Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1880 – 1889

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

An FPGA Implementation of an Investment Strategy Processor

Christoph Starke, Vasco Grossmann, Lars Wienbrandt, Manfred Schimmler

Department of Computer Science
Christian-Albrechts-University of Kiel
24098 Kiel, Germany
{*chst, vgr, lwi, masch*}*@informatik.uni-kiel.de*

Abstract

A special purpose processing element is described which can be used to optimize an investment strategy for financial securities. It has been used to configure the FPGAs of the massively parallel hardware platform RIVYERA. Using this configuration, the compute intensive part of the technical analysis of financial markets can be accelerated with a speedup of more than 17,000 compared to a high-performance PC.

Keywords: RIVYERA, FPGA, high-performance computing, parallel processing, technical investment strategy

1. Introduction

Modern technical financial market analysis is no longer restricted to the observation and prediction of charts [1, 2]. Instead, new methods are coming up that involve computationally intensive tasks as e.g. data mining [3, 4, 5, 6]. The quality of these new approaches can be measured in terms of an outperformance in comparison to a simple buy-and-hold strategy. Due to the extreme high number of mathematical calculations that have to be performed in a limited time interval, these methods are mostly implemented on supercomputers or on special purpose hardware [7, 8].

In this paper we present an approach of the latter category. Its idea is to perform an exhaustive search for an optimal set of indicator weights in the space of all possible weights. The required computation steps are described in Section 2. Since, on the one hand, the search space grows exponentially with the number of indicators, the method is highly demanding with respect to computing power. However, on the other hand, the same calculations have to be performed with similar data again and again such that the problem can trivially be performed in parallel. Due to this fact, the massively parallel FPGA architecture RIVYERA is a promising hardware platform for this particular problem. Section 3 describes the RIVYERA with its communication structure. The new processing element which has been multiply implemented on the FPGAs of RIVYERA is explained in Section 4. Section 5 evaluates the speedup gained by this implementation in comparison to a sequential implementation on a high-performance PC. These values are measured and based on existing systems. A conclusion is given in Section 6.

1.1. Related work

FPGA-based systems are discussed whenever there is a need to speed up a compute intensive task consisting of many independent identical operations. Option pricing is an example for such an application in the area of financial marked prediction. There are several approaches to exploit the degree of parallelism provided by an FPGA based on finite difference or Monte-Carlo methods [7, 9, 10]. Another approach is using data mining methods for market

doi:10.1016/j.procs.2012.04.206

prediction. Kannan et al. [11] combine classical fundamental and technical indicators in order to estimate the performance of a security during the following trading day. Langdell [12] implements data mining methods together with neural networks and decision trees for predicting currency exchange rates. Neural networks have also been used for FPGA-based stock market forecasting in [13]. However, as far to our knowledge, there are no publications available about the performance of these theoretical approaches in real financial markets.

2. Description of the Investment Strategy for Securities

The main objective is the identification of a strategy that successfully generates profitable market orders for a single security P by analyzing a set of known indicators. This strategy is chosen by the score measured by profit within a given historical time interval. Common indicators are e.g. S&P500, EuroStoxx50, DAX, Nikkei225, Hang Seng, EUR/JPY, EUR/USD.

The set of considered indicators depends on P. It is assumed that the short-term influence between every indicator and P is approximately linear with a variable strength. These influences, called weights in the following, are initially unknown and have to be determined. Charges and taxes are not considered due to reasons of space. It will not be discussed whether these assumptions are optimal or even reasonable. However, several tests for different securities and time peroids reveal significant increases in comparison to a common buy-and-hold strategy.

Starting with a fundamental issue concerning technical analysis, we assume that patterns of past price fluctuations will appear in a similar manner in the future [1]. As a consequence, meaningful weights can be determined by the analysis of historical data.

This set of historical data is known and given by the $(m+1)\times n$ matrix V that consists of prices for n indicators I_0 to I_{n-1} and for $m+1$ sequent trading days d_0 to d_m (weekends and worldwide nonbusiness days disregarded), hereinafter referred to as calibration period. Thus, $V_{i,j}$ is the value of indicator I_j at day d_i. Tests with different securities point that an assurance of robust values requires $m \geq 125$ and $n \geq 7$. The $m \times n$ matrix of price fluctuations R can be extracted from V. Provided that $V_{i,j} \neq 0$ for all i, j, it holds $R_{i,j} := \frac{V_{i,j}-V_{i-1,j}}{V_{i-1,j}}$ for $(1 \leq i \leq m)$ and $(0 \leq j \leq n-1)$. Let R_i be the vector $(R_{i,0}, R_{i,1}, \ldots, R_{i,n-1})$. $R_{i,j}$ is the ratio that describes the percentaged difference of indicator I_j from d_{i-1} to d_i. The vector R_i is the i-th row of the matrix R. Furthermore, it is assumed that for all i with $0 \leq i \leq m$, the historical prices P_i of P at d_i are known and $P_i > 0$ holds. Exactly one value per day is considered in the following.

The weights should ideally represent the effective influence of the corresponding indicator to the security P. To determine the weight w_j for every indicator I_j, different weight combinations are tested. Such a combination of weights is a weight vector $w = (w_0, w_1, \ldots, w_{n-1})$.

For every possible weight vector, a score Z_m is calculated in the following way: Initial point is the day d_0 with a cash value C_0 and a depot value D_0. Generally, let C_i be the cash money, D_i the depot value concerning the pieces of P in the depot, and $Z_i := C_i + D_i$ the total property at day d_i $(0 \leq i \leq m)$. Let $f(R_i)$ be a function that returns a numerical instruction for the next trade. A positive value of f indicates a buy indication, a negative one a sell indication. Accepting an insignificant inaccuracy, the algorithm can be simplified: The unbounded order size O_i^* is given by $O_i^* = Z_i \cdot f(R_i)$. A buy at day d_i is limited to C_i in order not to overdraw the cash account and a sell is limited to D_i, accordingly. By ensuring that $-D_i \leq O_i^* \leq C_i$, O_i^* becomes the valid order size O_i. In case of offense, O_i^* has to be bounded. The next step is the calculation of the cash and depot values for the next day. While it holds $C_{i+1} := C_i - O_i$, the new depot value is computed with $D_{i+1} := (D_i + O_i) \cdot \frac{P_{i+1}}{P_i}$. These sequences are calculated up to the last day m, where the final score Z_m is evaluated. Finally, let w^* be the weight vector that yields the maximum Z_m for the considered time period. The investment strategy is based on the evaluation of $f(R_x)$ using w^* on every new day d_x. This value can directly be translated into a buy or sell indication.

The extremly high number of possible combinations results in the inoperability of a direct brute force approach, even on a supercomputer. Assuming only 8 indicators and 100 values for each of those, there are 100^8 possible candidates for w^*. Considering a randomly chosen calibration period of half a year (26 weeks with 5 trading days per week), the function f is called $26 \cdot 5 \cdot 100^8 = 1.3 \cdot 10^{18}$ times. Even the RIVYERA implementation presented in this paper would require 377 days to evaluate such a number of combinations.

Instead of this direct approach, w^* is calculated iteratively. Only 8 values for every of the 8 indicators are used in a single iteration. Thus, these values can be displayed in the 8×8 matrix W_k where k is the index of the current iteration. In every iteration k, an exhaustive search over all weight vectors that can be derived from W_k is executed.

Figure 1: RIVYERA S3-5000. The housing was opened to reveal the 16 FPGA cards.

Considering the same calibration period, $26 \cdot 5 \cdot 8^8 \approx 2.2 \cdot 10^9$ function calls have to be done per iteration. Actually, the problem size is reduced further by equally dividing the search space into b disjunct subspaces. The value b depends on the implementation and specifies the number of jobs that are done in parallel. In every iteration k, the best weight combination $w^*_{(k,q)}$ is calculated for the qth subspace ($0 \leq q < p$). While the first iteration is based on a predefined matrix W_1, the computation of W_k ($k > 1$) is based on the results of the previous iteration. So, the b optimal weight vectors $w^*_{(k-1,q)}$ are utilized to create an improved weight matrix W_k. An interesting fact is that the quality of W_k depends on the degree of parallelism as an increasing b causes more input values for the calculation of W_k.

There are many reasonable heuristic approaches to manage this computation, e.g. calculate the average of the optimal weights and create new weights in its environment. However, neither are these approaches easily parallelizable nor is the computational effort significant in comparison to the exhaustive searches. Thus, these computations are not specified in the following. All in all, every iteration consists of two steps: First, the b best weight combinations of iteration $k-1$ are used to calculate W_k. Second, the search space given by W_k is seperated in b subspaces and an exhaustive search is performed on all of these. The best weight vector evaluated by the last iteration is w^*.

The compute-intensive part of the application is the execution of exhaustive searches. Hence, the following sections will focus on the FPGA-based acceleration of these computations.

3. FPGA-based Hardware Platform RIVYERA

The FPGA-based hardware platform RIVYERA, developed and distributed by SciEngines GmbH [14], has found applications in several areas of cryptanalysis (e.g. [15]) and bioinformatics ([16, 17, 18]). As described in this article, it is also suitable for high-performance stock market analysis. The specific RIVYERA S3-5000, as depicted in Figure 1, is used for the presented application.

RIVYERA consists of two basic elements. On the one hand, a standard server grade mainboard equipped with an Intel Core i7-930 processor, 12 GB of RAM and 2TB of hard disk space, provides the resources for quick pre- and postprocessing purposes (Figure 2, left side). The installed standard Linux operating system is completely independent. It is referred to as host system in the following. On the other hand, the FPGA-based super computer provides the resources for parallel high-performance applications (Figure 2, right side). It consists of a backplane and up to 16 FPGA cards, each equipped with eight user configurable Xilinx Spartan3-5000 FPGAs. Since the test system referred to in this article is fully equipped, 128 FPGAs can be utilized in total. Furthermore, a DRAM module with a capacity of 32*MB* is directly attached to each FPGA.

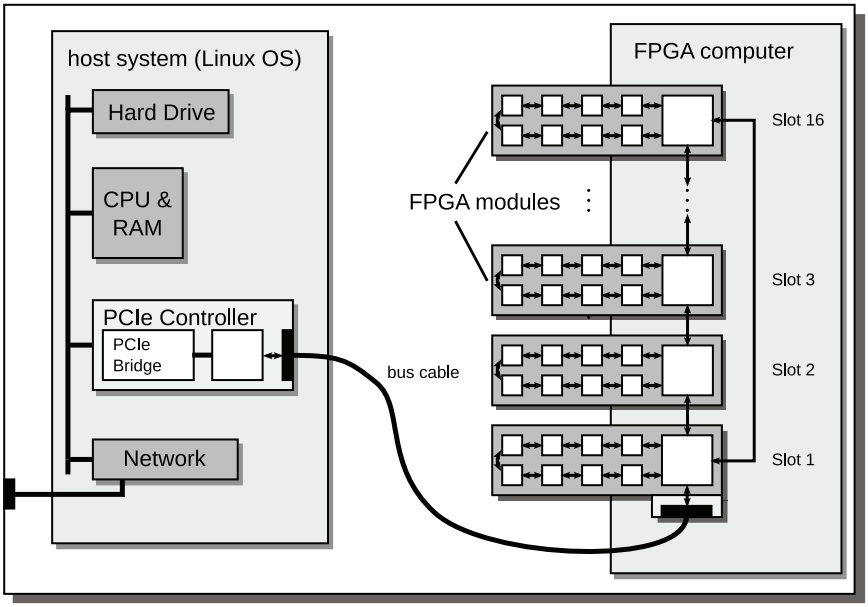

Figure 2: RIVYERA S3-5000 hardware structure.

A systolic-like bus system connects all FPGAs. The communication on each FPGA card is provided by a ring architecture. All inherent FPGAs including the communication controller are connected with their two direct neighbors. Likewise, the interconnection between two communication controllers is provided by a ring that is formed by the connection of all neighboring FPGA card slots on the backplane. The communication is physically realized by high-throughput symmetric LVDS point-to-point connections.

The host-mainboard and the FPGA-based computer communicate via a PCIe controller card that is directly connected to the first communication controller of the FPGA cards. The requirement of a higher bandwidth from the host system to the FPGA-based computer may be fulfilled by attaching more PCIe controllers to other FPGA cards. The measured net bandwidth for a configuration as it is used for this application reaches up to $66MB/s$. Of course, the latency differs depending on the length of the communication chain, according to which clients communicate with each other.

The applicaction development is assisted by an API for each of the two basic elements. An API concerning the data transfer between host and FPGA system (including broadcast facilities) is provided as well as an API for the user defined hardware configuration of the FPGAs including the usage of the attached DRAM. This implies functionality for the data transfer to other FPGAs and the host system as well.

4. Implementation

4.1. Implementation Overview

Likewise, the application itself consists of two basic elements. As already mentioned, a single iteration step k is separated into the computation of W_k, the division into b subspaces and the performance of b exhaustive searches. While W_k is calculated and segmented on the host system, the compute-intensive exhaustive searches are executed on the FPGA modules. Since these calculations can be evaluated independently, the algorithm is suitable for massive parallelization. In the following, let g be the total number of FPGAs while the number of search units synthesized per single FPGA is denoted p. As a subspace is assigned to every search unit on every FPGA, it holds $b = g \cdot p$. An increasing b directly causes smaller subspaces. Thus, the computational speed rises approximately linear with the number of FPGAs and the number of search units per FPGA.

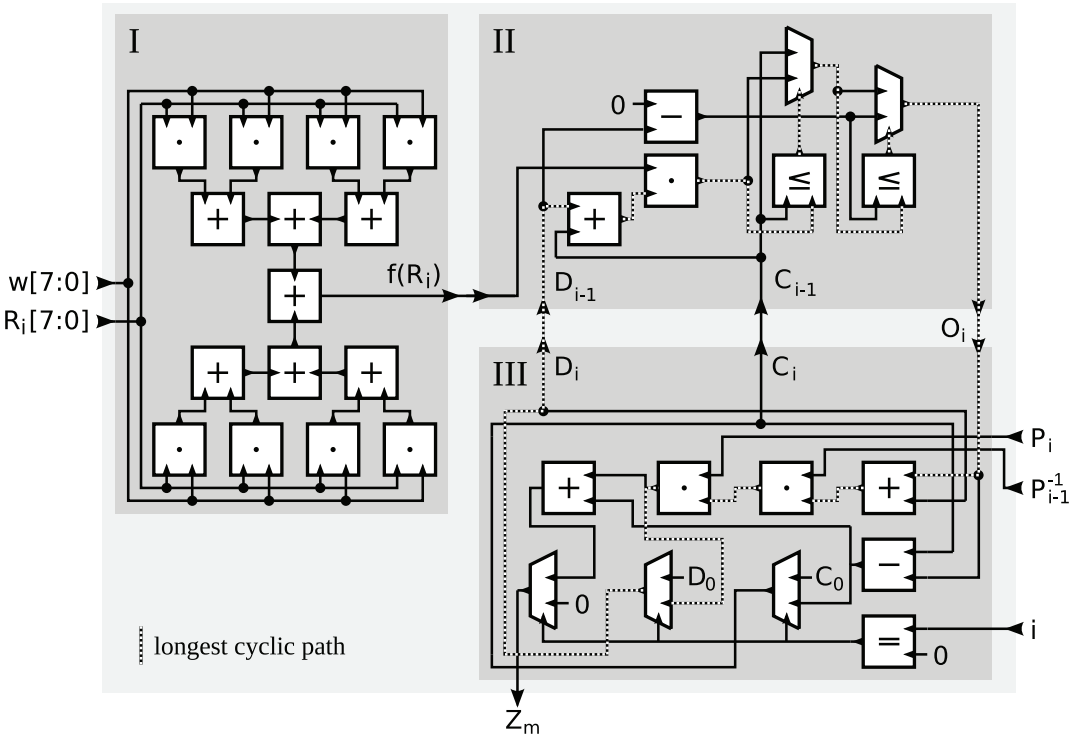

Figure 3: Scoring pipeline for 8 indicators.

4.2. Processor Architecture

The following description focuses on the FPGA-based part of the application. As the execution of the exhaustive searches is data-parallel and only differs in the considered subspace, the implementation is completely scalable. Thus, it is reasonable to restrict to contemplations of the utilization of a single FPGA. Regarding the underlying RIVYERA platform, the implementation presented here is optimized for Xilinx Spartan3-5000 FPGAs.

4.2.1. Implementation of the Scoring Function

Only a few computation steps are required for the presented algorithm. However, these steps have to be executed m times for the calculation of a single score Z_m. Considering a common search space of some billions of possible combinations, trillions of executions would be necessary. Hence, the major part of the computational effort is based on these evaluations and therefore, it is obligatory to maximize the throughput of the intended scoring unit. A pipeline architecture is the most capable approach concerning this objective as it enables the calculation of one pair (C_{i+1}, D_{i+1}) per clock cycle.

As the values C_{i+1} and D_{i+1} are defined recursively, the input values for the next iteration are not known until the pipeline finishes its execution. Therefore, the pipeline is only working to full capacity by the concurrent evaluation of l different weight vectors. l is the length of the longest cyclic path and given by the number of clock cycles that are necessary to compute C_{i+1} and D_{i+1} from C_i and D_i (see Figure 3).

Basically, the structure can be subdivided into three segments that are shown in Figure 3. The first stage is given by the following equation:

$$\text{I:} \quad f(R_i) = \sum_{j=0}^{n-1} w_j \cdot R_{i,j}$$

Assuming n indicators, the evalation of $f(R_i)$ requires n multiplications and $n-1$ additions. After the evaluation of $f(R_i)$, the order size O_i at day i is computed:

$$\text{II:} \quad O_i := \begin{cases} -D_i, & \text{if } O_i^* \leq -D_i \\ C_i, & \text{if } C_i \leq O_i^* \\ O_i^*, & \text{else.} \end{cases}$$

The intermediate result $O_i^* = (D_i + C_i) \cdot f(R_i)$ is restricted to O_i by the usage of two multiplexers and corresponding comparators.

Finally, new cash and depot values are calculated:

$$\text{III:} \quad (C_{i+1}, D_{i+1}) := \begin{cases} (C_0, D_0) & \text{if } i = 0 \\ (C_i - O_i, (D_i + O_i) \cdot \frac{P_{i+1}}{P_i}) & \text{else.} \end{cases}$$

The value i rises until the end of the calibration period ($i = m$). In this case, i is set to 0 which causes the start of the evaluation of a new weight vector. The sequences C_{i+1} and D_{i+1} are reset to the default values C_0 and D_0. In the same clock cycle the sum of C_m and D_m is calculated and transmitted to the multiplexer that refers to Z_m.

The resource consumption of a division unit is considerably higher than the consumption of a comparable multiplication unit [19]. Thus, the quotient $\frac{P_{i+1}}{P_i}$ is replaced by the multiplication $P_{i+1} \cdot P_i^{-1}$. Although the inverse elements have to be stored, this feature does not cause any appreciable disadvantages. In contrast to the usage of limited logic elements, the usage of additive memory is uncritical (see in Section 4.2.4).

The synchronicity of the pipeline is assured by the utilization of shifting registers. For example, stage III (see Figure 3) receives the input values O_i and P_{i+1}. While P_{i+1} is given, O_i depends on the calculations of stage I and II. The transfer of P_i is delayed using shifting registers. Optimized in regard to spatial cost, the longest cyclic path includes $l = 57$ clock cycles.

Assuming n indicators, the scoring pipeline consists of $n + 3$ multiplications, $n + 2$ additions, 2 subtractions, 3 comparators and 5 multiplexers (see Figure 3). These are $15 + 2n$ operations in total.

Since the search space can perfectly be divided into sub spaces, the computation time is inversely proportional to the number of scoring pipelines. Hence, the fundamental objective is to maximize the number of scoring pipelines per FPGA.

4.2.2. Reducing the Size of a Scoring Pipeline

A Spartan 3-5000 FPGA provides 8,320 CLBs (Configurable Logic Blocks), each consisting of four slices. That are 33,280 slices in total. Additionally, 108 dedicated 18×18 bit multipliers can be assigned for synthesis [19]. A single precision floating point representation of all variables is assumed. In case of 8 indicators and the usage of 32 multipliers, 8,584 slices (i.e. 25%) are required. Considering 16 indicators and 44 multipliers, the amount of slices rises to 14,064 (i.e. 42%). This increasement can exclusively be reduced to the requirement of 8 additional adders and multiplicators in first stage of the pipeline. The large difference between the resource utilizations of these two applications is caused by the comparatively high spatial cost of floating point units [19]. We reserve 10% of the slices for control units to trigger the pipelines. Hence, three pipelines fit on a single FPGA in the case of 8 indicators and two pipelines in the case of 16 indicators.

To react on optimization purposes, all variables are analyzed concerning co-domain and required precision. Especially the calculation of $f(R_i)$ is very area expensive. The involved values are the matrix R and the weight vector w. Since the daily price fluctuations rarely exceed the interval $[-10\%, 10\%]$, the values of R are stored in an 18-bit fixpoint representation. The decimal place is coded in 12 bits, such that the new co-domain is the interval $[-32\%, 32\%)$ with a precision of $2^{-12}\%$. This seems to be the best trade-off between precision and overflow immunity. Likewise, the weight vector is stored in an 18-bit fixpoint representation. The position of the decimal point is not predetermined and can be explicitly specified for every application. This huge advantage is caused by the fact that every multiplication in stage I can directly be assigned to one of the dedicated 18×18 multipliers. A fixpoint representation is utilized for all other variables as well. Cash, depot value and stock prices are stored in 32 bits and can easily be converted to integer values by choosing the unit cent. Since it can be assumed that stocks are more expensive than 1 cent, the inverse stock prices are always smaller than 1 cent^{-1}. Therefore, these values are represented as the first 32 bits following the binary point.

Table 1: Resource allocation for one pipeline with different number representations.

Indicators	Floating Point		Fixpoint	
	Slices	Multipliers	Slices	Multipliers
8	8,584 (25%)	32(30%)	4,801 (14%)	12(11%)
16	14,064 (42%)	44(42%)	5,395 (16%)	20(19%)

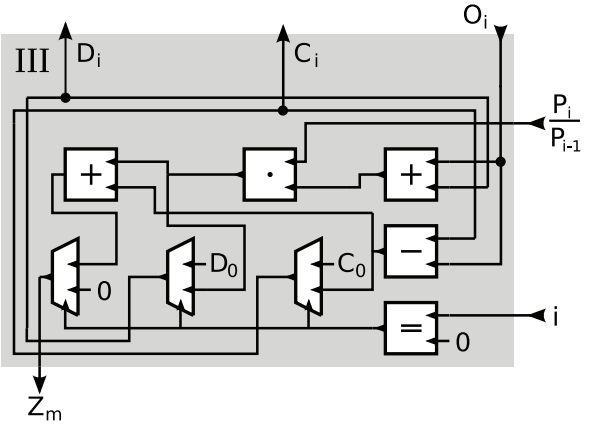

Figure 4: Error minimization by outsourcing the computation of $\frac{P_{i+1}}{P_i}$.

The resource utilization has been reduced further by minimizing the length of the pipeline and thus, the number of shifting registers. According to faster fixpoint units, the longest cyclic path reduces to 37 clock cycles.

With these optimizations, the total number of pipelines per FPGA has been increased to 6.

4.2.3. Examination and Reduction of Rounding Errors

The fix point representation reduces the necessary amount of logical elements but entails numerical inaccuracy. In general, impreciseness directly leads to an inexact selection of weight vectors and thus, should be as small as possible. Assuming $m = 125$, the presented approach yields to measured average rounding errors of Z_m of about 0.2% to 0.5%. The following regards the minimization of this value starting with theoretical considerations regarding the error size.

Let index F indicate the usage of the floating point representation while index O denotes the optimized fix point representation. The residual $e_f = f(R_i)_F - f(R_i)_O$ specifies the error relating to f and directly depends on the precision of R. Besides, this scales linearly with the number of indicators and the absolute values of the assigned weights. The range of the residual e_f is given by the interval $[-2^{-13} \cdot \|w\|_1 \leq e_f < 2^{-13} \cdot \|w\|_1]$. Assuming 8 indicators and an assignment of the maximum possible weight 32 to each indicator, it denotes a maximum residual of $e_{f_{max}} = 8 \cdot 32 \cdot 2^{-13} = 3.125\%$, i.e. up to 3.125% of the total property may be traded accidentially every day. However, this value is only of theoretical concern. The effective residuals are considerably smaller since the error size of every indicator is uniformly distributed with a standard deviation of 2^{-14}. This causes a standard deviation $\sigma(e_f) \ll 2^{-14}$ and an expected value $E(e_f) = 0$ for 8 indicators. As all errors may be positive or negative, most inaccuracies compensate each other due to their accumulation. Furthermore, $f(R_i)$ is only considered if it holds $-D_i < (D_i + C_i) \cdot f(R_i) < C_i$. As a consequence, these errors can be regarded as statistically insignificant.

The analysis of the error of the quotient $\frac{P_{i+1}}{P_i}$ shows a contrary result. For the expected value of the residual $e_P = \frac{P_{F,i+1}}{P_{F,i}} - \frac{P_{O,i+1}}{P_{O,i}}$ it likewise holds $E(e_P) = 0$ since it is uniformly distributed. However, the higher standard deviation $\sigma(e_P) = 2^{-14}$ causes a major probability for perceptible errors. Additionally, there is no dependency regarding the utilizaion of this value. Assuming m days, m multiplications with probably inaccurate quotients are definitely executed. On that account, the first implementation is improved by outsourcing the division to the host system (see Figure 4). This does not cause any further computational effort since these values have to be calculated only once. Thus, it is reasonable to execute these computations with floating point precision to exclude rounding errors specified by $\frac{P_{i+1}}{P_i}$.

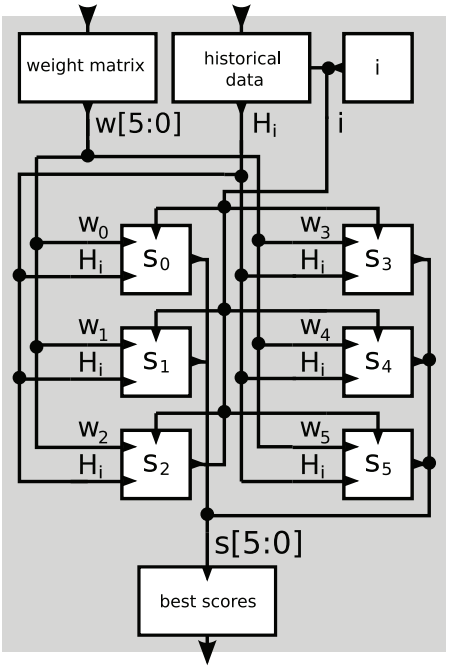

Figure 5: Outline of the processor architecture.

Another fundamental benefit is given by the removal of the constraint of integer cent values for P_{i+1} and P_i. Instead, the precision can be arbitrarily adjusted to the precision of the given historical data set. Assuming $32bit$ values and that the quotient $\frac{P_{i+1}}{P_i}$ will not exceed 400%, the fix point precision is set to 2^{-30}. By this means, the measured average inaccuracy of Z_m is reduced to the domain between 0.002% and 0.01%.

4.2.4. Triggering the Scoring Pipelines

Four input values are transfered to every scoring pipeline: the vector R_i and the vector w are required to evaluate $f(R_i)$. $\frac{P_{i+1}}{P_i}$ and i are necessary during the third stage. Since all values except w only depend on i, it is reasonable to set i globally for all scoring units. As a consequence, all pipelines can directly be triggered with the same values. Thus, the historical data is suitable to be stored in Block RAM words since one memory access per clock cycle is required. The data set for a day i is the word $H_i = (\frac{P_{i+1}}{P_i}, R_i)$ that has a size of $32 + 18 \cdot n$ bits, e.g. 176 for $n = 8$ indicators. Since a Spartan3-5000 provides 104 RAM blocks with $1,872kbit$ in total, historical data for more than 9,000 days can be stored. In fact, the RAM address i has to be changed every l clock cycles only since l independent score evaluations are executed in parallel.

In constrast, a unique weight vector for every pipeline has to be created and distributed in every clock cycle. The search space is determined by the 8×8 matrix W that is equal for every FPGA. It consits of $|W| = 8^8 \approx 16.7 \cdot 10^6$ combinations. To avoid multiple evaluations of elements of the search space, it is equally divided in $g \cdot p$ subspaces. Thus, a unique set of combinations is assigned to every pipeline. Every element is identified by an $24bit$ identifier in the range of $[0, |W|)$. These are 8 sequences, each consisting of 3 bits. Every sequence represents an indicator, 3 bits are necessary to identify the 8 given candidates. Pipeline q $(0 \leq q < p)$ receives the interval $[q \cdot \frac{|W|}{g \cdot p}, (q+1) \cdot \frac{|W|}{g \cdot p})$.

A unique weight vector can be composed by extracting the $3bit$ sequences of every indicator. A reasonable way is the creation of bit masks that can be reduced to a wiring problem in case of hardware. In doing so, the matrix elements are identified efficiently. Nevertheless, $p \cdot n$ different values have to be loaded in one clock cycle where n is the number of indicators. The only way to achieve such a computational flexibility is the direct storage in logical elements.

Since i is triggered synchronously for all pipelines, the scores Z_m are evaluated at the same time. Obviously, it is not reasonable to store all 8^8 values. Likewise, a list of the best scores creates a high computational effort as it implies

Table 2: Runtime comparison of an Intel Core i7-970 PC and RIVYERA S3-5000. The calibration phase is set to 26 weeks for all measurements.

Runtime		
No. of weight vectors	Core i7-970	RIVYERA S3-5000
$1 \cdot 10^9$	44h 56m	9.15s
$50 \cdot 10^9$	93d 14h	7m 37s
$1 \cdot 10^{12}$	5.13 years	2h 32m

Table 3: Comparison of the energy requirements of an Intel Core i7-970 PC and RIVYERA S3-5000 for the same test sets as in Table 2. Power consumption is $250W$ for the Core i7-970 and $600W$ for RIVYERA S3-5000. The energy costs are calculated using $0.20€/kWh$.

Power consumption				
No. of weight vectors	Core i7-970		RIVYERA S3-5000	
$1 \cdot 10^9$	$11.23kWh$	2.25€	$1.53Wh$	0.0003€
$50 \cdot 10^9$	$562kWh$	112.33€	$76.25Wh$	0.015€
$1 \cdot 10^{12}$	$11.23MWh$	2246.64€	$1.53kWh$	0.31€

that 6 results have to be sorted into the list in a single clock cycle. The best trade-off is to store only the best result of each subspace. Utilizing $p = 6$ pipelines and $g = 128$ FPGAs, $b = 768$ maxima are computed in one iteration step. This set seems to be comprehensive and suited as a basement for the evaluation of the new matrix W. The resulting composition of the processor components is shown in Figure 5.

5. Performance Analysis

We compared our implementation on RIVYERA (including the improved fixpoint representation) to a multi-threaded software implementation on a high-performance PC system, equipped with an Intel Core i7-970, 6 cores (12 threads with multi-threading), each running at $3.2GHz$, an ASRock X58 Extreme mainboard and $8GB$ GeIL DIMM DDR3-1066 RAM. The software has been implemented in C++, compiled with GNU g++ v4.1.2 using the maximum optimization flag `-O3`. All threads of this system were fully utilized for the comparison. The performance results regarding runtime and energy consumption are shown in Tables 2 and 3.

The clock rate of the FPGA implementation is $50MHz$. In case of 8 indicators, 6 pipelines are synthesized per FPGA. Considering that a pair (C_{i+1}, D_{i+1}) is calculated in every clock cycle, $128 \cdot 6 \cdot 50,000,000 = 38.4 \cdot 10^9$ pairs per second are evaluated. The specified PC version only reaches $2.26 \cdot 10^6$ values per second. The measured speedup based on a comparison of two existing implementations is $17,676.93$ and corresponds to these considerations. The power consumption of RIVYERA is up to $600W$, while the high-performance PC consumes about $250W$ for this application (measured with a customary power measurement device). Regarding the solving of one particular problem on our test system compared to RIVYERA, the power consumption is reduced by up to 99.98%.

Certainly, the comparison of an FPGA-based super computer with a PC system is not uncontroversial and implies the requirement of further explanations. The reasons for the unexpected low performance of the PC version are discussed in the following.

Intel declares 76.8 GFLOPs for an i7-970 [20]. The FPGA-based calculation of one pair requires $15 + 2n = 31$ operations for $n = 8$ indicators. Considering the same number of operations, it could be deduced that the referred processor reaches up to $\frac{78.6 \cdot 10^9}{31} \approx 2.54 \cdot 10^9$ pairs per second. Obviously, there is a large gap between this conclusion and the performance of the actual implementation. In fact, the computing power is not a problem. However, the bottleneck is formed by the intense memory utilization. The calculation of a single pair (C_{i+1}, D_{i+1}) requires access to $\frac{P_{i+1}}{P_i}$ and every element of R_i and w. Thus, even with the consideration of cache optimizations, multiple RAM and cache accesses are required. In constrast to the implementation on a PC, this could perfectly be integrated into a pipeline structure on an FPGA as the memory access is not random but constantly arranged. Summarized, the memory communication leads to a remarkable decelaration of the possible CPU throughput. This problem is totally avoided by the usage of FPGAs.

6. Conclusion

The presented application aims at the optimization of investment strategies and is perfectly suited for massively parallel computing. Thus, the implementation on the FPGA-based platform RIVYERA yields to a speedup of more than 17,000 and energy savings of more than 99% in comparison to a single high-performance PC. For future work, a portation of this application to the already available Spartan6-based RIVYERA S6-LX150 is considered. Accordingly, we expect an additional speedup by the factor of four.

References

[1] Murphy, John J., Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications, New York Institute of Finance, New York, NY, 1999.
[2] Schwager, Jack D., Technical Analysis, Schwager on futures, John Wiley & Sons, New York, Chichester, Brisbane, Toronto, Singapore, c1996.
[3] M. Gavrilov, D. Anguelov, P. Indyk, R. Motwani, Mining the Stock Market: Which Measure is best?
[4] K. S. Kannan, P. S. Sekar, M. M. Satik, P. Arumugam, Financial Stock Market Forecast using Data Mining Techniques, in: Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol. 1, 2010.
[5] S. Langdell, Examples of the use of data mining in financial applications, in: Financial Engineering News, 2002.
[6] T. Rathburn, Data Mining the Financial Markets, Part 1.
[7] A. H. Tse, D. B. Thomas, K. H. Tsoi, W. Luk, Efficient reconfigurable design for pricing asian options, SIGARCH Comput. Archit. News 38 (2011) 14–20.
[8] Q. Jin, W. Luk, D. Thomas, On comparing financial option price solvers on fpga, in: Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on, 2011, pp. 89 –92.
[9] Q. Jin, W. Luk, D. B. Thomas, On comparing financial option price solvers on fpga, Field-Programmable Custom Computing Machines, Annual IEEE Symposium on 0 (2011) 89–92. doi:http://doi.ieeecomputersociety.org/10.1109/FCCM.2011.30.
[10] A. H. T. Tse, D. B. Thomas, K. H. Tsoi, W. Luk, Reconfigurable control variate monte-carlo designs for pricing exotic options, in: Proceedings of the 2010 International Conference on Field Programmable Logic and Applications, FPL '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 364–367.
[11] K. S. Kannan, P. S. Sekar, M. M. Sathik, P. Arumugam, Financial stock market forecast using data mining techniques, Computer I (2010) 555–559.
[12] S. Langdell, Examples of the use of data mining in financial applications.
[13] M. Reaz, S. Islam, M. Ali, M. Sulaiman, Fpga realization of backpropagation for stock market prediction (2002) 960–964.
[14] SciEngines GmbH, `http://www.sciengines.com`.
[15] J. Fan, D. V. Bailey, L. Batina, T. Güneysu, C. Paar, I. Verbauwhede, Breaking Elliptic Curve Cryptosystems Using Reconfigurable Hardware, in: IEEE Field Programmable Logic, 2010, pp. 133–138.
[16] M. Schimmler, L. Wienbrandt, T. Güneysu, J. Bissel, COPACOBANA: A Massively Parallel FPGA-Based Computer Architecture, in: B. Schmidt (Ed.), Bioinformatics – High Performance Parallel Computer Architectures, CRC Press, 2010, pp. 223–262.
[17] L. Wienbrandt, S. Baumgart, J. Bissel, F. Schatz, M. Schimmler, Massively parallel FPGA-based implementation of BLASTp with the two-hit method, in: ICCS2011, Procedia Computer Science, Vol. 1, 2011, pp. 1967–1976.
[18] L. Wienbrandt, S. Baumgart, J. Bissel, C. M. Y. Yeo, M. Schimmler, Using the reconfigurable massively parallel architecture COPACOBANA 5000 for applications in bioinformatics, in: ICCS2010, Procedia Computer Science, Vol. 1, 2010, pp. 1027–1034.
[19] Xilinx Inc., Xilinx UG331 Spartan-3 Generation FPGA User Guide, `http://www.xilinx.com` (Mar. 2011).
[20] Intel Corporation, Intel Core i7-900 Desktop Processor Series, `http://www.intel.com`.