Parallel Genetic Algorithms for Financial Pattern Matching using NVIDIA GPU's

João Baúto
Instituto de Telecomunicações
Instituto Superior Técnico
Lisboa, Portugal
joao.bauto@tecnico.ulisboa.pt

Nuno Horta
Instituto de Telecomunicações
Instituto Superior Técnico
Lisboa, Portugal
nuno.horta@tecnico.ulisboa.pt

Rui Neves
Instituto de Telecomunicações
Instituto Superior Técnico
Lisboa, Portugal
rui.neves@tecnico.ulisboa.pt

***Abstract*—Pattern recognition or matching algorithms process large datasets of information that can be either images for recognition or time series in pattern matching. To gather accurate results, these algorithms are trained using advanced techniques of machine learning such as Neural Networks (NN) and Genetic Algorithms (GA) that require extensive periods of time and computational power.**

This paper presents a study of SAX/GA, an algorithm to optimize market trading strategies, to understand how the sequential implementation of SAX/GA and genetic operators work to optimize possible solutions. This study is later used as the baseline for the development of parallel techniques capable of exploring the identified points of parallelism that simply focus on accelerating the heavy duty fitness function to a full Graphical Processing Unit (GPU) accelerated GA. The implemented solutions accelerated the sequential SAX/GA solution in around 30× with a maximum of nearly 180×.

Keywords—Parallel Genetic Algorithms; Technical Analysis; Financial Computation; Pattern Matching; GPU; CUDA

I. INTRODUCTION

The financial system as it is currently known does not come from an idea invented one century ago but from the human ideal of trading goods. It is an idea that evolved into the current stock markets where goods are traded for a monetary value. The main objective of participating in trading in financial markets is to maximize the Return on Investment (ROI). There are two possible decisions that an investor can make, either play the safe and low risk game of buying a stock and hold it for a long period of time, also known as Buy & Hold strategy, or enter in the high risk high reward play of designing a trading strategy that involves multiple entries (open position) and exits (close position) in the market.

Creating a custom trading strategy that uses patterns as decisions points of entry or exit on the market can be a tedious and long process of optimization where multiple possible decisions sets must be tested against large time series. Researchers have been trying to ease the optimization process by reducing datasets while maintaining a precise representation with minimal data loss however this is a trade-over between lower execution time or less accurate trading decision set.

The use of a different execution system is the main ideal behind solving the trade-off, exploiting the characteristics of current state-of-the-art algorithms to the advantage of many-core systems while combining different time series representation. The Central Processing Unit (CPU) was for a while the main system used to execute algorithms with heavy workload however, with the increasing demand in computational resources, an alternative system had to be found. The NVIDIA's Graphical Processing Unit (GPU) as it is known did not appear until 2006 but researchers were already using them to accelerate highly parallel algorithms that resembled to graphical programs. The architecture of the GPU presents itself as an excellent alternative execution system that not only was meant to process high volume of information but also the access to a high-level Application Programming Interface (API) that allows a great manipulation of the GPU, the Compute Unified Device Architecture (CUDA) [1].

In the following section, the related work to pattern recognition and matching will be discussed. In section III, the sequential implementation of the SAX/GA algorithm is presented and explained in detail. A benchmark analysis of SAX/GA is performed in section IV. The proposed work to accelerate the SAX/GA algorithm in a GPU is presented in section V. The results for the implemented solutions is discussed in section VI and finally section VII concludes the work.

II. RELATED WORK

Pattern recognition, matching or discovery are terms associated with the identification of specific sequences that are either known *a prior* (recognition or matching) or are found and represent an important mark (discovery). The related techniques (Table 1) to pattern matching can be divided into three categories: Linear approximation, Aggregate approximation and Identification of relevant points. Although focus will be in pattern matching techniques applied to financial time series, these techniques proved to be very versatile and expandable to different areas, going from the medical sector with applications in Electrocardiogram (ECG) [2] to the energy sector with forecasting and modelling of buildings energetic profile [3].

A. Linear Approximation

In the first category, the Middle curve Piecewise Linear Approximation (MPLA) technique [4] tries to evade a problem associated with the simple Piecewise Linear Approximation (PLA), the late representation of a trend inversion. The traditional PLA implementation uses N equal sized linear segments to create an accurate approximation of a time series using the fewest segments possible. However, a trend inversion may occur in one segment being undetected and possibly affecting the representation of future segments. MPLA tries to find inversion points based on the amplitude of three consecutive points, q_{i-1}, q_i and q_{i+1}, and once an inversion point is located (p_j), it is saved to be used in the future to create a so call middle curve of the time series. The middle curve is then passed as input in a different PLA technique, the Divisive Piecewise Linear Approximation (DPLA), that minimizes the distance between a segment, $L(p_j : p_{j+1})$, and the original time series with a threshold constraint ε. If at any point the distance surpasses ε, the segment L is split into two possibly unequal sub-segments, $L(p_j : q_k)$ and $L(q_k : p_{j+1})$. The authors of [4] evaluated the MPLA techniques in an unknown market and period of time. A sub-sequence Q of length 50 was selected from a time series P and the MPLA approach not only could search the input pattern

TABLE 1 – IMPLEMENTATIONS OF PATTERN RECOGNITION TECHNIQUES IN THE LITERATURE

Ref.	Year	Technique	Architecture	Financial Market	Dataset	Performance
[9]	2007	PIP Template-based	CPU	Hang Seng Index	2532 points	≈ 96% accuracy
[9]	2007	PIP Rule-based	CPU	Hang Seng Index	2532 points	≈ 38% accuracy
[14]	2010	PIP-SOM	CPU	N/A	N/A	97.1% accuracy in HS pattern
[4]	2011	MPLA-DDTW	CPU	N/A	2000 points	2× lower error than PAA
[11]	2012	Shapelets	GPU	N/A	N/A	≈ 15× speedup
[15]	2013	PIP-GA	CPU	Taiwan Cap. Weighted Stock Index	N/A	Correct identification of 4 patterns
[8]	2013	SAX-GA	CPU	S&P500	January 2005 – April 2010	62.76% average return (ROI)

Q but also discover identical sub-sequences in *P*. A setback of MPLA is the increase in time complexity comparatively to other techniques.

B. *Aggregate Approximation*

This category groups techniques that, to some extent, divided a time series into equal sized frames, just like in the Linear approximation, and calculate the average of the values in each frame and uses it as the representative value of that portion of the time series. This is the idea behind the Piecewise Aggregate Approximation (PAA) that creates an approximated representation of a time series whose accuracy is directly linked with the number of frames used. To perform any type of comparison between two time series, these must be normalized (Eq. 1) to a common baseline allowing a direct comparison.

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{1}$$

where x_i represents a point, μ is the mean and σ is the standard deviation of the time series.

The time series is then divided into *N frames* of size *W* and the mean value ($\bar{p}_i$) inside that frame is used as the approximation of that portion (Eq. 2). Although PAA adopts frames with fixed size, it is possible applied this method to uneven frames where the border element of two frames split his value's contribution between the neighbor windows [5].

$$\bar{p}_j = \frac{W}{N} \sum_{i=\frac{N}{W}\cdot(j-1)+1}^{\frac{N}{W}\cdot j} x_i' \tag{2}$$

In order to discover similarities in time series, PAA uses a Euclidean distance (ED) based formula where, instead of the point-to -point calculation, it uses the mean value of the reduced series (Eq. 3).

$$Distance\,(P,Q) = \sqrt{\frac{N}{W}} \cdot \sqrt{\sum_{i=1}^{W} (\bar{p}_i - \bar{q}_i)} \tag{3}$$

Although PAA enables a direct comparison between two series, it does not indicate whether the distance obtained represent a low or high level of similarity.

The Symbolic Aggregate approXimation (SAX) [6] can be viewed as an improvement to PAA as it still uses this method to obtain a dimensional reduced time series but adds a new type of data transformation, numeric to symbolic representation. This transformation relies in a Normal distribution, $N \sim (0, 1)$, with α_N intervals where the probability of each interval is equivalent to the difference between the z-score (β_i and β_{i+1}) of both intervals must be equal to $\frac{1}{\alpha_N}$ and to each interval a symbol is assigned. For example, with $\alpha_N = 3$ there are 3 intervals, all with probability of 33.3% and with a symbolic representation,

$$\alpha = \text{'A'} \quad \text{iif} \quad -\infty < c_i < \beta_1$$
$$\alpha = \text{'B'} \quad \text{iif} \quad \beta_1 < c_i < \beta_2$$
$$\alpha = \text{'C'} \quad \text{iif} \quad \beta_2 < c_i < \infty$$

In Figure 1, frame 3 (c_3) has an average value of 1.5 and considering an alphabet with 3 letters (α = 3), it is possible to assess that, through Table 1, c_3 is between β_2 and ∞ and therefore, the corresponding symbol is '*C*'.

This method ensures that, in a fixed size alphabet, each SAX symbol has equal probability allowing a direct comparison and indication of the similarity level.

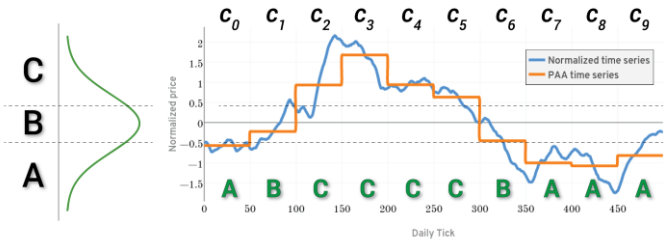

FIGURE 1- TRANSFORMATION OF A PAA SERIES INTO A SAX REPRESENTATION WITH 3 SYMBOLS.

TABLE 2 - Z-SCORES OF NORMAL (0,1) FOR A = [2,8]

β \ α	2	3	4	5	6	7	8
1	0	-0.43	-0.67	-0.84	-0.96	-1.06	-1.15
2		0.43	0	-0.25	-0.43	-0.56	-0.67
3			0.67	0.25	0	-0.18	-0.31
4				0.84	0.43	0.18	0
5					0.96	0.53	0.31
6						1.06	0.67
7							1.15

With the SAX representation is possible to assess if two time series are identical but understanding the level of similarity requires a formula capable of translating the difference between two sequences into a distance value. SAX adapted the distance measure used in the PAA method to now take in consideration the z-scores used to transform a numeric to a symbolic (Eq. 4).

$$MINDIST(\hat{P},\hat{Q}) = \sqrt{\frac{N}{W}} \cdot \sqrt{\sum_{i=0}^{W} (dist(\hat{p}_i,\hat{q}_i))^2} \tag{4}$$

The main difference between of the distance measure (3) and (4) lies in the *dist* function that calculates the distance of symbol P_i and Q_i (Eq.5).

$$dist(\hat{p}_i, \hat{q}_i) = \begin{cases} 0, & |i-j| \leq 1 \\ \beta_{j-1} - \beta_i, & i < j-1 \\ \beta_{i-1} - \beta_j, & i > j+1 \end{cases} \quad (5)$$

The SAX symbolic representation can produce a very compact and efficient time series however it is subject to a particular problem, mainly caused by PAA. Since the symbolic representation of each window is calculated using the mean value of the series in that window, it cannot accurately represent a trend as important points will be ignored. An alternative solution, Extended SAX (eSAX) [7], can be used to fix this issue. Instead of only considering the mean value of the frame, two additional points are added, the maximum and minimum values. This value composed a string of ordered triplets, $< v_{min}, v_{avg}, v_{max} >$, that can help understand the behaviour of the time series inside each frame.

An application of SAX representation combined with a Genetic Algorithm approach applied to investment strategies based on pattern discovery was proposed by [8]. The authors present a multi-chromosome GA with individuals divided into 2 categories, the parameters that support buy or sell decisions and the SAX patterns used to identify similarities to the company stock series.

C. Identification of important points

The MPLA technique tried to find inversion points in the time series to build an approximate representation of the information. A similar approach is the Perceptually Important Points (PIP) and, as the name suggests, the PIP method searches for points that are humanly identifiable.

The method starts with two fixed PIPs, the first (p_1) and last points (p_2) of the time series. The next PIP can be obtained by maximizing the distance of the lines that unites two consecutive PIPs to a point of the series. For instance, $p_3$is the result of maximizing the accumulative distance between the segments $\overline{p_1 : p_{test}}$ and $\overline{p_{test} : p_2}$. This is an iterative process so that for each PIP, two more are generated, meaning that there is not a defined stopping condition with the possibility of having $len(dataset) - 1$ PIPs.

The PIP approach can take advantage of two types of pattern recognition, template- or rule-based matching. The template matching requires that the user defines patterns meant to be matched against the series. The template comparison of two PIP series adopted the ED of minimizing the point-to-point distance and two different measures are necessary, vertical and horizontal, so that series that are distorted either in frequency or amplitude can be compared. Based in experimental results, [9] suggests that both vertical and horizontal distance have equal contribution to the final template distance.

Using the template approach can be a tedious process since the user needs to define the input patterns and the results obtained are directly linked to the accuracy of these. To overcome this issue, a different approach can be taken through the definition of a set of rules to the PIP values allowing a broader number of possible solutions

The Turning Points (TP) approach (Figure 4) follows an identical idea to MPLA since both methods maximum and minimum values however the TP technique applies a filter where points that have a low contribution to the overall shape of the time series are suppressed. The filter is defined by 4 cases.

D. Shapelets

The previous techniques focused in matching an input pattern to a time series. Reference [10] introduces a supervised approach that instead of searching for specific patterns in a time series, the algorithm searches for the best pattern or pivot that is capable of characterizing a series class, a group of identical series T_i, e.g., species of leaves as used in [10], with a unique identifier. The objective is to divide the original dataset *D* into two sub-groups, D_L and D_R, based on a threshold distance of the pivots (θ),

$$Distance(Pivot_j, T_i) \leq \theta, \forall T_i \in D_L$$
$$Distance(Pivot_j, T_i) > \theta, \forall T_i \in D_R \quad (6)$$

The pivot *(j,i)* of $T_{i,C}$ that ends up on maximizing the distance between elements of different classes and θ, will be used as the shapelet that "identifies" class *C*.

Until now, all techniques present used a very familiar system that integrates the Central Processing Unit (CPU) as the responsible for the execution of the algorithms. Nevertheless, Reference [11] introduces the Graphical Processing Unit (GPU) to accelerate the Shapelets algorithm where the computation of the distance between a pivot and T_i is performed one thread.

Several methods were presented with different approaches to achieve one common objective, a high fidelity algorithm for pattern matching or discovery. Many of them are based in a tedious iterative process of matching either a known pattern or discovering one. The Shapelets algorithm took advantage of the GPU to accelerate the process of locating the best pivot parallelizing the iteration of pivots. Looking to the overall structure of each technique, there is one that stands out for the rest, the SAX method. SAX relies in "compressing" possibly equal sized frames in an independent process and allows the observation of two sequences through a distance measure that guarantees a fair comparison. Combining the SAX technique of pattern recognition with the GA's ability of optimizing a group of patterns and a powerful and highly parallel technique is created.

III. SAX/GA CPU Approach

The SAX/GA algorithm [8,12] was developed with the purpose of optimizing a trading strategy with multiple patterns for entry and exit in the Standard & Poor 500 (S&P500) index either in a long or short position. A long position is associated with the expectation of an increase in the appreciation of a stock while the short position expects a decrease in the value of a stock.

A. Training Approach

The algorithm uses daily historical data extracted from the S&P500 index which is then divided in a sliding window fashion (Figure 5).

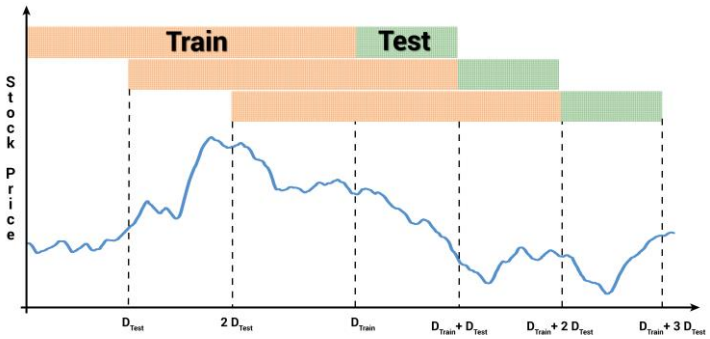

FIGURE 2 - SLIDING WINDOW METHOD.

The initial training dataset, D_{train}, is succeeded by a test dataset, D_{test}, that is used to validate the best strategy obtained during the training phase. Once testing is completed, D_{train} is shifted so that the end of the next D_{train} matches the end of the previous D_{test} and the optimization process restarts until the dataset reaches the end or a stopping criteria is met.

B. Population Generation

The optimization of the trading strategy starts with the generation of a random population of chromosomes with size *NumChromo*. The chromosome structure (Figure 6) is divided in four categories, each representing a type of trading rule. These rules are market entry with a long positon, leave market in a long position, enter with short position and exit in short. Each rule consists in one or more genes and each gene is comprised in a SAX sequence and the characterization parameters (*Window Size*, *Word Size* and *Alpha Size*). In order to facilitate the crossover operator, all four categories are created with a *NumPattern* genes but are internally limited to Enter Long (*IL*), Enter Short (*IL*), Exit Long (*OL*) and Exit Short (*OS*) genes.

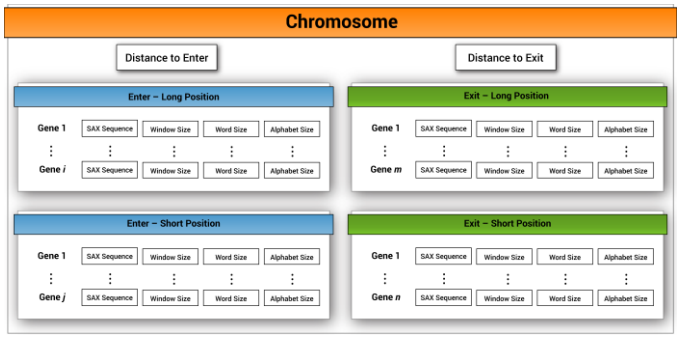

FIGURE 3 - CHROMOSOME STRUCTURE.

Two additional values are introduced, D_{enter} and D_{exit}, both setting the minimum distance allowed between an individual's SAX pattern and the converted dataset SAX sequence to either enter or exit the market respectively. A key aspect of SAX/GA is that it uses the same population for all training windows which may be beneficial for the overall convergence of the algorithm since the next pair has already been trained with a large part of the dataset.

C. Fitness Evaluation

The fitness function iterates the training dataset following a Finite-state machine (FSM) with four possible states (Figure 7),

- **State 1 (S1)** – The individual is in the market and is now looking to close his position, either through a long or short exit. If a pattern of either category is found, the next sate will be *S4* otherwise remains in *S1*.
- **State 2 (S2) –** The chromosome triggered an entry with long position in the previous state and must now process the current balance and ROI. This sets the next state to *S1*.
- **State 3 (S3) –** In the previous iteration, the algorithm successfully found a pattern to enter in short and must now calculate the current balance and ROI. Sets next state to *S1*.
- **State 4 (S4) –** The chromosome is out of the market and is searching for a pattern to enter in long or short. Similar to *S1*, if a pattern is found then the next state is either *S2* or *S3*, subject to the category of the pattern found. Additionally, if the previous state was *S1*, then it is necessary to process the balance and ROI values.

To switch from SAX state (*S1* and *S4*), the distance between the gene's SAX sequence and the converted pattern must be smaller than the gene's encoded distance however there are specific cases where a looser rule is applied. Considering Table 1, with larger alphabets the distance between two consecutive symbols is relative small and, for example, with $D_{enter} = 0.273$ and a pool of distance, $P = [0.564; 0.436; 1.076]$, the SAX/GA would not indicate that a pattern was found. Nonetheless, all distances are relative close and, in large alphabets, only with a difference of one or two symbols and therefore, the state transition is triggered by the product or the difference of all distances.

The fitness score of each chromosome is based on the ideal trading strategy and the trading solution achieved by the chromosome. The ideal strategy corresponds to a long position during an up rise of the stock prince and a short position while there is a decrease in value (Eq. 7).

$$E_i = E_{i-1} - |P_i - P_{i-1}| \tag{7}$$

where E_i is the accumulated profit at day *i* and P_i is the stock price in day *i*.

As for the chromosome strategy, it is evaluated identically to (7) but instead, it takes in consideration whether, at day *i*, the chromosome is in a short or long position (Eq. 8).

$$E_i^c = \begin{cases} E_{i-1}^c, & if\ Out\ of\ market \\ E_{i-1}^c - (P_{i-1} - P_i), & if\ Short\ position \\ E_{i-1}^c - (P_i - P_{i-1}), & if\ Long\ position \end{cases} \tag{8}$$

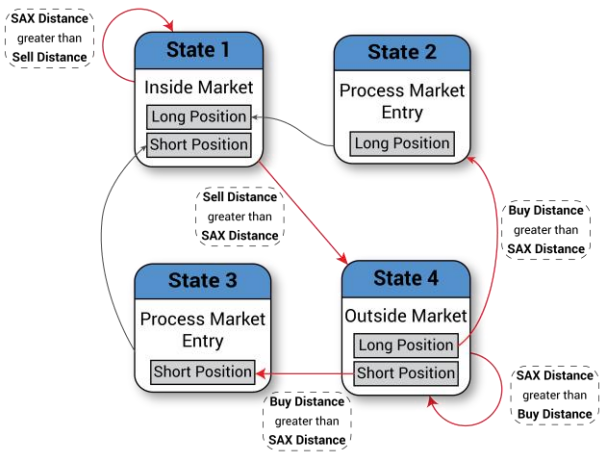

FIGURE 4 - REPRESENTATION OF THE FITNESS FSM WORKFLOW.

Finally, the chromosome fitness (Eq. 9) is based on the sum of the difference between (7) and (8) where a smaller score is better. Furthermore, the chromosome must enter and exit the market at least once in each position avoiding rare cases of pattern matching and must have a positive ROI.

$$Fitness\ score = \begin{cases} \dfrac{\sum_{i=0}^{n} |E_i - E_i^c|}{(1 + ROI)}, & entry \cdot exit > 1, ROI > 0 \\ 3.40282E + 38, & entry \cdot exit = 0, ROI \leq 0 \end{cases} \tag{9}$$

D. Population Selection

SAX/GA uses a uniform ranking selection that essentially sorts the population based on the fitness score preserving the best half and introducing elitism in the GA. Uniform ranking selection has proven to reach good results in [8], [12] and in [13] where ranking based implementations converged into higher quality solutions in fewer generations comparatively to tournament selections and others.

E. Population Crossover

The SAX/GA crossover uses a slightly different approach of the conventional operator where instead of splitting the chromosome in *N* points and alternately transfer genes from both parents to the offspring, the implemented method performs the crossover at an inner gene level, the SAX sequence, in an attempt of more diversified solutions and a broader search space.

Furthermore, since SAX/GA uses a dynamic system of genes added care is needed. Each chromosome is generated with a fixed number of genes however it is only allowed to use *IL*, *IS*, *OL* and *OL* genes meaning that in a pair of parents where these values are not equal, at least one or more genes would not be selected and transferred to an offspring. In those cases, the extra unused genes come in as all genes would be paired up and able to crossover even though they might not be utilized. Figure 8 demonstrates a crossover example.

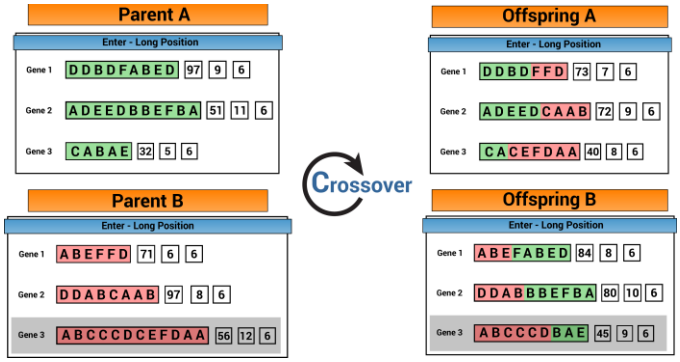

FIGURE 5 - CROSSOVER EXAMPLE BETWEEN PARENT A WITH 3 IL AND B WITH 2 IL GENES.

F. Chromosome Mutation

The mutation operator is only applied to a relative small sample of the population and besides that, in SAX/GA, it is far more aggressive than what is the conventional mutation process. Conventionally, the mutation introduces modifications to the genetic information which would be the SAX sequence and characterization parameters. However, since changing a symbol in a SAX sequence has a limited impact in the distance, the SAX/GA mutation creates or removes complete genes through the modification of the internal limits (IL, IS, OL and OS), increasing or decreasing them.

IV. SAX/GA CPU PERFORMANCE ANALYSIS

To understand whether SAX/GA can take advantage of a GPU architecture, the algorithm was subject to a benchmark analysis to assess the execution time and locate areas with heavy computational work under different conditions. The platform specifications are detailed in Table 2 and the benchmark parameters in Table 3. The SAX/GA sequential program was compiled using *gcc* compiler with *C++11* standard and optimization flags set to *-O3*.

TABLE 3 - BENCHMARK PLATFORM SPECIFICATIONS

	Specifications
CPU	i7 6700 @ 3.4 GHz
RAM DDR4 (GB)	8
OS	Ubuntu 14.04.2 kernel 4.2.0-42
GCC Version	4.8.4
GPU	GTX 760 – Kepler GK 104
CUDA SDK Version	7.5

These parameters were selected so that virtually all operators would suffer changes in the workload. The number of chromosomes is directly linked with all operators while the training size dictates the number of cycles of the fitness FSM.

Table 4 shows the average results for a 5 runs benchmark test. Increasing the population in a ratio of two has a proportional impact in all genetic operator although, in the crossover and mutation, it only becomes visible once the number of individuals reaches 256.

TABLE 4- SAX/GA BENCHMARK PARAMETERS

	Parameters
Population Size	64; 128 256; 512
Training Size	128; 256; 512
Test size	60
Generations	200
Window Size	15 → 100
Word Size	4 → 16
Alphabet size	6
Genes per chromosome	1 → 3
Mutation rate	10%
Historical dataset size	3094

For an equal number of chromosomes, a larger training dataset also doubles the execution time of the fitness function however, since the training dataset dictates the number of training windows, increasing the dataset represents fewer windows and therefore, less total generations decreasing the execution time for the crossover and mutation.

A straightforward conclusion and an expected one is that the fitness operator is the main source of heavy computational work being responsible for 93.9% up to 99.6% of the execution time. So, under the same conditions, an additional test was performed with the objective of pinpointing what is causing the bottleneck.

TABLE 5 - EXECUTION TIME OF THE SAX/GA ALGORITHM AND THE GENETIC OPERATORS (TIME IN SECONDS).

Training Size	**Genetic Operator**	**Population size** 64	128	256	512
128	Fitness	398.92	780.21	1621.74	3231.24
	Crossover	14.26	18.83	28.94	51.68
	Mutation	0.035	0.058	0.11	0.19
	Total	412.29	799.17	1650.88	3283.19
256	Fitness	822.06	1590.17	3276.14	6467.38
	Crossover	14.01	18.09	27.85	50.11
	Mutation	0.031	0.05	0.10	0.18
	Total	836.17	1608.39	3304.17	6517.85
512	Fitness	1503.07	3150.40	6313.04	12156.7
	Crossover	12.57	15.51	25.64	45.40
	Mutation	0.026	0.044	0.079	0.13
	Total	1515.735	3167.02	6338.83	12202.32

It became obvious that, the need to convert a time series into a SAX sequence at a rate of once per gene, creates a colossal bottleneck on the fitness operator making up for 99.5% of the execution time across all test cases. Designing an alternative version of the SAX representation method, capable of exploiting the GPU architecture is the logical path to follow.

V. GPU-ACCELERATED SAX/GA

The SAX transformation was identified as the main cause of high execution time and, recalling Figure 7, there are two possible scenarios that require a call to the SAX method, S1 and S4, however it is not possible to know when a SAX transformation will be necessary.

A. Speculative GPU SAX transformation

Considering a speculative FSM, SAX/GA predicts that all FSM states require a SAX transformation however there is the possibility of misprediction caused by a market entry or exit. The misprediction rate is heavily minimized since financial datasets

tend to display low variance in conjunction with the fact that SAX/GA uses daily prices and therefore, when a chromosome triggers a transition to *S1* or *S4*, there is a high probability that it will remain in that state for more than one iteration. In the best case, a chromosome never enters the market and the prediction rate is 100% while, in the worst case, the chromosome is constantly changing positions and may reach a 66.6% hit rate depending of the length of the dataset.

The fitness function is now divided into a hybrid architecture where the CPU is responsible for the iteration of the fitness FSM while the GPU performs the SAX transformation for all genes of all FSM states. The CPU execution becomes far more simple only having to access an array that contains all SAX distance when necessary triggering state transition based on the distance rules. To perform a SAX transformation, the GPU will need three types of information, the training dataset, the chromosome's genetic information and the look-up table to translate a PAA value into SAX symbol and the distance between SAX symbols (Table 1).

With a training dataset with length D_{tsize}, for each gene, a portion of the training dataset needs to be converted in a SAX sequence and then compared to the gene's SAX sequence. The transformation of a portion of the dataset is an independent process that can be executed in a thread. Given this, the SAX GPU kernel is configured with a one-dimensional grid of blocks where each block has D_{tsize} threads, each responsible for a SAX transformation. The total amount of blocks varies with the number of genes that are meant to be analyzed in each generation.

Since the *i*th block is assigned with the *i*th gene, it is possible to use the GPU memory primitives in an efficient manner, mainly the shared memory. At the start, thread *j*th of a block loads the *j*th element of training dataset into shared memory and, since the portion of the dataset that thread *j*th is meant to transform starts in element *j*th and ends in *j*th + $gene^{i}_{window\ size}$ of the dataset, a request to a shared memory address is served in a single memory transaction without bank conflicts.

With a highly diverse number of genes per chromosome, it was necessary to come up with a solution that could associate a block's ID with the corresponding gene (Figure 9). Blocks are organized by the gene's category in such way that, all *IL* genes are processed by the first *IL* blocks. The following blocks process the *IS* genes and so forth.

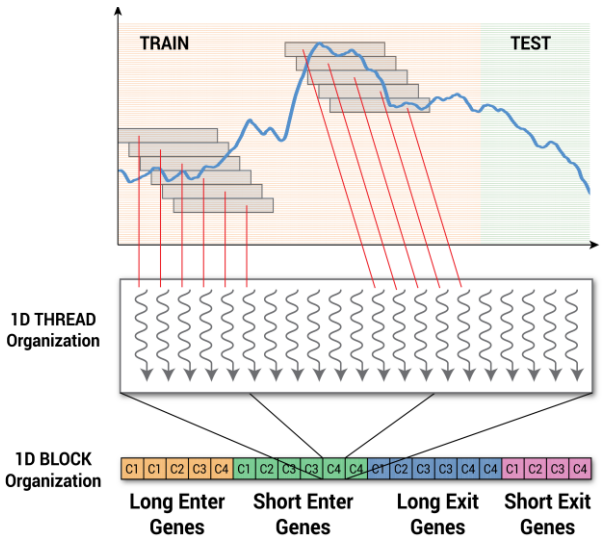

FIGURE 6 - WORK DISTRIBUTION ON THE GPU.

A thread execution path is divided into three different areas, the dataset normalized, PAA and SAX transformation. The data normalization requires the mean and standard deviation of the dataset which is later used in the PAA transformation. To avoid having to store an additional array of values corresponding to the data normalization, Eq. 2 is partially modified so that the PAA transformation is performed at the time of the data normalization reducing memory usage and transactions while decreasing the number of instructions. The SAX transformation is a relative simple operation where, for each PAA value, the thread iterates a look-up table and translates the value into a SAX symbol. Once the *k*th SAX symbol is obtained, the distance between the gene and the converted *k*th symbol is calculated and later saved in global memory.

B. Parallel SAX/GA Training

SAX/GA optimizes investment strategies meant to be applied for extended periods of time and, in order to do so, a sliding window training process is used where the GA trains a population of individuals, validates the best fitted solution and passes the population to the next training window and the optimization process restarts. An advantage of this training method is that the vast majority of chromosomes were already trained for a large portion of the dataset and therefore, in theory, the algorithm convergence should improve greatly with the knowledge transfer between windows.

An attempt to increase the level of parallelism achieved with Solution A and understand whether the continuous process of optimization brings any benefit to the quality of the trading strategy, the training process is now performed in parallel where each training window has its own population of individuals. A disadvantage of this approach is that the genetic operators are still assigned to the CPU, mainly FSM iteration, crossover and mutation, register an increase of workload proportional to the number of training windows. With that, the remaining part of the fitness function still performed by the CPU is transferred to the GPU with the fitness process being a full GPU method now.

For the SAX kernel, the main difference is the kernel configuration. Previously, the kernel took a one-dimensional grid of blocks where, to each block, a gene was associated. The parallel training allows a two-dimensional configuration where the *x*-axis of block stays the same as before while the *y*-axis represents the index of the training window. The implementation used to locate a gene in the GPU's global memory also suffered an expansion to a two-dimensional structure with identical behaviour of the kernel.

The FSM iteration has a simple organization with the training datasets being distributed across a one-dimensional grid of blocks with threads exploring a one-dimensional space (Figure 10). The number of threads in each block depends on the amount of individuals meant to have the fitness score recalculated and there is the possibility that two blocks will have a different number of threads.

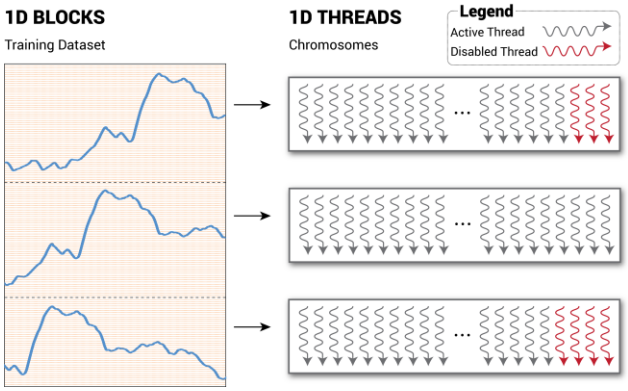

FIGURE 7 - EXAMPLE OF THE FSM KERNEL ORGANIZATION.

The execution flow of a block's threads has unavoidable thread divergence and the kernel needs to rely in predictive flags used to forecast which will be execution path of a diverging branch. Besides the thread divergence, memory transactions are less efficient comparatively to the SAX kernel with coalesced read and writes given the unpredictable course of the FSM.

C. Fully GPU-accelerated SAX/GA

The previous solution took advantage of the increase in the overall workload of the GA operators and transferred what remained of the CPU fitness execution into the GPU leaving the crossover and mutation untouched. The FSM showed several problems associated with the implementation of SAX/GA fitness function and how unpredictable algorithms can complicate the execution in a GPU causing divergence and inefficient usage of memory resources.

Although the fitness function was identified as the heavy duty task with the highest execution time of all genetic operators, the inflation of workload increased the already existing pressure in the crossover and mutation operators along with the population creation that were still being executed in the CPU. Furthermore, the memory allocations and transactions constantly being performed in each iteration of the GA have reached such size that, the transfer time now represents a larger portion of the fitness execution time. The transfer issues could be mitigated if all genetic operators worked over one common memory space instead of having to constantly transfer from CPU to GPU and vice-versa. This requires three new GPU implementations for the population creation, crossover and mutation of individuals while incorporating the existing kernels.

C.1 Population Generation Kernel

A chromosome is divided into four categories each representing a trading solution to buy or sell shared with different strategies, short or long. To trade shares, the algorithm tries to match the historical stock prices with randomly generated patterns encoded inside each gene. The generation of a gene is an independent process with an identical execution flow for all genes and easily transferred to the GPU. The main concern is associated with possible thread divergence since if each thread is responsible by the computation of one gene, it is practically impossible to avoid divergence as the length of each gene is randomly generated.

The kernel (Figure 11) does not take a special configuration only that a two-dimensional block organization is beneficial exploiting an identical approach of the previous solution. The *y*-axis is used as an indicative of the training dataset while in the blocks of the *x*-axis, threads generate the necessary genes. The amount of blocks and the number of threads in those only needs to be enough to generate all genes. Taking this in consideration, the block organization is calculated so that it has enough threads to help mask memory latency, it has a power-of-two threads and there is an even distribution of blocks in the SMs.

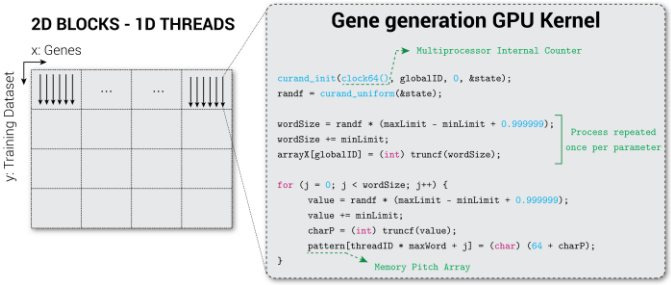

FIGURE 8 - EXECUTION FLOW OF SAX/GA POPULATION GENERATION KERNEL.

C.2 Crossover Kernel

In terms of parallel execution in a GPU, the crossover has the highest level of divergence among all genetic operators which can lead to significant impact in the kernel performance. The crossover performs an exchange of genetic information at the SAX sequence level that will later require the re-calculation of the characterization parameters. The crossover point is calculated based on the length of the SAX sequence and it is used a single-point crossover. The best ranked individuals are selected and used to generated two offspring that will take the position of the worst chromosomes. This process can be executed in parallel where each crossover is performed by a thread.

As for the kernel configuration, the threads are organized in a two-dimensional structure so that the *y*-axis represents the genes of two parents and the *x*-axis is used to associated gene x_i of parent A and B. Blocks follows a two-dimensional organization, identical to previous kernels, with the training datasets being distributed across the *y*-axis while the amount of *x*-axis blocks depends on the number of chromosomes and genes processed.

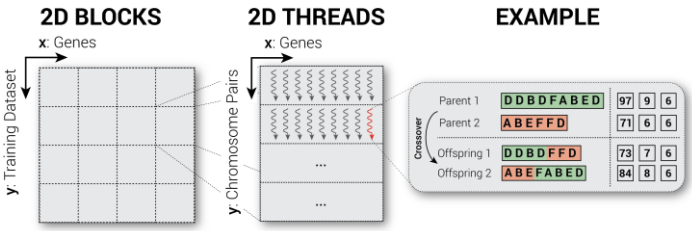

FIGURE 9 - KERNEL CONFIGURATION FOR THE CROSSOVER OPERATOR WITH AN EXECUTION EXAMPLE OF A THREAD.

C.3 Mutation Kernel

The SAX/GA mutation operator is an aggressive approach to introduce diversity in the population of solutions that is applied that the gene level. It selects four genes, one for each category, that can either be regenerated or deleted to introduce or remove new solutions respectively. If the selected gene's ID is higher than the internal limit of the category, the limit is increased and the SAX sequence of said gene is regenerated with random characterization parameters, and when the ID is lower than the limit, the gene is regenerated, in a *quasi*-elimination process, and the limit is decreased.

The GPU kernel of the mutation operator is an adaptation of the population generation kernel with the additional random selection of the gene. The selected chromosomes are based in the mutation rate and the total amount of genes of the chosen individuals in an attempt to obtain a power-of-two genes per training dataset.

VI. EXPERIMENTAL RESULTS

The two metrics used to evaluate each solution are the speedup (Eq. 11) and ROI indicator (Eq.12) which validates the quality of the trading strategy.

$$Speedup = \frac{SAX/GA\ CPU\ Exec.Time}{SAX/GA\ CPU + GPU\ Exec.Time} \quad (11)$$

$$ROI = \frac{P(t) - P(0)}{P(0)} = \frac{P(t)}{P(0)} - 1 \quad (12)$$

To each solution, the benchmark described in Table 2 was applied with an additional population size (1024) and training size (1024). The benchmark test values were selected based on two aspects, first they should be representative of common

investment periods and secondly, they need to be a multiple of 32.

A. *Execution Time*

For testing purposes, the benchmark applied to each solution was executed 10 times to obtain statistically valid results. All solutions were compiled with *gcc* compiler using *C++11* standard. The optimization flags were set to *-O3* and the fast math library was used, benefiting from faster implementations of the division and square-root operations which are fundamental to the SAX transformation.

A.1 SAX/GA with Speculative FSM

Applying the benchmark to this solution, it became evident that computing the SAX transformation and distance in the GPU brings great benefits to the overall execution time of the SAX/GA.

TABLE 6- EXECUTION TIME (IN MILLISECONDS) OF THE SAX KERNEL AND THE FITNESS FUNCTION PER GENERATION.

Operator	Training Size	Population Size				
		64	128	256	512	1024
SAX Kernel	128	0.15	0.20	0.32	0.56	1.05
	256	0.22	0.35	0.59	1.03	1.95
	512	0.45	0.69	1.27	2.41	5.21
	1024	0.94	1.83	3.77	7.91	12.68
Fitness	128	0.80	1.08	1.41	2.05	3.45
	256	1.01	1.43	1.99	3.16	5.49
	512	1.42	2.15	3.43	6.46	12.15
	1024	2.36	3.91	7.32	15.29	25.45

Observing the results in Table 5 and recalling that the SAX representation was responsible for 99.5% of the total execution time, this solution no longer faces this absurd disparity between the total time of the fitness function and that spent in transforming SAX sequences. If two kernel calls were selected with identical number of blocks and different amount of threads, the ratio of the execution time between those two should be proportional to the thread ratio, however this is not observed in Table 5 so this must indicate that, for populations with the same size, the workload of the GPU varies with the length of the training dataset and that the SAX/GA algorithm converges into solutions with different configurations, In Figure 13 it is possible to observe this effect with the increase of the training dataset size.

Identical results were observed when fixing the training dataset size and varying the number of individuals in the population. The speedups achieved (Table 6) proved that the implemented solution is capable of taking advantage of the GPU with values that range from 34× and up to almost 160×.

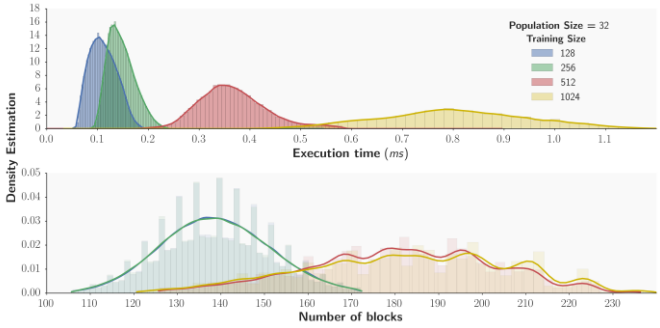

FIGURE 10 - DISTRIBUTION OF THE SAX KERNEL EXECUTION TIME AND WORKLOAD ALONG 4 DIFFERENT TRAINING DATASETS.

TABLE 7 - SPEEDUP OF SOLUTION A RELATIVELY TO THE SEQUENTIAL IMPLEMENTATION AND THE SM OCCUPANCY.

Training Size	Population Size				
	64	128	256	512	1024
128	34.05 (76.7%)	41.04 (84.6%)	46.54 (91.5%)	49.74 (94.9%)	50.12 (96.5%)
256	60.38 (86.9%)	69.66 (93.2%)	87.42 (94.9%)	88.23 (96.4%)	87.376 (96.9%)
512	98.54 (95.3%)	115.99 (96.4%)	137.17 (97.1%)	134.09 (97.4%)	134.77 (97.5%)
1024	133.80 (98.0%)	148.42 (98.1%)	157.49 (98.3%)	153.52 (98.4%)	139.19 (98.5%)

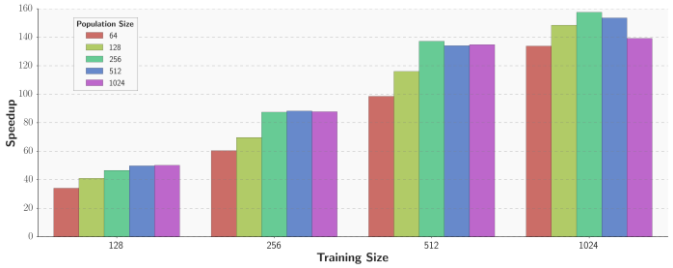

FIGURE 11 - SPEEDUP OF SOLUTION A.

A.2 Parallel SAX/GA Training

The parallel training was introduced as an attempt to increase the level of parallelism of the previous solution and to understand if the continuous optimization of a population does bring benefits to the quality of solutions.

The expected increase of workload in all genetic operators showed significant impact in the fitness function where, the time spent with control operations such memory allocations and transactions and generation of look-up tables, rinsed to 22% of the fitness function execution time. The crossover expresses an identical behaviour where the ratio between the time of the crossover and fitness operators reaches almost double for larger populations and training datasets (Figure 15).

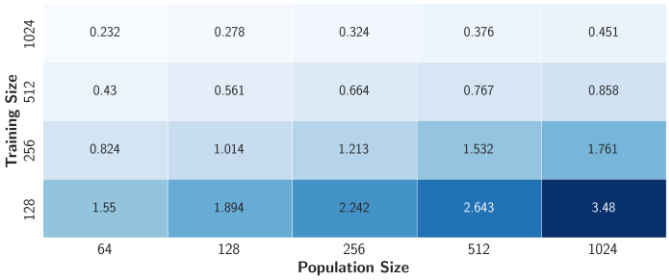

FIGURE 12 - VARIATION OF THE RATIO BETWEEN THE EXECUTION TIME OF THE CROSSOVER AND FITNESS OPERATORS.

Even with such impact of the crossover operators in the SAX/GA algorithm, the parallel training solution was capable of improving the speedup of solution A for smaller populations and training datasets while displayed a moderate loss in the larger test cases. Figure 16 presents the speedup of solution B.

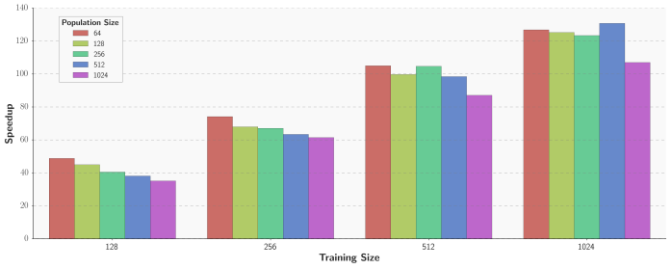

FIGURE 13 - SPEEDUP OF SOLUTION B.

Comparing the speedup differential between solution A and B (Figure 17), solution B shows better speedups smaller populations however with the increase of this, the crossover operator introduces a negative impact in the algorithm and forces the speedup differential to decrease.

The fundamental idea that solution B was based on, the parallel dataset training, ended up in introducing limitations to the SAX/GA algorithm.

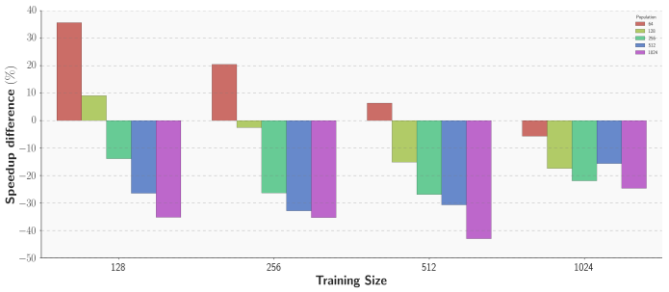

FIGURE 14 - DIFFERENCE BETWEEN THE SPEEDUP OF SOLUTION B AND A.

The fitness control operations forced the CPU to perform a task that saw its load greatly along with a very impactful crossover operator that reduces the achievable speedup gain versus solution A.

A.3 Fully GPU-accelerated SAX/GA

Solution C was developed with the purpose of solving the issues introduced with the parallel dataset training and the increase of workload. The most noticeable modification added in comparison with solution B is the switch of a "shared" memory space between the CPU and GPU to a common space that is the GPU global memory. This approach reduced the time spent with auxiliary operations such as memory allocations and transaction from 22% to 2.85% of the fitness function time on average.

The newest addition on the GPU side is the crossover operator. The high ratio between crossover and fitness in solution B reinforces the need to transfer the crossover operator to GPU which did improve the crossover/fitness ratio as displayed in Figure 18. Even with a decrease in the execution of the fitness function due to the reduction in auxiliary operations, the crossover now has half of the impact it had on solution B and therefore, solution C should clearly outperform B but, against solution A, it is difficult to take a concrete conclusion.

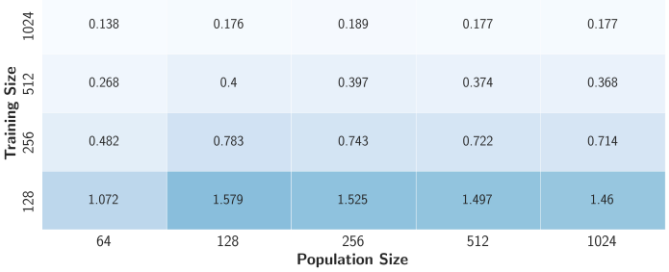

FIGURE 15 - VARIATION OF THE RATIO BETWEEN THE CROSSOVER AND FITNESS OPERATORS EXECUTION TIME.

The speedup of solution C (Figure 19) presents an identical behaviour to solution A where there is a gradual increase of the speedup with larger populations although, for the same population size, A keeps improving due to a higher SM occupancy while C has already reached the theoretical limit and maintains a steady speedup with the increase of training days.

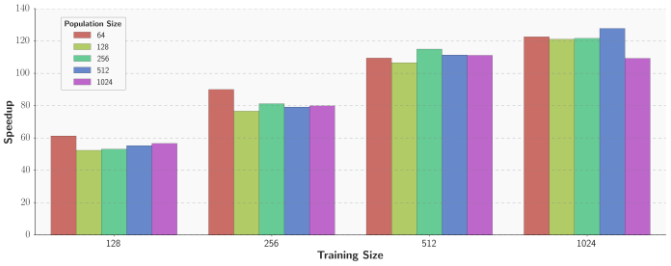

FIGURE 16 - SPEEDUP OF SOLUTION C.

As for the speedup differential between solution A and C (Figure 20), C was capable of reducing the differential that was observed with solution B although there are still test cases that have a negative difference however with a reduced impact.

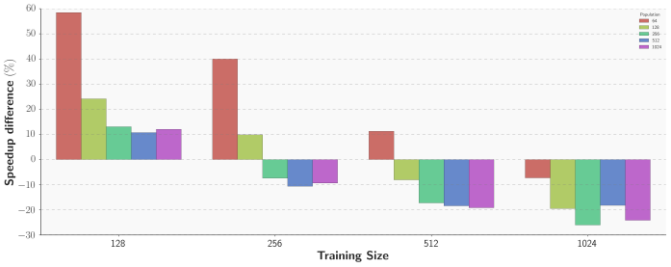

FIGURE 17 - DIFFERENCE BETWEEN SPEEDUP OF SOLUTION C AND A.

The most noticeable improvements in solution C are with training sizes of 128 where all populations where executed in a shorter period of time in comparison with solution A and with the maximum differential where, for positive differences, there is an increase of 20%, on average, while for negative differences a decrease of also 20% is observed.

B. Quality of Solutions

Until now, the results of the developed solutions show that using a GPU to accelerate the SAX/GA algorithm can bring excellent improvements to the performance of the algorithm. However, these results are only beneficial if the solutions optimized by each implementation are identical to the original version of SAX/GA with the main difference being the type of optimization used.

The benchmark applied to all solutions used historical data from the S&P500 index, more specifically the Air Products and Chemicals, Inc. company, and from a period of time that includes a steady increase in the appreciation of the company that is followed by a rapid decrease in value due to the market crash of 2008. Introducing a period where there is a rapid inversion of trend allows testing the GA behaviour under extreme conditions. Since the objective is to validate the developed implementations and understand what is the effect of the optimization process, solution A is used as the continuous optimization solution and solution C for the parallel training. For each solution, the average and best ROI were calculated.

Considering the average case, SAX/GA was capable of converging into different individuals that are capable of perform trading actions that lead into a positive ROI (Figure 21). On average, the vast majority of test cases obtained a positive ROI and there is a trade-off across all solutions between the best average.

In the ideal situation, the best trading solution is the one selected to be executed in a real market transaction. The best strategy obtained by each solution is influenced by multiple parameters and it is almost guaranteed that is different from one solution to another. Observing Figure 22, solution A was the only one that achieved only positive ROIs even though it is a minor accomplishment. Taking a closer look at the ROI values for each solution, the original CPU achieved slightly better results than solution A with both sharing the same optimization strategy, the continuous optimization, however solution C presents overall higher ROIs comparatively to A even though there is a test case that lead to losses for the user.

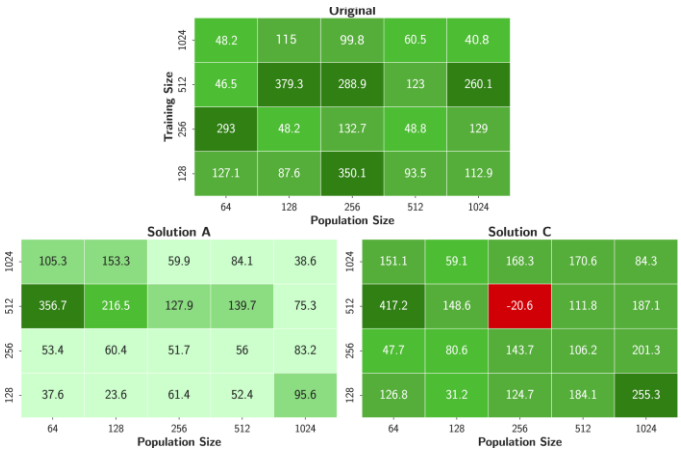

FIGURE 18- BEST ROI (%) FOR ALL TEST CASES**.**

The continuous training process can directly influence the GA into poor optimization due to an overwhelming accumulation of knowledge that is passed through each window and once the algorithm enters in a validation phase completely counters all that has been learnt, such as a market crash, it starts displaying an erratic behaviour and is said to be over-fitted. Using the parallel training of solution C, the GA completely ignores how it behaves in the previous windows focusing only in the current one which, in periods of high volatility, proved to beneficial.

VII. CONCLUSIONS

SAX/GA was initially designed to be an evolutionary algorithm that could explore a vast search space with relative small populations of individuals. To accomplish such implementation, genetic operators modify the information of an individual in far more aggressive manners than conventional GAs however their complex implementation lead to partially inefficient executions in the GPU.

Based in the speedups solely, solution A comes out as the winning implementation while B and C failed to meet the baseline speedup of A. However, B and C were not developed for pure speedup improvements but to study the optimization process. The continuous process presented positive average ROI values that were higher than the parallel optimization but, looking to the best trading strategy, the parallel training obtained higher best ROIs in far more cases than the continuous optimization. The accumulated knowledge converged the population into trading strategies that display good behaviour while there is a well maintained trend but upon reaching an opposite trend such as a market crash, the GA was not capable of adapting into different solutions due to the overpowering of the current ones.

REFERENCES

[1] NVIDIA Corporation, "NVIDIA CUDA – Compute Unified Device Architecture Programming Guide", https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, 2015, accessed: 15-11-2015

[2] W.-S. Chen, L.Hsieh, and S.-Y Yuan, "High performance data compression method with pattern matching for biomedical ECG and arterial pulse waveforms", Computer Methods and Programs in Biomedicine, vol. 74, no. 1, pp. 11 – 27, 2004.

[3] F. Iglesisas and W.Kastner, "Analysis of similarity measures in time series clustering for the discovery of building energy patterns", Energies, vol. 6, no.1 2, p.579, 2013.

[4] H. Li, C. Guo, and W.Qiu, "Similarity measure based on piecewise linear approximation and derivative dynamic time warping for time series mining", Expert Systems with Applications, vol. 38, no. 12, pp.14732-14743, 2011.

[5] L. Wei, "SAX: N/n not equal to an integer case". [Online], http://alumni.csucr.edu/~wli/

[6] L.Wei, E. Keogh, S.Lonardi, and B.Chiu, "A symbolic representation of time series, with implications for streaming algorithms" in Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, ser. DMKD'03. New York, NY, USA: ACM, 2003, pp.2-11.

[7] B. Lkhagva, Y. Suzuki, and K.Kawagoe, "Extended SAX: Extension of Symbolic Aggregate Approximation for Financial Time Series Data Representation, Proceedings of DEWS, 2006.

[8] A. Canelas, R. Neves and N.Horta, "A SAX/GA approach to evolve investment strategies on financial markets based on pattern discovery techniques, Expert Systems with Applications, vol. 40, no.5, pp.1579-1590, 2013.

[9] T.-c. Fu, F.-I. Chung, R.Luk, and C.-m. Ng., "Stock time series pattern matching: Template-based vs. Rule-ased approaches", Engineering Applications of Artificial Intelligence, vol.20, no. 3, pp. 347-364, Apr.2007.

[10] L. Ye and E.Keogh, "Time series shapelets", Proceedings of the 15th ACM SGKDD International Conference on Knowledge discovery and data mining – KDD '09, p.847, 2009.

[11] K.W. Chang, B. Deka, W. M. W. Hwu, and D. Roth, "Efficient pattern-based time series classification on GPU", Proceedings – IEEE International Conference on Data Mining, ICDM, pp. 131-140, 2012.

[12] A. Canelas, R. Neves, and N. Horta, "Multi-dimensional pattern discovery in financial time series using SAX/GA with extended robustness", GECCO, 2013.

[13] B. tak Zhang and J. jib Kim, "Comparison of selection methods for evolutionary optimization", Evolutionary Optimzation, vol. 2, pp.55-70, 2000.

[14] A. Zapranis and P.Tsinaslanidis, "Identification of the head-and-shoulders technical analysis pattern with neural networks," in Artificial Neural Networks – ICANN 2010, ser. Lecture Notes in Computer Science, K.Diamantaras, W.Duch, and L.Iliadis, Eds. Springer Berlin Heidelberg, 2010, vol. 6354, pp. 130-136.

[15] C.-H. Chen, V. Tseng, H.-H. Yu, and T.-P.Hong, "Time series pattern discovery by a pip-based evolutionary approach", Soft Computing, vol. 17, no. 9, pp. 1699-1710, 2013