reach the market, reduced from minutes to milliseconds, making system latency critical.
- Trading volume increased due to a decrease in cost, requiring systems to be able to handle large amounts of throughput.
- Market spread (price difference) shrank as a direct correlation to the increase in volumes and competitiveness, requiring investors to trade more frequently.

Algorithmic trading engines trade large quantity orders over a set amount of time. They can be described as complex event processors, consuming real-time market information, making decisions on this information and their internal state. The requirement for high throughput and low latency was realised in [3] for equity trading and CEP.

These attributes align with those that reconfigurable hardware provides making it an ideal solution, it has previously been discussed in some detail [4], but also explains why there hadn't been much market penetration. Some of those issue have now been solved and our approach maps a selection of algorithms into reconfigurable hardware to take advantage of these qualities. We have used a software solution to provide a comparison against our results. The contributions of this paper include the following:

- An introduction to algorithmic trading and the specifics examined in this paper; in Section II.
- An algorithmic trading architecture based on reconfigurable hardware that processes available information to decide whether an order should trade; in Section III.
- Implementation details and initial results from the hardware architecture proposed, compared against a comparative software implementation; in Section IV.
- Evaluation of the proposed approach and comparison with a related software solution, illustrating its potential; in Section V.

Currently, Credit Suisse has a joint venture with Celoxica to use reconfigurable hardware in their electronic trading systems [5], showing that these technologies are used in industry. Academic research on this subject has been limited however. One study [6] which is particularly relevant explores parsing market data feeds directly from a network interface in hardware. The study found that the hardware implementation could process 3.5 million updates per second, or roughly 12 times the current real-world maximum rate as well as improved latency.

II. Algorithmic Trading

Algorithmic trading is concerned with taking a large quantity order and breaking it up into multiple, much smaller orders, before sending them to market throughout the trading day [7]. This minimises market impact and achieves a consistent price which is less dependent on varying market conditions. Equity orders trade shares (stock) in a company which has a varying value dependent on the companies performance, the industry and global economics. The algorithm design determines when smaller orders are created and what quantity of shares they have depending on market conditions, internal parameters and state.

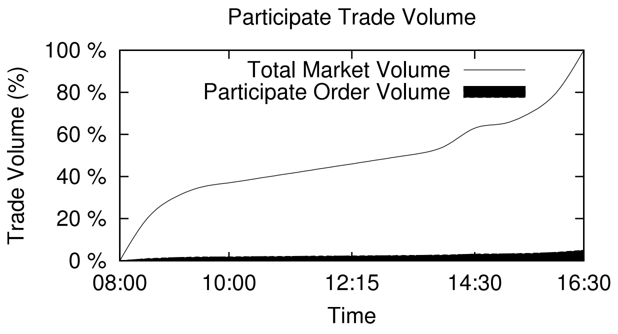

Figure 1. The Participate algorithm trading volume over a trading day compared to total market volume.

We looked at three trading algorithms, each with its own benchmark to beat. The benchmarks are documented and used in industry, measuring the performance of prices over time. Each captures a slightly different approach, we will only examine one algorithm in this paper, Participate as this is the most concise.

Participate [8], aims to trade a percentage against the total market activity. The average price of executions as shown in equation (1), is used as the price comparison benchmark. The total market volume decides the amount to trade.

$$\text{AP} = \frac{\sum_{t=0}^{n} P_t}{n} \tag{1}$$

Where P_t is the price executed at time t and n is the total number of prices seen. Figure 1 shows how a Participate traded order trades over a full trading day, aiming to be 5% of the traded market volume. It shows that the majority of trading happens at the start and end of the trading day.

Threshold Trading [8] is used to help increase the price performance [9] by giving the algorithm the ability to trade ahead or behind its predetermined target. This was based on an attractiveness function of the current market state which effects the size of a child order (a sub-order of the main order or parent order). A parameter limits this effect to prevent a large deviation from the original target.

System Overview, figure 2 shows a high-level overview of an algorithmic trading engine. The two inputs are orders and market data updates. An underlying order manager maintains the orders and handles communication between other systems. The algorithms take the orders and their state with market updates and calculate whether specific orders should trade [10]. Returning the parameters for the new child order to be created. In this paper we map the implementation of the algorithms into hardware.

III. Hardware Architecture

This section covers the system architecture of the algorithmic trading engine described in Section II. We will cover the most specific parts of the hardware architecture.

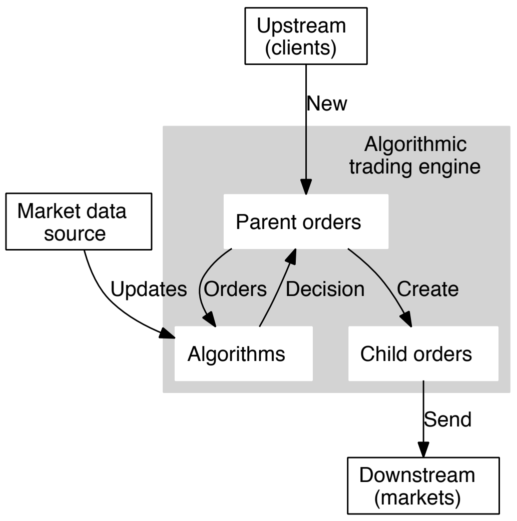

Figure 2. Process diagram for the algorithmic trading engine, depicting parent order input and child order output. Market data updates are supplied from a source which updates algorithm state.

Equity order and market updates are modelled in the system as a collection of parameters, such as the side of the order, the quantity and information that is specific to the algorithm trading them. Market updates have information about its symbol, the price and quantity for the entries.

Trading on Market Updates, orders can be traded on market data updates. If the price is favourable then it would be beneficial to trade the order ahead to achieve a better price. A price metric is used as a comparison, with the difference between the actual and theoretically traded volumes. The actual volume is how much the order has already traded and theoretical volume is what the order should have traded without threshold trading. If the market metric is positive then the market is favourable and the order should trade to improve its price. The amount to trade is normalised to prevent the order exiting trading boundaries and parameters governing the size of child orders before being sent.

Trading on Events, alternatively orders can be traded on events, Participate events are executions. The event updates algorithm state, which might allow orders to trade. The amount to trade is calculated from the order parameters, this is normalised and checked against boundaries and parameters, the only exception is when the remaining volume is less than any lower quantity allowed. This then allows the order to completed. If the market is unfavourable then the amount to trade can be reduced before being sent.

Market Favourability, is calculated against the price metric that the algorithm uses. The entries on the market are compared to the price metric and a weighted sum is taken over the entries after the comparison. If the price metric is greater than an entry, then that entry is favourable. If the entries start favourably then the market is favourable and unfavourable entries are discarded.

Parallel Processing, the hardware architecture supports three forms of parallelism, task-level, pipeline parallelism and expression evaluation parallelism. Placing multiple algorithms on a single device will result in task-level parallelism. Pipeline parallelism is used extensively when processing orders or entries in a market data update, as this information is independent. Expression evaluation parallelism is utilised to reduce the number of cycles required for calculations on data where results are independent, such as calculating the child order quantity. One trade-off a long parallel pipeline is increased clock speed and power consumption [11], however this would increase latency, which is undesirable and therefore the designers need to make a trade-off.

Reconfigurability, our hardware architecture makes use of arrays of registers for performance. This aids parallelism but the array sizes have to be determined at compile time. Our solution is to allow batch processing, the array size is set to the batch size which determined at compile time. Orders and entries are processed in batches until complete. A trade-off exists between the batch size and the number of algorithms placeable on a single device.

IV. Implementation and Results

The hardware implementation was developed in Handel-C [12], the hardware development environment used was Mentor Graphics DK Design Suite 5.2. This compiled the Handel-C into Electronic Design Interchange Format (EDIF), allowing Xilinx ISE Webpack 11.1 to place and route the design on the target chip, in this case the Xilinx Vertex 5 xc5vlx30.

The software implementation was developed in C++, the software development environment used was Eclipse with the CDT plug-in to aid C++ development. The compiler g++ 4.2.1 on Mac OS X 10.6.3 was used with full optimisation enabled. The host machine was a 2.33GHz Core 2 Duo with 3GB of 667MHz DDR2. All software tests where executed multiple times to produce and avoid any skew in the results from the operating system prioritising other processes on some test runs.

Hardware Results, using our implementation we can calculate the number of cycles required to complete tasks, this is possible because only a small set of statements in Handel-C consume a clock cycle. For the purpose of our performance analysis we set the batch sizes for both order and update entry processing to 10. The top half of table I shows the results for these calculations.

Table I
Number of cycles required for the three main activities and theoretical maximum performance on updates and orders processed per second.

Calculation	Participate Algo
Cycles for Update Handling	31
Cycles for Event Handling	41
Cycles for Market Metric Calculation	13
Max Updates per Second	25,900,000
Max Orders per Second	341,000,000

Table II
Resource usage for the algorithm when placed and routed on a Xilinx Vertex 5 xc5vlx30 FPGA.

Resource Usage	Participate Algo	xc5vlx30
Slice Logic Utilization	Used	Available
Number of Slice Registers	544	19,200
Number of Slice LUTs	1,732	19,200
Number of Occupied Slices	619	4,800
Number of Bonded IOBs	4	220
Number of BUFG/BUFGCTRLs	1	32
Number of DSP48Es	4	32
Number of Route-Thrus	88	-
Number of LUT Flip Flop Pairs	2,004	-
Number of Engines	7	-

The implementation was placed and routed by Xilinx's ISE which produces details on the resource usage for the specific device. Table II shows how many implementations of an algorithm the Xilinx Vertex 5 xc5vlx30 chip can support with its total 4,800 logic slices available, algorithms can be placed multiple times concurrently on a single chip. The xc5vlx30 has a realistic maximum is 6 algorithms, performance may be impacted with any more. The number is calculated by dividing the usage by the availability for each resource and then taking the minimum value produced. We will allow for 6 algorithms to be used simultaneously to calculate total maximum performance of the device.

Xilinx ISE also provides the Critical Path Delay (CPD) of the system, from which we can derive the theoretical maximum clock speed. In this case the CPD calculated is 1.765ns which equates to 566.6MHz. This is very high and it is unlikely that the device would be run at this speed, a more realistic clock speed would be 200MHz which we will assume as the device's configuration. With the clock speed we can calculate the estimated performance of the device with our previous information on the number of clock cycles required. The bottom half of table I shows the throughput of the implementation at this speed.

Software Results, the software implementation was tested on the development machine. The test market updates and orders where a simple selection of parameters which were repeatedly fed into the process for testing. The tests used the same batch size of 10 and the average execution time was 30ms executing on a single processor. With the prevalence of multi-core processors, running multiple instances of the system would result in near linear scaling of performance. For a dual-core processor, doubling performance and a quad-core processor, quadrupling performance.

The software results found that the maximum number of updates processed per second was 139,000 per process. On the dual-core test machines the result was double, at 279,000 updates per second.

Comparison. The hardware must run at a frequency which will provide greater performance than the software to give reason for using a hardware implementation. Using the software results and our information on the number of cycles the calculations require, when can calculate the intersection point at which the hardware will eclipse the software performance. The result is 2.5MHz which is low given the theoretical maximum clock speed. This means that there is plenty of headroom for additional functionality which would be required in a production system and possible to see significant performance gains.

V. Conclusion

In this paper we have proposed an implementation of algorithms for trading equity orders in reconfigurable hardware and compared it to its software implementation. We have evaluated the performance and found it to be superior, increasing maximum throughput. In addition we have looked at the size of the implementation and the number of algorithms that can concurrently be placed on a chip. This makes the implementation more cost effective, flexible and increases performance further.

The hardware is approximately 133 times faster at 200MHz than the equivalent software using 6 instances on a chip. More complex algorithms would be expected to show similar gains over their software counterparts. Floating and fixed point arithmetic needs to be investigated and its trade-off between accuracy and performance.

A limitation of the hardware is in supporting a large number of differing algorithms, possible solutions are run-time reconfiguration, use of larger capacity or additional chips.

Further Work. There is a large scope for further work in this area, a power consumption comparison between the implementations should also provide compelling reason for a system of this nature to be used in industry. Other possibilities include a high-level language to describe algorithms, validation of algorithms and moving more of the software into hardware.

Acknowledgment

We would like to thank Qiang Liu and Tim Todman at Imperial College London for their help with the Mentor Graphics DK Design Suite and Xilinx ISE. Additionally we would like to thank the three anonymous reviewers for their invaluable feedback that enabled this paper to be improved.

References

[1] R. Martin, "Wall Street's quest to process data at the speed of light," *Information Week*, Apr 2007.

[2] NYSE, "http://www.nyse.com/," *nyse.com*, 2010.

[3] M. Migliavacca *et al.*, "High-performance event processing with information security," Imperial College London, Tech. Rep., 2010.

[4] Automatedtrader.net, "FPGA's - Parallel perfection?" *Automated Trader Magazine Issue 02*, July 2006.

[5] I. Schmerken, "Credit Suisse hires Celoxica for low-latency trading and market data," *FinanceTech*, Nov 2009.

[6] G. Morris, D. Thomas, and W. Luk, "FPGA accelerated low-latency market data feed processing," in *High Performance Interconnects, 2009*, Aug. 2009, pp. 83–89.

[7] Wikipedia, "http://en.wikipedia.org/wiki/algorithmic_trading," *wikipedia.org*, 2010.

[8] Automatedtrader.net, "Scalable participation algorithm," *Automated Trader Magazine Issue 10*, Q3 2008.

[9] T. C. Group, "Algorithmic trading trends and drivers," *Tellefsen.com*, Jan 2005.

[10] Hot Bridge, *Product Overveiw*, Hotbridge.co.uk.

[11] S. Bard and N. Rafla, "Reducing power consumption in FPGAs by pipelining," in *Circuits and Systems*, 2008.

[12] *Handel-C Language Reference Manual*, Mentor Graphics.