Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 646984, 10 pages
doi:10.1155/2012/646984

Research Article

Optimizing Investment Strategies with the Reconfigurable Hardware Platform RIVYERA

Christoph Starke, Vasco Grossmann, Lars Wienbrandt, Sven Koschnicke, John Carstens, and Manfred Schimmler

Department of Computer Science, Christian-Albrechts-University of Kiel, 24098 Kiel, Germany

Correspondence should be addressed to Christoph Starke, chst@informatik.uni-kiel.de

Received 30 September 2011; Accepted 4 January 2012

Academic Editor: Thomas Steinke

The hardware structure of a processing element used for optimization of an investment strategy for financial markets is presented. It is shown how this processing element can be multiply implemented on the massively parallel FPGA-machine RIVYERA. This leads to a speedup of a factor of about 17,000 in comparison to one single high-performance PC, while saving more than 99% of the consumed energy. Furthermore, it is shown for a special security and different time periods that the optimized investment strategy delivers an outperformance between 2 and 14 percent in relation to a buy and hold strategy.

1. Introduction

The goal of technical financial market analysis is to predict the development of indices, stocks, funds, and other securities by evaluating the charts of the past. A method to find such predictions can lead to an investment strategy. Many well-known chart-analysis methods (e.g., Elliot waves [1], Bollinger Bands [2]) try to extract patterns from the charts, expecting that such patterns will come up in similar ways again in the future. There are more than 100 different chart-analysis methods but their success is doubted [3]. In most cases, the current development of the markets significantly affects the quality of the different investment strategies. Since the business volume per year on the worldwide stock markets is more than USD 35 trillions [4], it is not surprising that successful investment strategies are in the focus of intensive research.

In general, there are lots of indicators influencing the chart of a security. Those are not only economical and political indicators but also psychological ones. It is very difficult to decide which weight should be assigned to which indicator, the more so as there are known and unknown tradeoffs between different indicators. Furthermore, weights change in time. Recent papers [5–8] try to apply data mining methods on historical market rates, in order to find investment strategies that perform significantly above the average. This approach is extreme compute-intensive since every day there are millions of quotations that are fixed worldwide. Even with the use of high-performance computers the reduction of this amount of data is required. But as shown in the literature [5–8], data mining helps to keep the essential information contents in order to come to successful investment strategies.

In this paper, we present an investment strategy using a novel data mining method, which is discussed in Section 2. It results in a performance significantly above average for certain periods. It is based on the idea of an iterative search for an optimal set of indicator weights in the space of all possible weights of the indicators. Since this space grows exponentially with the number of indicators, the method is very compute-intensive but can optimally be parallelized. Therefore, an FPGA implementation seems to be very promising, because the computational core can be kept very simple and small in hardware. The two phases of the method have been implemented on the FPGA-based massively parallel computer RIVYERA. The RIVYERA architecture and its idea of efficiently exploiting 128 modern FPGAs in parallel are explained in Section 3.

Section 4 describes the architecture and the implementation of one processing element for the main computation. The speedup achieved by such an FPGA approach is investigated in Section 5. For different time intervals, the advantage in comparison to a single buy-and-hold strategy is determined. We do not want to discuss the investment strategy itself in this paper. Instead, the main focus here lies on the improvement in terms of speed, energy, and cost efficiency of the new method in comparison to an implementation on a sequential computer architecture. Section 6 summarizes the results and concludes the paper.

2. The Process of Optimizing an Investment Strategy for Securities

For a single security P a successful strategy for buying and selling is desired. For simplicity, in this paper let P be an investment fund that can be traded without trading costs (there are several discount brokers offering such funds, e.g., Vanguard in USA, InvestSMART Financial Services Pty Ltd in Australia, European Bank for Fund Services GmbH in Germany). Since the taxation regulations are varying in different countries, these are not considered here either.

We consider n indicators $I_0, I_1, \ldots, I_{n-1}$ that might have influence on the chart of P. Typical indicators are S&P500, Nikkei225, EuroStoxx50, EUR/USD, and so forth. In other methods of technical analysis P itself is used as an indicator as well. We consider a time interval of the past consisting of $m+1$ subsequent trading days $d_0, d_1, \ldots, d_m$. m should be large enough to get significant results (e.g., $m \geq 125$).

R is an $m \times n$ matrix, where $R_{i,j}$ is the percentage difference of the indicator I_j from d_{i-1} to d_i. The vector R_i is the ith row of the matrix R: $R_i = (R_{i,0}, R_{i,1}, \ldots, R_{i,n-1})$. The required data for such a matrix can either be collected or downloaded from some trading platform in the internet.

At time d_0 we assume a cash capital of one million EUR and a depot with $D_0 = 0$ pieces of the security P. The value of one million has been chosen in order to be able to abstract from rounding errors. The results for other starting values can be computed proportionally. Generally, let C_i be the cash money, D_i the number of pieces of P in the depot, and Z_i the total property at day d_i ($0 \leq i \leq m$). The fund considered here has exactly one market price P_i per day. Therefore, the following condition holds:

$$Z_i = C_i + D_i \cdot P_i. \tag{1}$$

We are looking for a function $f(R_i)$ that computes the decision of buying or selling a certain amount of P from the values of R known up to d_i. The output of f is on the one hand the decision either to buy, to do nothing, or to sell, and, on the other hand, the amount for the first and the last case. The optimal function of this kind is the one that maximizes the value of Z_m. This approach is motivated by the assumption of technical analysis that a successful strategy of the past will also be successful in the future.

In order to simplify the search for f, we consider only functions of the kind $f(R_i) = \sum w_j \cdot R_{i,j}$. The weight vector is denoted by $w = (w_0, w_1, w_2, \ldots, w_{n-1})$. We define w^* as the vector which yields a maximum value for Z_m. A positive value of f is a buy indication, a negative sell indication. The amount of P to be traded is $Z_{i-1} \cdot |f(R_i)|$. A buy at day d_i is limited to C_{i-1} in order not to overdraw the cash account and a sell is limited to D_{i-1}, accordingly. The decision to cut down on functions of the kind $f(R_i) = \sum w_j \cdot R_{i,j}$ is based on the assumption that the influence of the different indicators is almost linear. Although this cannot be proven here, the results with this simplification are already remarkable. However, it is still worth to investigate modifications of this method with nonlinear functions.

There is one problem with investment funds: at that point in time at the day d_i where the trading decision is made, the exact value of P_i is not known. Therefore, at this point of time it is not possible to compute the exact number of pieces to be traded without overdrawing the cash account. For a buy order, we therefore transmit not the number of pieces but the amount of money for which we want to buy pieces. Vice versa, for a sell order, we should transmit the exact number of pieces to be sold.

Let B_i be the amount of money for which pieces of P should be bought at d_i and S_i the number of pieces to be sold at d_i. Obviously, the following condition holds: $(B_i = 0) \vee (S_i = 0)$.

Furthermore, if $f(R_i) \geq 0$, it holds:

$$\begin{aligned} B_i &= \min\left(Z_{i-1} \cdot f(R_i), C_{i-1}\right), \\ S_i &= 0, \\ C_i &= C_{i-1} - B_i, \\ D_i &= D_{i-1} + \frac{B_i}{P_i}. \end{aligned} \tag{2}$$

And if $f(R_i) < 0$,

$$\begin{aligned} B_i &= 0, \\ S_i &= \min\left(Z_{i-1} \cdot \frac{|f(R_i)|}{P_{i-1}}, D_{i-1}\right), \\ C_i &= C_{i-1} + S_i \cdot P_i, \\ D_i &= D_{i-1} - S_i. \end{aligned} \tag{3}$$

It is computationally unfeasible to determine w^* with a brute force approach, even on a supercomputer. If one considers only 100 different values for each of 8 indicators, then there are 100^8 different combinations of those. Using a calibration period of 26 weeks, with 5 trading days per week the required number of computations of the function f would be

$$26 \cdot 5 \cdot 10^{16} = 1.3 \cdot 10^{18}. \tag{4}$$

As specified in the following examinations, even the presented RIVYERA implementation would require 377 days to evaluate that number of combinations. Instead of such a brute force method, we use an iterative approach: initially a very rough grid of 16 values per indicator is used. For each of the 16^8 weight vectors, the final property Z_m is computed. The areas in the grid, where Z_m is relatively large, are the targets of the next iteration: in the environment of

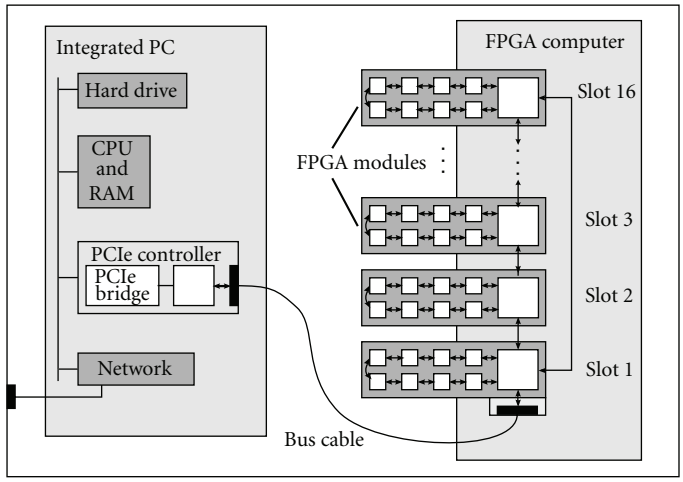

Figure 1: RIVYERA S3-5000 hardware structure.

the corresponding weights the grid, is refined. If a component of a promising weight vector is at the boundary of the grid then the grid is extended in this direction. In the same way, we keep on refining the grid. Already after 100 iteration steps, the results are satisfying. In each iteration step 16^8 weight vectors are considered, resulting in

$$26 \cdot 5 \cdot 16^8 \approx 5 \cdot 10^{11} \tag{5}$$

calculations of the function f plus the resulting calculation of the development of the depot value under the assumption that the corresponding buying and selling decisions are taken into account. The process of refining the grid is part of the host system and not specified in this paper.

The high computational effort is caused by the exhaustive search concerning the evaluation of w^*. This calculation can be remarkably accelerated by an FPGA-based implementation. Hence, the concrete task is the following: How can we accelerate the identification of the optimal weight vector w^* which maximizes Z_m out of a given set of weight vectors?

3. FPGA-Based Hardware Platform RIVYERA

Introduced in 2008, the massively parallel FPGA-based hardware platform RIVYERA [9] is the direct successor of the COPACOBANA, presented in 2006 for cost optimized breaking 56 bit DES ciphers in less than two weeks [10]. Besides applications in cryptanalysis (e.g., [11]), RIVYERA finds its applications in the fields of bioinformatics [12–14] and now stock market analysis, as described in this paper.

For the application presented here, the specific RIVYERA S3-5000 is used, distributed by SciEngines GmbH [15]. RIVYERA is designed to be a completely scalable system consisting of two basic elements. Firstly, the in-built multiple FPGA-based supercomputer provides the resources for parallel high-performance applications (Figure 1, right side). Secondly, a standard server grade mainboard equipped with an Intel Core i7-930 processor, 12 GB of RAM, and 2 TB of hard disk space, provides the resources for quick pre- and postprocessing purposes (Figure 1, left side). The RIVYERA S3-5000 is powered by two 650 W supplies and packed in a standard rack mountable 3 U housing. It is running a standard Linux operating system and, therefore, presents an independent system. The details are discussed briefly in the following.

The FPGA-based supercomputer consists of a backplane and up to 16 FPGA cards (fully equipped for the application described in this paper). Each FPGA card is equipped with eight user configurable Xilinx Spartan3-5000 type FPGAs and one additional FPGA as communication controller. In total, these are 128 user configurable FPGAs. Additionally, a DRAM module with a capacity of 32 MB is directly attached to each user FPGA.

All FPGAs are connected by a systolic-like bus system. Each FPGA on an FPGA card is connected with two neighbors forming a ring including the communication controller. The FPGA card slots are connected to each neighboring slot as well on the backplane, providing the connections between the communication controllers on each FPGA card. The communication is physically realized by high-throughput symmetric LVDS point-to-point connections. The communication of the FPGA-based computer to the host-mainboard follows a connection via PCIe controller card directly to a communication controller on a chosen FPGA card. For applications requiring a higher bandwidth from the host system to the FPGA-based computer, more than one PCIe controller may be attached to other FPGA cards as well. For a configuration as used for this application, the measured net bandwidth from the host to the FPGA computer reaches up to 66 MB/s. Of course, the latency will be different dependent on which clients are communicating with each other, according to the length of the communication chain.

For application development, the RIVYERA provides an API for each of the two basic elements, that is, an API controlling the data transfer between the host software and the FPGAs including broadcast facilities, and an API for the user defined hardware configuration of the FPGAs

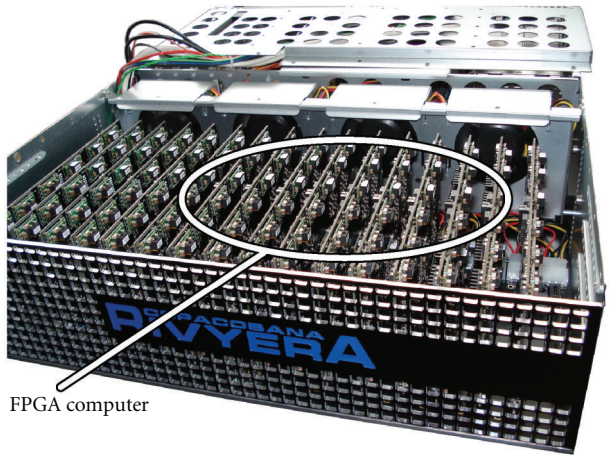

Figure 2: RIVYERA S3-5000. The 16 FPGA-cards forming the FPGA-computer are highlighted. The integrated standard PC cannot be seen behind the cover.

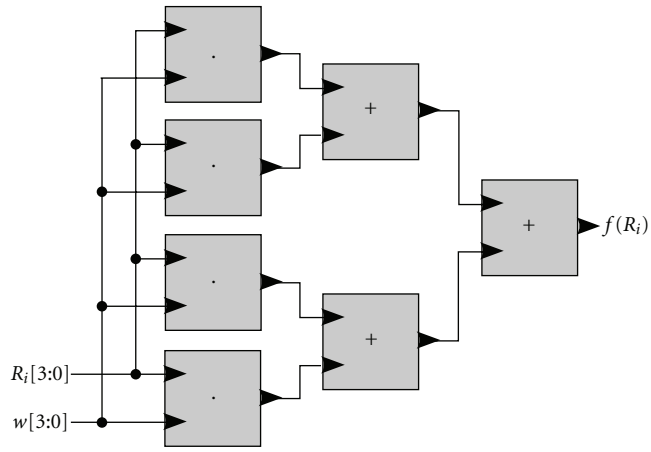

Figure 3: Calculation of $f(R_i)$ with four indicators.

controlling the data transfer to other FPGAs and the host as well.

A picture of the RIVYERA S3-5000 is shown in Figure 2.

4. Processor Architecture

The FPGA-based part of the presented algorithm is based on exhaustive searches. As different weight vectors can be evaluated independently, the algorithm is suitable for massive parallelization. Therefore, the following description of the technical implementation only considers a single FPGA. Assuming uniform programming of all available FPGAs and an equally divided search space, the computational speed rises approximately linear with the number of FPGAs. According to the RIVYERA platform, the implementation presented here is optimized for Xilinx Spartan3-5000 FPGAs [9, 16].

The key aspect concerning the identification of valuable weight vectors is the calculation of the score Z_m for every possible element of the search space. Since these evaluations are the fundamental issue of the computational effort, the success of creating an efficient processor architecture is directly linked to the performance of the underlying implementation of the scoring function. Thus, the main objective, and therefore starting point for the design of the processing element, should be the creation of a scoring unit with a high throughput.

4.1. Scoring Pipeline. The evaluation of Z_m consists of repetitive computations of the sequences C_i and D_i. Therefore, the throughput of the scoring unit is directly connected to the performance of the computation of these two sequences. Thus, despite the high spatial cost, the advantages of a pipeline architecture are persuasive. The implementation presented here is based on pipelines that yield a new pair (C_i, D_i) in every clock cycle. As the values C_i and D_i are defined recursively, the pipeline has to wait for its own outputs. Thus, to avoid idle time, l scores for different weight vectors are evaluated concurrently, where l is the length of the longest cyclic path. Hence, l is given by the number of clock cylces that are necessary to compute C_i and D_i from C_{i-1} and D_{i-1}.

Basically, the structure can be subdivided into three segments. The first one is described by the function $f(R_i) = \sum w_j \cdot R_{i,j}$. Assuming n indicators, the calculation of $f(R_i)$ needs n multiplications and $n-1$ additions. The corresponding structure for $n = 4$ is shown in Figure 3. As all following calculations directly depend on $f(R_i)$, this computation is part of the longest path of the pipeline. Hence, the additions should be arranged in a way that only the minimum number of steps (1 multiplication and $\lceil \log_2 n \rceil$ additions) is required. However, the path is not element of the longest cyclic path because w_j and $R_{i,j}$ do not depend on the outputs of the pipeline. A different arrangement of the additions has no effect to l.

Due to resource reduction the buy order size B_i and the sell order size S_i are combined to a general order size O_i. A negative value indicates a sell order, and a positive value denotes a buy order. Instead of the sequence D_i, the pipeline calculates the values $D'_i = D_i \cdot P_i$ as it enables to the evaluate Z_m just by one addition. The calculation of O_i is given by the following instruction:

$$O_i := \begin{cases} -D'_i, & \text{if } O^*_i \le -D'_i, \\ C_i, & \text{if } C_i \le O^*_i, \\ O^*_i, & \text{else.} \end{cases} \tag{6}$$

When the evaluation of $f(R_i)$ is finished, the order size O_i at day i is computed as shown in Figure 4. 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. The total property $C_i + D'_i$ and the negative depot value $-D'_i$ do not depend on $f(R_i)$ and, thus, can be calculated in parallel to its evaluation. Hence, the longest path of the pipeline is extended by the multiplication and the comparator chain.

After the calculation of the order size, new cash and depot values are computed. The value i that identifies the day of

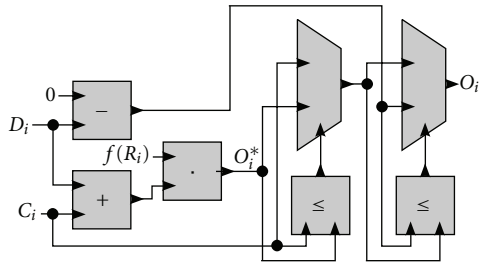

Figure 4: Calculation of order size O_i.

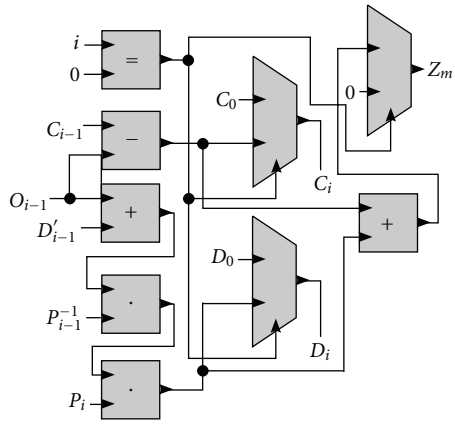

Figure 5: Evaluation of Z_m.

the historical data set rises by 1 since the end of the given time period ($i = m$) is reached. In this case i is set to 0 which implies the start of the evaluation of a new weight vector. The sequences C_i and D'_i 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:

$$(C_i, D'_i) := \begin{cases} (C_0, D'_0) & \text{if } i = 0, \\ \left(C_{i-1} - O_{i-1}, (D'_{i-1} + O_{i-1}) \cdot \dfrac{P_i}{P_{i-1}}\right) & \text{else.} \end{cases} \quad (7)$$

In comparison to a multiplication, a division is much more expensive in regard to resource usage [16]. As a consequence, the quotient P_i/P_{i-1} is realized as the multiplication $P_i \cdot P_{i-1}^{-1}$. On the one hand, this implies the additional calculation and storage of inverse elements. On the other hand, every calculation needs to be done only once and can be outsourced to the host system. Likewise, the additive memory usage can be disregarded as we will see in Section 4.3.

As considered, the algorithm is trivially parallelizable. The computational speed depends linearly on the number of FPGAs. Likewise, this statement can be assigned on the number of pipelines. But how many pipelines can be synthesized on an FPGA and are there further possibilities to increase that number?

Table 1: Synthesis result with floating point representation.

Indicators	Slices	Multipliers
8	8584 (25%)	32 (30%)
16	14064 (42%)	44 (42%)

4.2. Optimized Fixpoint Representation. All in all, one scoring pipeline is built of $n + 3$ multiplications, $n + 2$ additions, 2 subtractions, 3 comparators, and 5 multiplexers where n is the number of indicators (see Figure 6) that are $15 + 2n$ operations. A Spartan 3–5000 FPGA consists of 8,320 Configurable Logic Blocks which can be separated in 33,280 Slices [16]. Additionally, 108 dedicated 18×18 bit multipliers can be assigned for synthesis.

The allocation report of two synthesis results is shown in Table 1. A single precision floating point representation of all variables is assumed in both cases. Using 32 multipliers, 8 indicators yield a consumption of 25% of the available slices. Assuming that 10% of the slices are reserved for further control units, three pipelines can be synthesized on the FPGA. In case of 16 indicators, additional 17% are required. The drastic increase results from the 8 additional adders and multiplicators and the comparatively high spatial cost of floating point units [16]. An important point is the synchronization of the different pipeline stages. For example, the third stage (see Figure 5) receives, amongst others, the input values O_{i-1} and P_i. While P_i is given, O_i is only known after several calculations. Hence, to provide synchronicity, the transfer of P_i is delayed using shifting registers. The longest cyclic path consists of 2 additions, 3 multipliers, 2 comparators, and 3 multiplexers. As an extension of the pipeline implies the requirement of more shifting registers, the path should be as short as possible. Optimized in terms of space, the longest cyclic path comprises $l = 57$ clock cycles. All in all, only two pipelines are possible in this case.

To counter that problem, a fixpoint representation will be introduced in the following. The idea is motivated by the fact that many of the given values are located in limited ranges. For example, the daily price fluctuations in R rarely exceed the interval $[-10\%, 10\%]$. That is the reason why the values of R will be stored in 18 bits where the decimal place is coded in 12 bits and the new codomain is the interval $[-32\%, 32\%)$ with a precision of $2^{-12}\%$. Likewise, the elements of the weight vector will be stored in 18 bits. While a decimal place of 12 bits seems to be the best tradeoff between overflow immunity on the one hand and precision on the other hand, the range of the weight vectors may be determined specifically for every use case. Cash, depot value, and stock prices are stored in integer values in cent. The inverse prices are multiplied with 2^{32} and also stored in integer values.

The 18-bit representation of $R_{i,j}$ and w_j promotes the efficient usage of the dedicated 18×18 multipliers. Furthermore, the transfer from floating point to fixpoint units leads to a considerable decrease of the allocated resources.

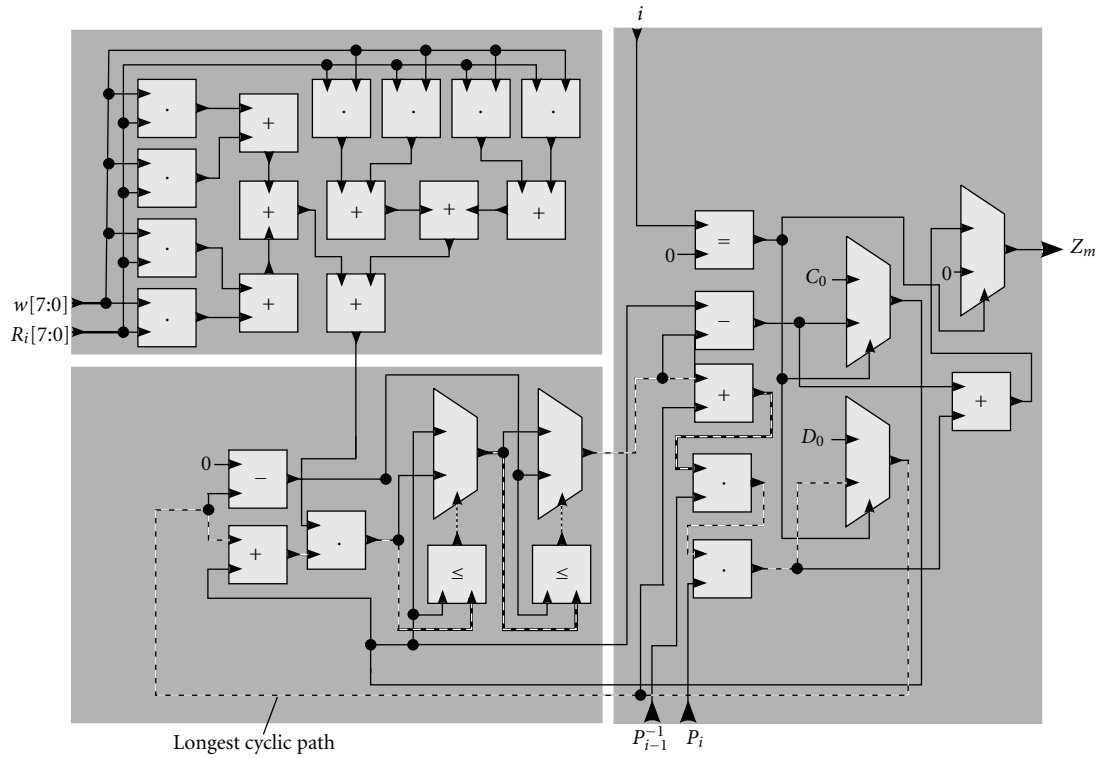

Figure 6: Scoring pipeline for 8 indicators.

The length of the longest cyclic path can be reduced to 37. As shown in Table 2, the available resources suffice for up to 6 pipelines per FPGA.

4.3. FPGA Overview. The pipelines are triggered synchronously. The trading period d and the corresponding historical data are set globally for all scoring units. Since l independent score evaluations are calculated in parallel in every pipeline, the value of d has to change only once every l clock cycles. To trigger the pipeline in the $i-1$th recursion, the historical information of day i is necessary. This set consists of the vector of price fluctuations R_i and the values P_i and P_{i-1}^{-1}. To transfer these values within one clock cycle, the historical data of day i H_i is stored in a single Block RAM word. Such a word $H_i = (P_{i-1}^{-1}, P_i, R_i)$ consists of $32+32+18\cdot n$ bits, for example, 208 bits for $n = 8$ indicators. Spartan3-5000 provides 104 RAM blocks with 1,872 KB in total [16]. This is obviously enough in our case, as it suffices for over 9000 days relating to 8 indicators.

As the optimization is based on an exhaustive search, it is necessary to determine the search space. The declared objective is to identify the optimal weight combination for 8 indicators. 8 possible candidates are given for every indicator. So, the search space is declared by an 8×8 matrix. Every row describes an indicator and consist of 8 values. Each of these values can be used as a weight to the correspondent indicator. As there are 8 possible candidates for each of 8 indicators, the number of possible weight vectors is $|W| = 8^8 \approx 16.7$ million. So, one FPGA is able to calculate the optimal weight vector out of 16.7 million combinations. One unique combination for every pipeline has to be calculated in every clock cycle. To accomplish this, every possible combination is declared by an 24-bit identifier in the range of $[0, |W|-1]$. An equally divided subspace is $([0, |W|/6 - 1], [|W|/6, |W|/3 - 1], \ldots]$ assigned to every pipeline. The weight vector is extracted by masking the identifier. The bits $3\cdot j+2$ to $3\cdot j$ show the position of coefficient w_j of the weight vector w. For example, the identifier $387_{10} = 110.000.011_2$ references the matrix items $W[0][3]$ $(011_2 = 3_{10})$ for w_0 and $W[2][6]$ for w_2. This interpretation is very efficient as the effort of bit masking is comparatively small. Thus, 6 different weight vectors can be selected in a single clock cycle and assigned to the pipelines.

Table 2: Synthesis result with fixpoint representation.

Indicators	Slices	Multipliers
8	4801 (14%)	12 (11%)
16	5395 (16%)	20 (19%)

As the data flow is synchronous, the scores Z_m of all pipelines are calculated at the same time. Assuming 6 pipelines, 6 results are returned per clock cycle. Obviously, it is neither possible nor does it make sense to store 8^8 values. Likewise, the effort to administrate a list of the best scores is too high as it implies the sorting of 6 results into the list in a single clock cycle. The examination of this problem shows that a good tradeoff is the storage of the best result of every pipeline. Utilizing 6 pipelines and 128 FPGAs, 768

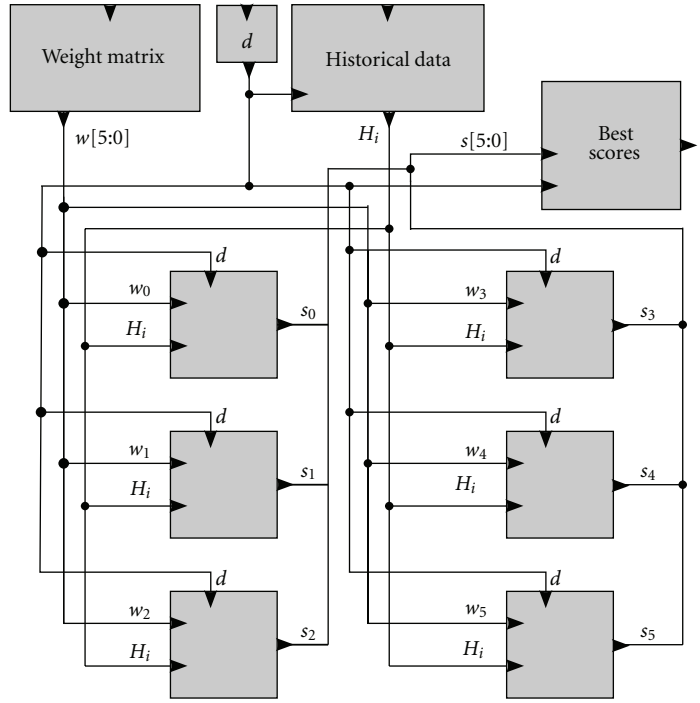

Figure 7: Outline of the processor architecture.

results are evaluated in every iteration. This set seems to be widespread enough to calculate new weight coefficients for the next iteration. An overview of the FPGA structure is shown in Figure 7.

5. Results and Performance Analysis

For further research, we will now consider results and performance for a certain security, the investment fund DWS Convertibles, ISIN DE0008474263, that is operating internationally.

As described in Section 2 (referred as calibration phase in the following), the optimal weight vector w^* is determined for the security and furthermore for a randomly chosen time period of 26 weeks (calibration time interval). We chose 8 indicators that widely represent the current economical environment: S&P 500, DAX, EuroStoxx 50, ASX 200, Nikkei 225, Hang Seng, S&P 500 Future, and EUR/USD. The goal is to find w^* with the maximal value of Z_m.

The computational effort with 8 indicators is already rather high. In this paper, we disclaim to investigate more indicators since, on the one hand, these 8 indicators represent the activities on the international stock markets to some high degree, and on the other hand, the results with this restriction are already remarkable.

We now focus on the investment strategy where at day d_i for indicators I_j the $R_{i,j}$ are calculated and then from $f(R_i)$ the volume of buying or selling orders is computed based on the value of w^*. To determine the quality of the vector w^*, we test it in a different period of time referred to as the evaluation phase. Of course, this makes only sense for a time interval (the so-called evaluation time interval) which does not overlap with the calibration time interval. We have chosen three different evaluation time intervals of 26 weeks as well. The question is whether or not the new investment strategy gives an outperformance in comparison to a buy-and-hold strategy. Buy-and-hold means P is bought at the beginning of the evaluation time interval and sold at the end.

Figure 8 shows an example of the chart of the security in comparison to the performance of our investment strategy with the same security within three different evaluation time intervals T_k of 26 weeks each, $k \in \{1, 2, 3\}$. T_1 (2009-09-14–2010-03-15) is a period where tendency for the fund is rising. T_2 (2010-09-27–2011-03-28) is a period without a clear tendency and T_3 (2011-03-28–2011-09-26) is a period where tendency for the fund is falling.

The values of w_k^* had been determined for each T_k in the iterative way described previousuly. The resulting investment strategy S_k was then applied for the evaluation time intervals T_e, where $e \in \{1, 2, 3\}$ and $e \neq k$. The chart $P_{k,e}$ shows the performance of the monetary assets in the evaluation time interval T_e using investment strategy S_k.

In all time intervals, an outperformance of the investment strategies S_k over P between 2% (see $P_{2,1}$ in Figure 8) and 14% (see $P_{1,3}$ in Figure 8) can be seen. Although this is no proof in a mathematical sense that such an investment strategy can be applied to arbitrary securities in arbitrary time periods, it seems to be very promising to further improve the method described here.

Considering computing performance as well, the RIVYERA or similar computer architectures are perfectly suited for such research. Table 3 shows a comparison between the RIVYERA-based approach and a PC version of the algorithm implemented in C. The test system uses an Intel Core i7–970

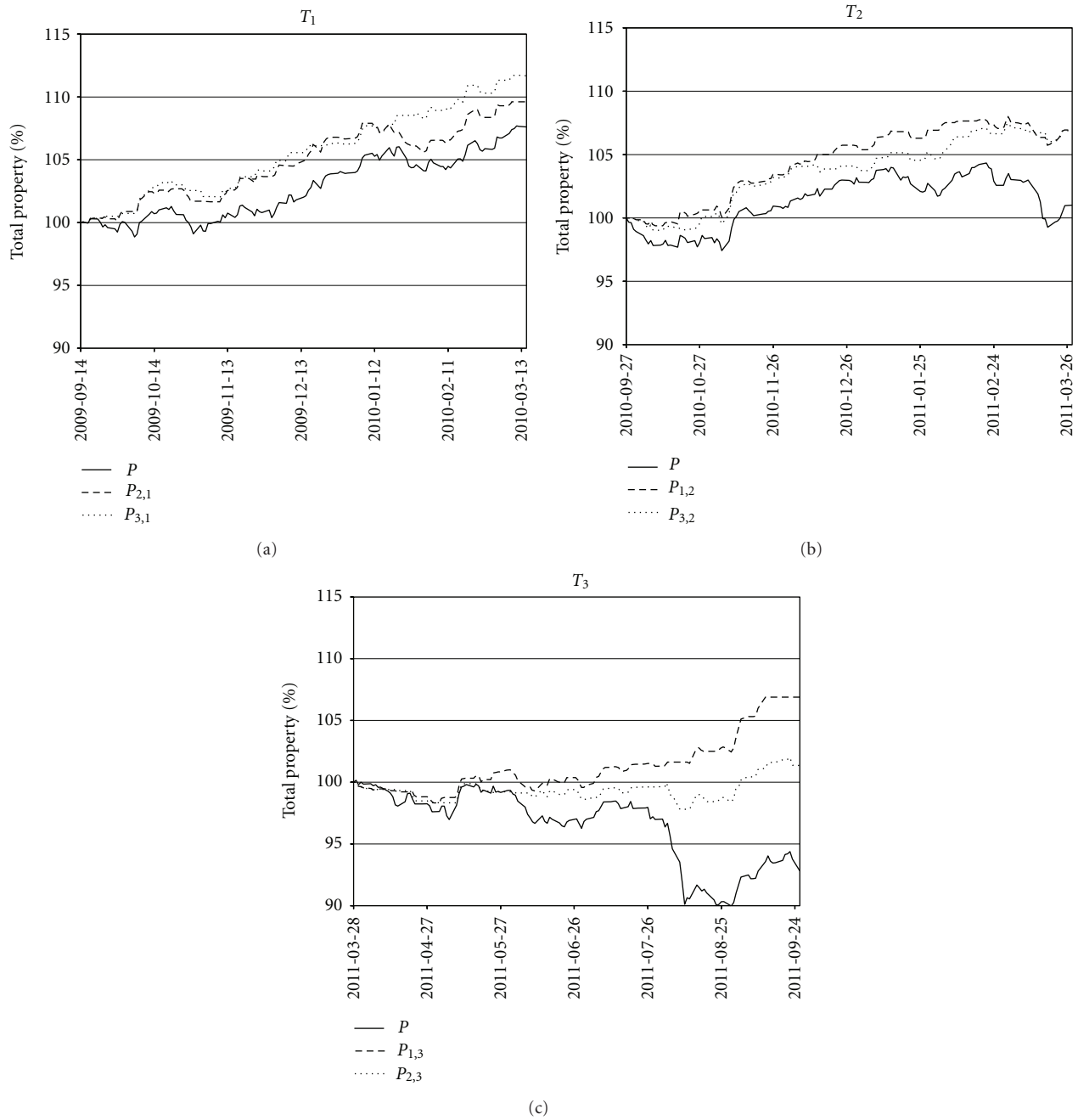

Figure 8: Performance of the investment strategy in different evaluation intervals.

with 6×3200 MHz, an ASRock X58 Extreme mainboard and 8 GB GeIL DIMM DDR3-1066 RAM. The implementation uses all cores of the processor. In addition, the improved number representation is used in the PC version as well.

$128 \cdot p$ pairs (C_i, D_i') are calculated in every clock cycle on RIVYERA where p is the number of pipelines per FPGA. The clock rate of the implementation is 50 MHz. Assuming 8 indicators, 6 pipelines can be synthesized on an FPGA. This yields $128 \cdot 6 \cdot 50,000,000 = 38.4$ billion pairs per second. Examinations of the PC version denote that 2.26 million calculations per second are possible on the specified test system. The conclusion is a speedup of about 17,000. While RIVYERA requires up to 1300 W, 300 W is supposed for a standard PC. Accordingly, the power consumption is reduced by up to 99.975%.

Of course, such a comparison yields a number of questions. Intel declares 76.8 GFLOPs for i7–970 [17]. The presented FPGA design needs $15+2n = 31$ operations for $n = 8$ indicators. Assuming that the PC version manages to work with the same number of operations, one could deduce that the referred processor reaches up to $78,600,000,000/31 = 2.54$ billion pairs per second. This is more than 1,000 times

Table 3: Comparison of PC and RIVYERA.

Runtime (calibration phase: 26 weeks)		
Number of weight vectors	PC	RIVYERA
1 billion	44 h 56 m	9.15 s
50 billion	93 d 14 h	7 m 37 s
1 trillion	5.13 years	2 h 32 m
Power consumption		
Number of weight vectors	PC (300 W)	RIVYERA (1300 W)
1 billion	13.48 KWh (2,70€)	3.31 Wh (0,0006€)
50 billion	0.674 MWh (137,49€)	165.20 Wh (0,04€)
1 trillion	13.48 MWh (2695,80€)	3.31 KWh (0,66€)

faster than the actual implementation. So, what is the reason for this gap?

In fact, the computing power of the processor is not the bottleneck. The main problem is located in the intensive memory communication. Even a cache-optimized version needs several RAM accesses (and of course many cache accesses) to calculate a single pair. The pipeline structure cannot directly be translated but only be simulated by further memory instructions. In contrast, the FPGA Block RAM modules are triggered in parallel to the actual calculations. Thus, there is absolutely no latency concerning memory operations. This is a reason why this algorithm is very suitable for massively parallel computing. A further interesting issue would be a comparison in performance using GPGPU.

As well, the time complexity of the presented algorithm differs in regard to the different platforms. On standard processors, the complexity is $\mathcal{O}(w_{\max}^n \cdot n \cdot m)$ where $w_{\max}$ denotes the maximum number of possible coefficients for one indicator. The factor n occurs because the evaluation of $f(R_i)$ needs n multiplications and $n-1$ additions that has to be executed sequentially. A RIVYERA pipeline calculates one pair (C_i, D_i') per clock cycle in every case. So, there is obviously no such dependency. Therefore, the time complexity is $\mathcal{O}(w_{\max}^n \cdot m)$. However, the dependency on n is not erased by this approach. While the size of a standard processor remains constant for an increasing n, more adders and multipliers are necessary in terms of an FPGA-based implementation. According to this, the spatial complexity is $\mathcal{O}(n)$. As this may lead to less pipelines per FPGA, an indirect influence to the runtime cannot be concealed.

6. Conclusion

The FPGA-machine RIVYERA is very suitable for optimization of the investment strategy as it was presented in this paper. A speedup of 17,000 and an energy saving of more than 99% in comparison to one single high-performance PC has been determined. The investment strategy which is optimized with RIVYERA delivers for the special investment fund and different time periods reviewed a significant outperformance in relation to a buy and hold strategy.

Several other securities for different time periods were tested. Although always the same, simple indicators were used, the optimization of the investment strategy by using RIVYERA delivered almost in every case a significant outperformance.

References

[1] R. N. Elliott, *The Wave Principle*, 1938.

[2] J. Bollinger, *Bollinger on Bollinger Bands*, McGraw-Hill, 2001.

[3] C. Park and S. Irwin, "What do we Know about the profitability of technical analysis?" *Journal of Economic Surveys*, vol. 21, pp. 786–826, 2007.

[4] M. Nagler, *Finanzcrash und Systemkrise*, 2008.

[5] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani, "Mining the stock market: which measure is best?" in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[6] K. S. Kannan, P. S. Sekar, M. M. Sathik, and P. Arumugam, "Financial stock market forecast using data mining techniques," in *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS '10)*, vol. 1, pp. 555–559, 2010.

[7] S. Langdell, "Examples of the use of data mining in financial applications," in *Financial Engineering News*, p. 2002.

[8] T. Rathburn, *Data Mining the Financial Markets, Part 1*, Data Mining, Down Under Forums, 2007.

[9] G. Pfeiffer, S. Baumgart, J. Schröder, and M. Schimmler, "A massively parallel architecture for bioinformatics," in *Proceedings of the 9th International Conference on Computational Science (ICCS '09)*, vol. 5544 of *Lecture Notes in Computer Science*, pp. 994–1003, Springer, 2009.

[10] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, and M. Schimmler, "How to Break DES for €8,980," in *Proceedings of the Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems*, Cologne, Germany, 2006.

[11] J. Fan, D. V. Bailey, L. Batina, T. Guneysu, C. Paar, and I. Verbauwhede, "Breaking elliptic curve cryptosystems using reconfigurable hardware," in *IEEE Field Programmable Logic*, pp. 133–138, 2010.

[12] M. Schimmler, L. Wienbrandt, T. Guneysu, and J. Bissel, "COPACOBANA: a massively parallel FPGA-based computer architecture," in *Bioinformatics High Performance Parallel Computer Architectures*, B. Schmidt, Ed., pp. 223–262, CRC Press, 2010.

[13] L. Wienbrandt, S. Baumgart, J. Bissel, F. Schatz, and M. Schimmler, "Massively parallel FPGA-based implementation of BLASTp with the two-hit method," *Procedia Computer Science*, vol. 4, International Conference on Computational Science (ICCS '10), pp. 1967–1976, 2011.

[14] L. Wienbrandt, S. Baumgart, J. Bissel, C. M.Y. Yeo, and M. Schimmler, "Using the reconfigurable massively parallel architecture COPACOBANA 5000 for applications in bioinformatics," *Procedia Computer Science*, vol. 1, International Conference on Computational Science (ICCS '10), no. 1, pp. 1027–1034, 2010.
[15] SciEngines GmbH: http://www.sciengines.com/.
[16] Xilinx Inc.: Xilinx UG331 Spartan-3 Generation FPGA User Guide, March 2011, http://www.xilinx.com/.
[17] Intel Corporation: Intel Core i7-900 Desktop Processor Series, http://www.intel.com/.