Expert Systems With Applications 105 (2018) 77–88

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Parallel SAX/GA for financial pattern matching using NVIDIA's GPU

João Baúto, António Canelas, Rui Neves, Nuno Horta[*]

Instituto de Telecomunicações, Instituto Superior Técnico, 1049-001 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 3 December 2017
Revised 21 February 2018
Accepted 14 March 2018
Available online 20 March 2018

Keywords:
Genetic Algorithms
Finance computation
Pattern matching
Symbolic Aggregate ApproXimation
GPU
CUDA

ABSTRACT

This paper starts by presenting a study from a computational performance standpoint of SAX/GA, an algorithm that uses the Symbolic Aggregate approXimation (SAX), to dimensionally reduce time series, and the Genetic Algorithm (GA) to optimise market trading strategies. This study highlights how the sequential implementation of SAX/GA and genetic operators works. 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 single-core SAX/GA solution in about 30 times with a maximum of nearly 180 times.

1. Introduction

The financial system and, particularly, the stock markets represent a huge amount of real and dynamic data resources for researchers allowing the true exploration of new algorithms and methodologies. Technical analysis is a well-known tool used by traders to identify market decision points, i.e., buy and sell opportunities. Pattern recognition plays a key role when identifying market and/or stock trends, however, this is not feasible by hand anymore due to the huge size of financial time series and the large number of available stocks. Therefore, it is clear that creating a custom trading strategy that uses patterns as decision points of entry or exit on the market can be a tedious and long process of optimisation where multiple possible decisions sets must be tested against large time series. Researchers have been trying to ease the optimisation process by reducing datasets while maintaining a precise representation with minimal data loss, however, this is a trade-off between lower execution time or less accurate trading decision set. This paper presents a study from a computational performance point of view on how to accelerate the SAX/GA algorithm, an algorithm designed for financial trading over long periods, which uses the SAX representation to dimensionally reduce time series without significant information loss, providing a valid method for comparing two time series, while the GA optimizes a pool of possible solutions. The main downfall of using GA's or other type of "learning" algorithms is that the optimization process of solutions can be very time consuming even for modern Central Processing Units (CPU). Exploring an alternative execution system, like the Graphical Processing Unit (GPU), is the main ideal behind this paper however such task is not straightforward given the architectural differences between the CPU and GPU.

Section II presents a review of related work where the application of pattern recognition, matching or discovery techniques in the domain of computational finance. Then in Section III, the sequential CPU SAX/GA implementation is described and analysed to identify the execution bottlenecks to be consider in the parallelized solution. Moreover, the sequential SAX/GA implementation is considered for a performance benchmark, which is later used in a baseline comparison with the parallelized implementation, discussed in section IV. Based on the findings of the performance benchmark, Section V presents three solutions to accelerate the SAX/GA algorithm using a NVIDIA GPU. Solution A focuses on the fitness operator and, with that, minor modifications are proposed to the original fitness function, inspired by the out of order and speculative execution on the CPU architecture, to take advantage of the GPU. Further changes were implemented in solution B to understand what is the impact of knowledge transfer between generations which led to a complete parallel fitness process where all generations are processed at once. And finally, solution C is introduced to solve the performance issues directly caused by the parallel fitness process and the memory transfers associated, leading a fully accelerated GPU SAX/GA algorithm. The results obtained with each solutions are discussed in Section VI with particular interests in two metrics, the Return on Investment (ROI) and the speedup achieved when compared to the original algorithm. The ROI met-

[*] Corresponding author.
E-mail addresses: joao.bauto@tecnico.ulisboa.pt (J. Baúto), antonio.canelas@tecnico.ulisboa.pt (A. Canelas), rui.neves@tecnico.ulisboa.pt (R. Neves), nuno.horta@lx.it.pt (N. Horta).

https://doi.org/10.1016/j.eswa.2018.03.026
0957-4174/© 2018 Elsevier Ltd. All rights reserved.

Table 1
Implementations of pattern recognition techniques on the literature.

Refs.	Technique	Architecture	Financial market	Dataset	Performance
Fu et al. (2007)	PIP Template	CPU	HSI	2532 points	~96% accuracy
Fu et al. (2007)	PIP Rule	CPU	HSI	2532 points	~38% accuracy
Zapranis and Tsinaslanidis (2010)	PIP-SOM	CPU	N/A	N/A	97.1% accuracy in HS pattern
Li et al. (2011)	MPLA-DDTW	CPU	N/A	2000 points	2x lower error than PAA
Chang, Deka, Hwu, and Roth (2012)	Shapelets	GPU	N/A	N/A	~15x speedup
Chen, Tseng, Yu, and Hong (2013)	PIP-GA	CPU	TAIEX	N/A	Correct identification of 4 patterns
Canelas et al. (2013b)	SAX-GA	CPU	S&P500	Jan 2005 Apr 2010	67.76% average return (ROI)
Bakhach et al. (2016b)	DC	CPU	Forex	Jan 2015 Jul 2015	67–78% Profitability

ric serves as the financial baseline that verifies that the developed solutions and modifications introduced did not have a negative impact while the speedup is used to create the performance ground truth of the algorithm. Section VII concludes the proposed work with some final remarks regarding the results obtained.

2. Related work

Pattern recognition, matching or discovery are terms associated with the identification of specific sequences that are either known a priori (recognition or matching) or are found and represent an important discovery. Although the focus will be on pattern matching techniques applied to financial time series (Table 1), these techniques proved to be very versatile and expandable to different areas, from the medical sector with applications in Electrocardiogram (ECG) (Chen, Hsieh, & Yuan, 2004) to the energy sector with forecasting and modelling of building energy profiles (Iglesias & Kastner, 2013).

The Middle curve Piecewise Linear Approximation (MPLA) approach looks to evade a problem associated with the simple Piecewise Linear Approximation (PLA), the delayed representation of a trend inversion, by finding inversion points based on the amplitude of three consecutive points, q_{i-1}, q_i and q_{i+1}. Once an inversion point is located (p_j), it is saved to be used in the future to create a so-called middle curve of the time series which is then passed as input to a different PLA technique, the Divisive Piecewise Linear Approximation (DPLA). Li, Guo, and Qiu (2011) evaluated the MPLA techniques in an unknown market and period of time. A subsequence Q of length 50 was selected from a time series P and the MPLA approach not only could search the input pattern Q but also discover identical sub-sequences in P.

A disadvantage of MPLA is the increase in time complexity comparatively to other techniques such as the SAX. Similarities can be found when comparing MPLA to the Directional Changes (DC) approach (Bakhach, Tsang, & Jalalian, 2016a; Bakhach, Tsang, Ng, & Chinthalapati, 2016b) which, not only searches for extreme points or trend inversion points, but also points that confirm a directional change. The decision on whether a point confirms a change is made based on the absolute percentage change of the price at the previous inversion point and the current point.

Much like the MPLA technique, the Piecewise Aggregate Approximation (PAA) uses an identical principle of dividing a time series into smaller frames, however, instead of a linear representation for each frame, the average of a frame's values is used as the representative value of that portion in the time series. To perform any type of comparison between two time series, these must be normalised (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.

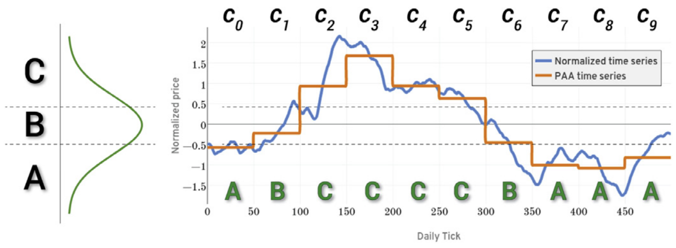


Fig. 1. Transformation of a PAA series into a SAX representation with 3 symbols.

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 contribution between the neighbour windows (Wei, 2006).

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

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 (3).

$$Distance(P, Q) = \sqrt{\frac{n}{w}} \cdot \sqrt{\sum_{i=1}^{w} (\bar{p}_i - \bar{q}_i)^2} \tag{3}$$

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

The Symbolic Aggregate approXimation (SAX) sax can be viewed as an improvement to PAA as it still uses this method to obtain a dimension reduced time series but adds a new type of data transformation, numeric to symbolic representation. This transformation relies on a normal distribution, $N\sim(0, 1)$, with 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, which must be equal to $\frac{1}{\alpha_N}$, and to each interval a symbol is assigned. For example, for 3 intervals, all with probability of 33.3% and with a symbolic representation,

$$\begin{aligned} \alpha &= \text{`A`} \quad \texttt{iif} \quad -\infty < c_i < \beta_1 \\ \alpha &= \text{`B`} \quad \texttt{iif} \quad \beta_1 < c_i < \beta_2 \\ \alpha &= \text{`C`} \quad \texttt{iif} \quad \beta_2 < c_i < +\infty \end{aligned} \tag{4}$$

In Fig. 1, frame 3 (c_3) has an average value of 1.5 and considering an alphabet with 3 letters ($\alpha = 3$), it is possible to assess that, through Table 2, is between c_3 and β_2 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.

Table 2
Z-scores of *Normal*(0, 1) for $\alpha = [2, 8]$.

	α_i						
β_j	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	0.56	0.31
6						1.06	0.67
7							1.15

Table 3
Benchmark platform specifications.

	Specifications
CPU	i7 6700 @ 3.4 GHz
RAM DDR4 (GB)	8
OS	Ubuntu 14.04.4 kernel 4.2.0-42
GCC Version	4.8.4
GPU	GTX 760 - Kepler
CUDA SDK Version	7.5

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 individual	1 → 3
Mutation rate	10%
S&P500 company	APD
Historical dataset size	3094 days
Historical period	Jan/2008 → Apr/2010

With the SAX representation it is possible to assess if two time series are identical but understanding the level of similarity requires an expression 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 into consideration the z–scores used to transform a numeric to a symbolic (5).

$$MINDIST(\hat{P}, \hat{Q}) = \sqrt{\frac{n}{w}} \cdot \sqrt{\sum_{i=1}^{w} \left(dist(\hat{p}_i - \hat{q}_i)\right)^2} \tag{5}$$

The main difference between the distance measure (3) and (4) lies in the *dist*(·) function that calculates the distance of symbol P_i and Q_i (6).

$$dist(\hat{p}_i - \hat{q}_i) = \begin{cases} 0, & |i-j| \leqslant 1 \\ \beta_{j-1} - \beta_i, & i < j-1 \\ \beta_{i-1} - \beta_j & i > j+1 \end{cases} \tag{6}$$

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) (Lkhagva, Suzuki, & Kawagoe, 2006), 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. These values comprise 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 the SAX representation combined with a Genetic Algorithm approach applied to investment strategies based on pattern discovery was proposed by Canelas, Neves, and Horta (2013b). 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.

The Perceptually Important Points (PIP) approach takes advantage of two types of pattern recognition, template- or rule-based matching (Fu, Chung, Luk, & Ng, 2007). The template matching requires that the user defines patterns meant to be matched against the series while rule matching defines range values for the PIP points.

Until now, the presented techniques focused on matching an input pattern to a time series. Ye and Keogh (2009) 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 can characterise a series class, a group of identical series, e.g., species of leaves as used by Ye and Keogh (2009), with a unique identifier. The objective is to divide the original dataset D into two sub-groups based on a threshold distance of the pivots. The pivot that ends up on maximising the distance between elements of different classes and θ, will be used as the shapelet that "identifies" class C.

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 on 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 on "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.

3. SAX/GA CPU approach

The SAX/GA algorithm (Canelas, Neves, & Horta, 2013a; 2013b) was developed and now reworked for the purpose of optimising 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.

3.1. Training approach

The algorithm uses daily historical data extracted from the S&P500 index, which is then divided in a sliding window fashion as illustrated in Fig. 2.

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, a shift is performed so that, the end of the next training window matches the end of the previous testing dataset in which the optimisation process restarts until the training dataset reaches the end or a stopping criterion is met.

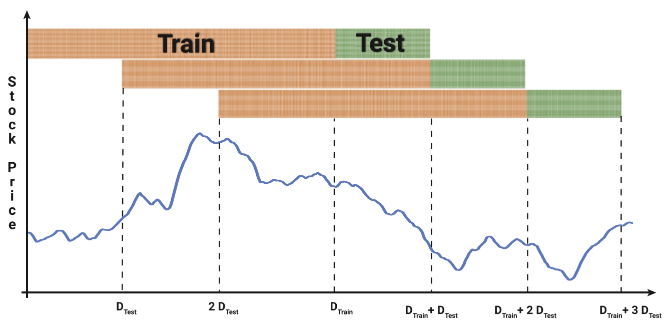


Fig. 2. Sliding window method.

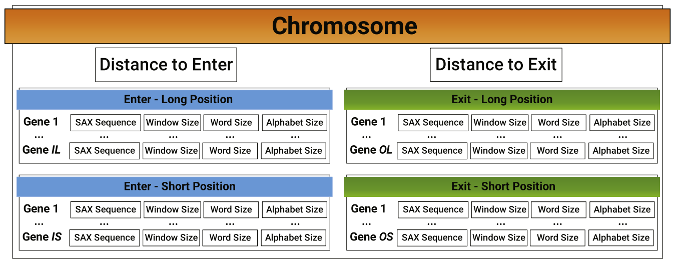

Chromosome
Distance to Enter
Distance to Exit
Enter - Long Position
Gene 1 SAX Sequence Window Size Word Size Alphabet Size
Gene *IL* SAX Sequence Window Size Word Size Alphabet Size
Exit - Long Position
Gene 1 SAX Sequence Window Size Word Size Alphabet Size
Gene *OL* SAX Sequence Window Size Word Size Alphabet Size
Enter - Short Position
Gene 1 SAX Sequence Window Size Word Size Alphabet Size
Gene *IS* SAX Sequence Window Size Word Size Alphabet Size
Exit - Short Position
Gene 1 SAX Sequence Window Size Word Size Alphabet Size
Gene *OS* SAX Sequence Window Size Word Size Alphabet Size

Fig. 3. Chromosome structure.

3.2. Population generation

The optimisation of the trading strategy starts with the generation of a random population of chromosomes with size *NumChromo*. The chromosome structure (Fig. 3) is divided into four categories, each representing a type of trading rule. These rules are used to define a market: (1) entry with a long position; (2) exit from a long position; (3) entry with a short position; and (4) exit from a short position. Each category/rule consists of one or more genes and each gene includes a SAX sequence and the SAX characterization parameters (Window, Word and Alphabet Size). To simplify the crossover operator, all four categories are created with a *NumPattern* genes but are internally limited to Enter Long (*IL*), Enter Short (*IS*), Exit Long (*OL*) and Exit Short (*OS*) number of genes.

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 portion of the dataset.

3.3. Fitness evaluation

With the population created, the GA optimisation starts by evaluating all chromosomes based on a fitness function that calculates which solution satisfy a minimum threshold, the fitness score, and shall be used later in the crossover operator. The fitness function iterates the training dataset following a Finite-state machine (FSM) with four possible states, as illustrated in Fig. 4.

- **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 state will be *S4* otherwise remains in *S1*.
- **State 2 (*S2*)** - The chromosome triggered an entry with a long position in the previous state and must now process the current balance and ROI. This sets the next state to *S1*.

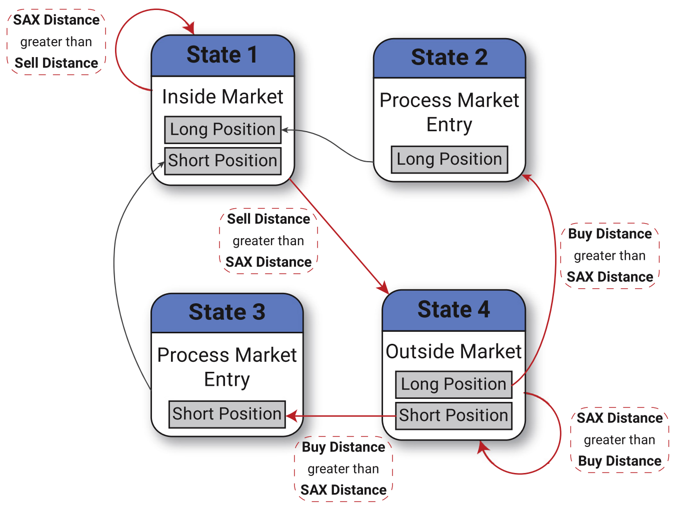

SAX Distance greater than Sell Distance
State 1
Inside Market
Long Position
Short Position
State 2
Process Market Entry
Long Position
Sell Distance greater than SAX Distance
Buy Distance greater than SAX Distance
State 3
Process Market Entry
Short Position
State 4
Outside Market
Long Position
Short Position
Buy Distance greater than SAX Distance
SAX Distance greater than Buy Distance

Fig. 4. Representation of the fitness FSM workflow.

- **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 a long or short position. 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 state (*S1* or *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 2, 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 relatively close and, in large alphabets, there is only a noticeable difference with two or more symbols and, therefore, the state transition is triggered by the product 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 price 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 Eq. (7) but instead, it takes into consideration whether, at day i, the chromosome is in a short or long position (Eq. (8)).

$$E^c_i = \begin{cases} E^c_{i-1}, & if\ Out\ of\ market \\ E^c_{i-1} + (P_{i-1} - P_i), & if\ Short\ position \\ E^c_{i-1} + (P_i - P_{i-1}), & if\ Long\ position \end{cases} \tag{8}$$

Finally, the chromosome fitness (Eq. (9)) is based on the sum of the difference between Eqs. (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.

$$Fitness\ score = \begin{cases} \dfrac{\sum_{i=0}^{D_{tsize}} |E_i - E^c_i|}{(1 + ROI)}, & E > 1, R > 0 \\ 3.40282e + 38, & E = 0, R <= 0 \end{cases} \tag{9}$$

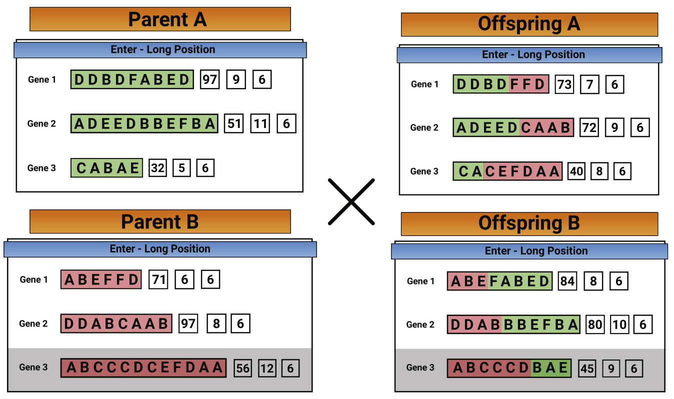


Fig. 5. Crossover example between parent A (3*IL*) and B (2*IL*).

where *E* represents number of entries and exits and *R* is the ROI indicator.

3.4. 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 achieve good results in Canelas et al. (2013a,b), and tak Zhang and jib Kim (2000) where ranking based implementations converged into higher quality solutions in fewer generations compared to tournament selections and others.

3.5. Population crossover

The SAX/GA crossover uses a slightly different approach to the conventional operator where, instead of splitting the chromosome into *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 to obtain 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 $4 \cdot NumPattern$ genes, however, it is only capable of accessing *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 and eventually, with enough iterations of the GA, all chromosomes would converge into solutions with one gene per category since an offspring inherits the parent's limits. In those cases, the crossover operator accesses all genes, bypassing the chromosome's internal limits, and is able to perform the crossover even though not all genes might be used. Fig. 5 shows a crossover example where Offspring A would not receive gene 3 if Parent B was created with less than 3 genes. The hidden gene 3 of Parent B unlocks the possibility of a complete crossover where offspring A and B can inherit the gene limits from the parents, 3 and 2 respectively, without crippling future generations of chromosomes.

3.6. Chromosome mutation

The mutation operator is only applied to a relatively 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 (Fig. 6). However, recalling Eq. (6), changing a symbol in a SAX sequence only has any effect in the SAX distance if the difference between the previous and mutated symbol is greater

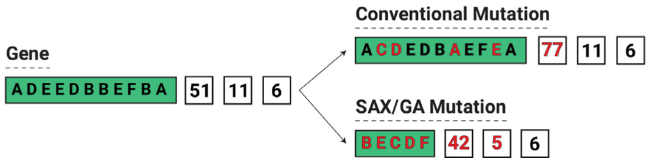


Fig. 6. Mutation comparison between the conventional and SAX/GA approach.

than one unit, e.g 'A' to 'C' or 'B' to 'E', otherwise the distance remains the same even though the gene structure was changed. So, the SAX/GA mutation selects between 1 and the internal gene limits, *IL, IS, OL* and *OS*, genes to either be completely regenerated or removed from a chromosome with the increase or decrease of the chromosome's gene limits.

4. SAX/GA CPU performance analysis

To understand whether SAX/GA can take advantage of a GPU architecture, the algorithm was benchmarked in a CPU runtime to assess the execution time and locate areas with heavy computational work under different conditions. The platform specifications are detailed in Table 3 and the benchmark parameters in Table 4. The SAX/GA sequential program was compiled using GCC compiler with *C++11* standard and optimisation flags set to *-O3*. 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 5 shows the average results for a 5 run benchmark test. Increasing the population in a ratio of two has a proportional impact in all genetic operators although, in the crossover and mutation, it only becomes visible once the number of individuals reaches 256. For an equal number of chromosomes, a larger training dataset also increases 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. 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.

5. GPU-accelerated SAX/GA

Before entering in details about the GPU-accelerated solutions, it is necessary to understand some key concepts about GPUs and how it differs from the known CPU architecture.

5.1. NVIDIA'S GPU architecture overview

NVIDIA's GPUs follow a unique architecture model, Single-Instruction Multiple-Thread or SIMT. To obtain a SIMT architecture, the GPU must be designed to execute hundreds of threads concurrently (Corporation, 2015). On a top-level, a GPU is a combination of multiple Streaming Multiprocessor (SM), independent multi-threaded units responsible from the creation, management, schedule and launch of threads, paired in groups of 32, called

Table 5
Execution time of the entire SAX/GA algorithm and genetic operators with execution environment depicted in Table 3 and execution parameters of Table 4 (time in *seconds*).

		Population size			
Training size	Genetic operator	64	128	256	512
128	Fitness	398.92	780.21	1621.75	3231.24
	Crossover	14.26	18.84	28.94	51.68
	Mutation	0.04	0.05	0.11	0.20
	Total time	413.29	799.17	1650.88	3283.19
256	Fitness	822.06	1590.17	3276.14	6467.48
	Crossover	14.01	18.09	27.85	50.11
	Mutation	0.03	0.05	0.10	0.18
	Total time	836.17	1608.39	3304.17	6517.85
512	Fitness	1503.07	3150.4	6313.04	12,156.7
	Crossover	12.57	15.51	25.64	45.4
	Mutation	0.03	0.04	0.08	0.13
	Total time	1515.74	3167.02	6338.83	12,202.32

warps. Each SM features an instruction cache, warp schedulers that selects warps ready to execute, instruction dispatch units that issues instruction to individual warps, a 32-bit register file, a shared memory, several types of cache and the most important element, the CUDA core or Streaming Processor (SP).

On the memory side, a GPU memory organization is divided in a 3 level hierarchical structure. Each level has a defined set of functions, benefits and limitations, and it is the programmer's responsibility to assure the appropriate use and correct management. All SM's are connected and can communicate through a global memory located off-chip and with a magnitude of Gigabyte that is linked to the CPU through the PCI-e bus. Being a "general" access off-chip memory leads to an important problem, the latency between requesting and retrieving information, which can be as high as 800 clock cycles depending on the device capability (NVIDIA Corporation, 2015).

On a second level, there is a set of caches and an core mechanism of communication between threads, shared memory. The latter memory consists in a fast high throughput access memory located inside each SM that is accessible, although only a small size is available (around a Kilobyte magnitude). Such advantages come with disadvantages mainly the access pattern by threads. To achieve peak throughput, organized shared memory in a modular structure of equally-sized memory modules called banks with memory lines of either 16 or 32 four bytes banks, compute capability dependent. Maximum memory bandwidth is obtained by performing read or writes in n addresses that match n unique banks however once m threads execute an instruction whose address falls in the same memory bank, it triggers a m-way bank conflict and each conflict is served in serially.

For the third and more restrict level, each SM is equipped with a 32-bit register file with the highest throughput available, dedicated for private variables of each thread.

The CUDA programming model introduced a general purpose parallel computing platform that unlocked the full potential of the GPU, combining well established programming languages with an highly parallel architecture. The execution environment consists in a kernel where it is formalized the routine that all threads will execute in the GPU and how they are organized. The way a kernel is defined reflects how the problem is spatially organized, e.g, parallel sum reduction over an array can be represented with a one-dimensional kernel while a multiplication between two matrices can take an 2*D* kernel.

5.2. Speculative GPU SAX transformation

The SAX transformation was previously identified as the main cause of high execution time and, recalling Fig. 4, there are two

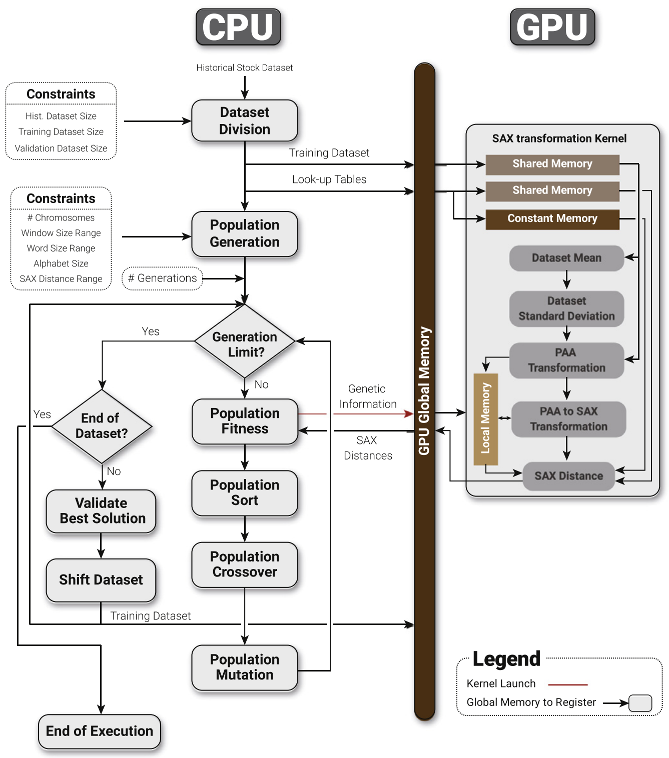


Fig. 7. Execution work-flow of SAX/GA with GPU accelerated Speculative SAX transformation.

possible states 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.

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 minimised 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 on the length of the dataset.

The fitness function is now divided into a hybrid architecture (Fig. 7) where the CPU is responsible for the iteration of the fitness

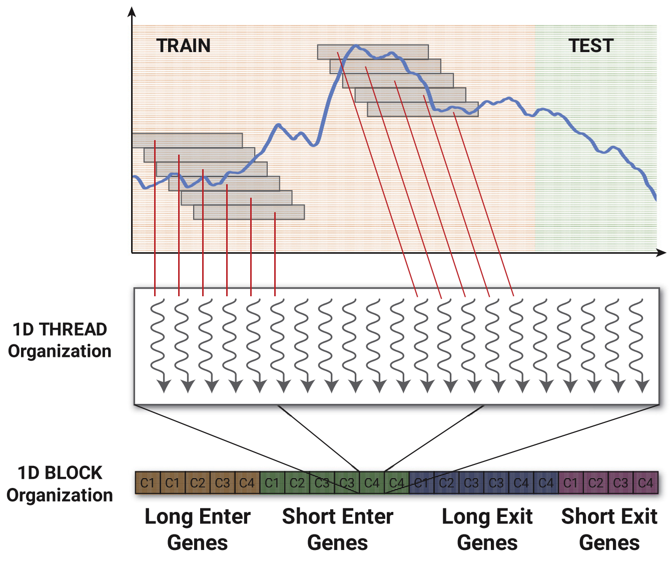


Fig. 8. Work distribution on the GPU.

FSM and the GPU performs the SAX transformation for all genes and of all FSM states, storing the SAX distance in global memory of the GPU. The CPU execution becomes far more simple only having to perform one memory transfer per generation from the GPU's global memory, iterate over the transferred array and trigger a state transition when the SAX distance is smaller than the gene's encoded distance for the corresponding state. To perform a SAX transformation, the GPU will need three types of information, the training dataset, the chromosome's genetic data, the look-up table to translate a PAA value into SAX symbol and the distance between SAX symbols (Table 2).

For each gene, a portion of the training dataset, with length D_{tsize}, needs to be converted into a SAX sequence and then compared to the gene's SAX sequence. The transformation of a part 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 number of blocks varies with the number of genes that are meant to be analysed in each generation.

Since the *ith* block is assigned with the *ith* gene, it is possible to use the GPU memory primitives in an efficient manner, mainly the shared memory. At the start, thread *j* of a block loads the *jth* element of the training dataset into shared memory and, since the portion of the dataset that thread *j* is meant to transform starts in element *j* and ends at $j + gene^{i}_{windowsize}$ of the dataset, a request to a shared memory address is served in a single memory transaction without bank conflicts. The shared memory usage is kept relatively low with $4 \cdot D_{tsize}$ bytes allocated for the training dataset plus 36 bytes for the look-up table (Table 2) and 28 bytes for the gene structure which, in the worst scenario of $D_{tsize} = 512$, approximately 2kB of shared memory is used.

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 (Fig. 8). Blocks are organised 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.

A thread execution path is divided into three different areas, the dataset normalisation, the PAA and then the SAX transformation. The data normalisation 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 normalisation, Eq. (2) is partially modified so that, the PAA transformation is performed at the time of the window normalisation, reducing memory transactions and usage while decreasing the number of instructions, merging Eqs. (1) and (2). The SAX transformation is a relatively simple operation where, for each PAA value, a thread iterates through a look-up table and translates the value into a SAX symbol. Once the *kth* SAX symbol is obtained, the distance between the gene and the converted *kth* symbol is calculated with Eq. (6) and is later saved in a GPU global memory array.

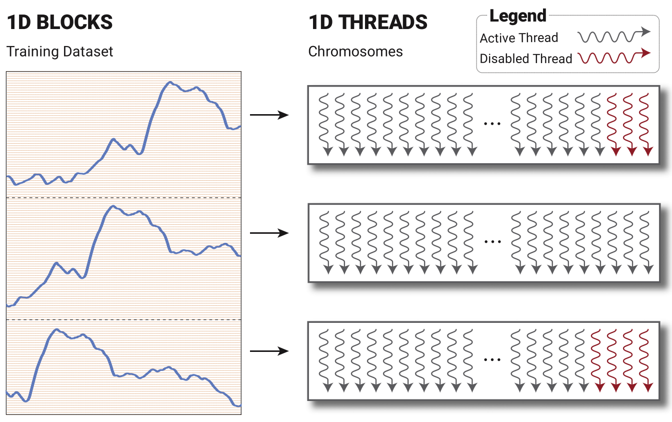


Fig. 9. Example of the FSM kernel organization.

5.3. Parallel SAX/GA training

SAX/GA optimises 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 resetting the optimisation process. 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 training windows.

In an attempt to increase the level of parallelism achieved with Solution A and understand whether the continuous process of optimisation 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 the FSM iteration, crossover and mutation, notice 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 fully GPU method.

For the SAX kernel, the main difference is the kernel configuration. Previously, the kernel took a one-dimensional grid of blocks where a gene was associated with each block. The parallel training allows a two-dimensional configuration where the *x*-axis of a 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 an identical behaviour of the kernel.

The FSM iteration has a simple organisation with the training datasets being distributed across a one-dimensional grid of blocks with threads exploring a one-dimensional space (Fig. 9). The number of threads in each block depends on the number of individuals meant to have the fitness score recalculated and, there is the possibility that two blocks will have a different number of threads.

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

5.4. 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 on the crossover and mutation operators, along with the population creation that was 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 will require three new GPU implementations for the population generation, crossover and mutation of individuals while incorporating the existing kernels (Fig. 12).

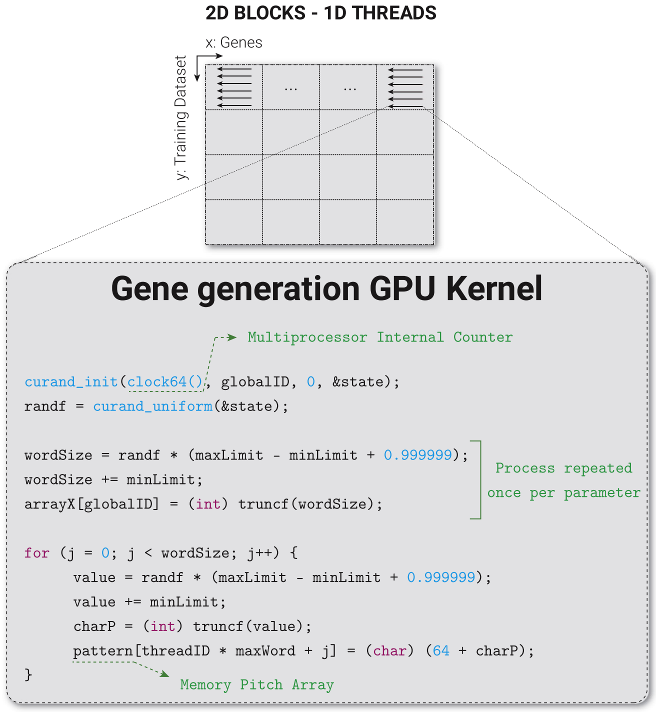
5.4.1. Population generation kernel

The generation of a gene is an independent process with an identical execution flow for all genes and can easily be 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 given that the length of each gene is randomly generated.

The kernel (Fig. 10) takes a two-dimensional block organisation where the y-axis is used as an indicative of the training window (Fig. 2) while in the blocks of the x-axis, threads generate the necessary genes for that window. The block organisation is calculated so that, it has enough threads to help mask memory latency and it has a power-of-two threads and there is an even distribution of blocks in the SM's. For example, with a population of 256 individual trained for 128 days and validated with 60 days, there will be 46 training windows (y-axis) each with 3072 genes (12 genes per individual) divided across 12 blocks (x-axis) with 256 threads each. This process generates a total of 141, 312 genes each requiring 28 bytes of memory.

5.4.2. 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 on 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 a single-point crossover is used. The best-ranked individuals are selected and used to generate two offspring that will take the position of the worst chromosomes. This process is executed in parallel with each crossover being performed by a thread.


2D BLOCKS - 1D THREADS
x: Genes
y: Training Dataset
...
Gene generation GPU Kernel
Multiprocessor Internal Counter
Process repeated once per parameter
Memory Pitch Array

```
curand_init(clock64(), globalID, 0, &state);
randf = curand_uniform(&state);

wordSize = randf * (maxLimit - minLimit + 0.999999);
wordSize += minLimit;
arrayX[globalID] = (int) truncf(wordSize);

for (j = 0; j < wordSize; j++) {
      value = randf * (maxLimit - minLimit + 0.999999);
      value += minLimit;
      charP = (int) truncf(value);
      pattern[threadID * maxWord + j] = (char) (64 + charP);
}
```

Fig. 10. Execution flow of SAX/GA population generation kernel.

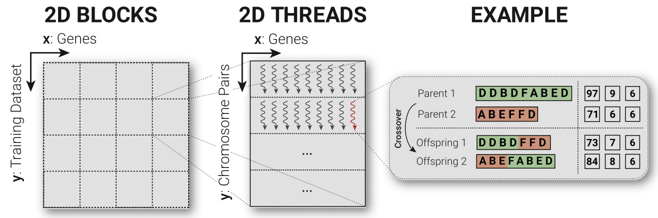


Fig. 11. Kernel configuration for the crossover operator with an execution example of a thread.

As for the kernel configuration (Fig. 11), the blocks are divided in a two-dimensional organization, identical to previous kernels, with the training windows being distributed across the y-axis while the number of x-axis blocks depends on the number of chromosomes and genes processed in a similar fashion as before. The threads are also 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 (e.g y_0, x_0 for parent A and B, y_0, x_1 for parent C and D, both from training window 0).

5.4.3. Mutation kernel

The SAX/GA mutation operator is an aggressive approach to introducing diversity in the population of solutions and is applied at 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 SAX 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 total amount of chromosomes selected are based on the mutation rate and the ceiling of this value to a power-of-two genes per training dataset.

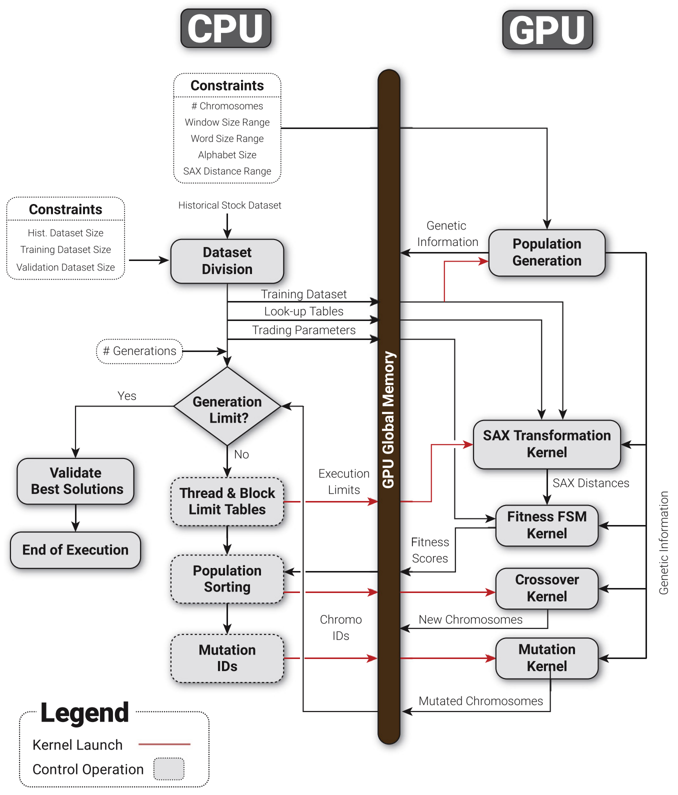


Fig. 12. Example of the FSM kernel organization.

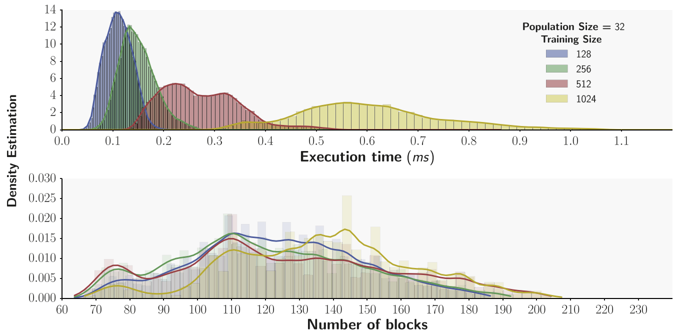


Fig. 13. Distribution of the SAX kernel execution time and workload along 4 different training datasets.

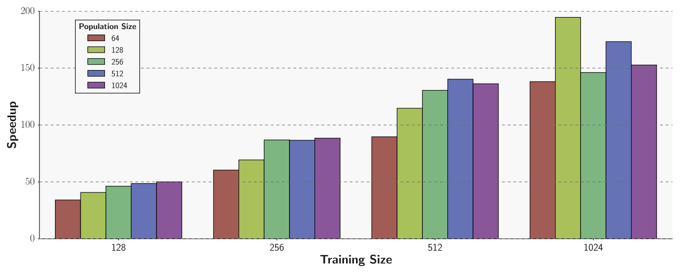


Fig. 14. Speedup of solution A.

6. Experimental results

The two metrics used to evaluate each solution are the speedup (Eq. (10)) and ROI indicator (Eq. (11)) which validates the quality of the trading strategy.

$$Speedup = \frac{\text{Execution Time of Original SAX/GA}}{\text{Execution Time of Solution } X} \tag{10}$$

$$ROI(day_t) = \frac{Price(day_t) - Price(day_0)}{Price(day_0)} \tag{11}$$

To each solution, the benchmark described in Table 3 was applied with an additional population size, 1024 chromosomes, and training size, 1024 days. 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.

6.1. 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 the *gcc* compiler using *C++11* standard. The optimisation 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.

6.1.1. 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.

Observing the results in Table 6 and recalling that the SAX representation was responsible for 99.5% of the total execution time, this solution no longer faces this disparity between the total time of the fitness function and that spent in transforming SAX sequences. If two kernel calls were selected with an identical number of blocks and a different number 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 6 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 to solutions with different configurations, In Fig. 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 (Fig. 14) achieved proved that this implementation is capable of taking advantage of the GPU with values that range from 34 times and up to almost 160 times.

6.1.2. Parallel SAX/GA training

Parallel training was introduced as an attempt to increase the level of parallelism of the previous solution and to understand if the continuous optimisation of a population does in fact being benefits to the quality of solutions.

The expected increase of workload in all genetic operators showed significant impact on the fitness function where the time spent on control operations such memory allocations and transactions and generation of look-up tables, rose 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 (Fig. 15).

Even with this 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. Fig. 16 presents the speedup of solution B.

Comparing the speedup differential between solution A and B (Fig. 17), solution B shows better speedups smaller for populations however with the increase of this, the crossover operator intro-

Table 6
Execution time (in *milliseconds*) of the SAX kernel and the fitness function per generation.

Genetic operator	Training size	Population size 64	128	256	512	1024
SAX kernel (*ms*/generation)	128	0.149	0.203	0.317	0.557	1.054
	256	0.222	0.354	0.598	1.037	1.948
	512	0.445	0.684	1.273	2.411	5.207
	1024	0.936	1.834	3.769	7.912	12.681
Fitness (*ms*/generation)	128	0.799	1.072	1.406	2.051	3.451
	256	1.004	1.432	1.996	3.145	5.490
	512	1.417	2.152	2.786	6.462	12.148
	1024	2.361	3.909	7.318	15.290	25.448

Table 7
Speedup of solution A relatively to the GPU sequential implementation and the SM occupancy for the average workload of each test.

Training size	Population size 64	128	256	512	1024
128	34.05 (76.7%)	41.04 (85.1%)	46.54 (91.5%)	49.74 (94.9%)	50.12 (96.5%)
256	87.70 (86.9%)	102.42 (93.2%)	129.94 (94.9%)	132.93 (96.4%)	149.17 (96.9%)
512	140.74 (95.3%)	153.46 (96.4%)	181.79 (97.1%)	186.44 (97.4%)	176.42 (97.5%)
1024	133.80 (98.0%)	148.42 (98.1%)	157.49 (98.3%)	153.52 (98.4%)	139.19 (98.5%)

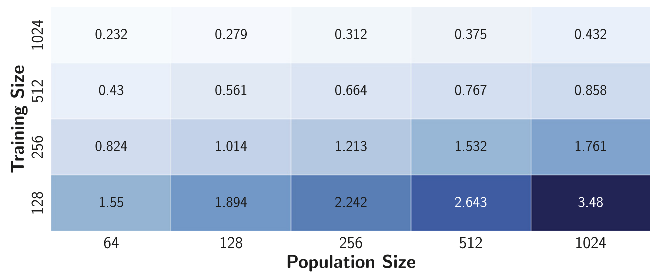


Fig. 15. Variation of the ratio between the execution time of the crossover and fitness operators.

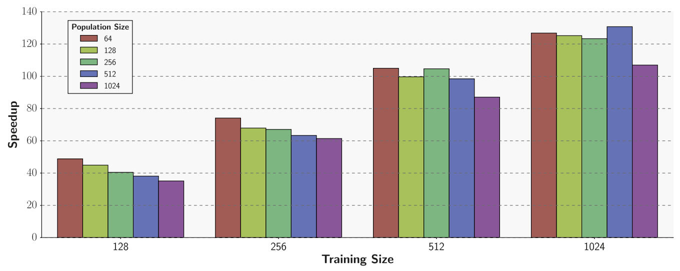


Fig. 16. Speedup of solution B.

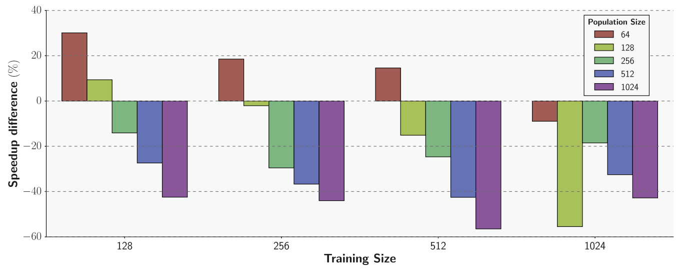


Fig. 17. Difference between the speedup of Solution B and A.

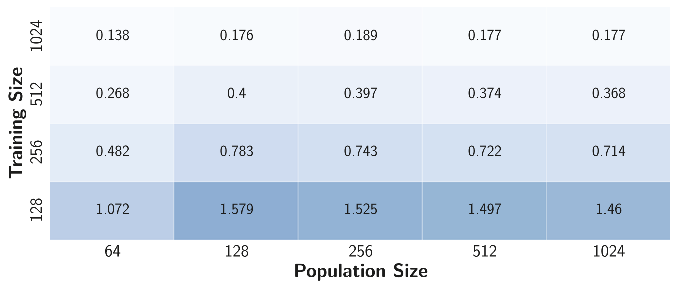


Fig. 18. Variation of the ratio between the crossover and fitness operators execution time.

duces a negative impact in the algorithm and forces the speedup difference to decrease.

The fundamental idea that solution B was based on, the parallel dataset training, ended up introducing limitations to the SAX/GA algorithm. The fitness control operations forced the CPU to perform a task that saw its load increase greatly along with a very significant impact from the crossover operator that reduces the speedup gain versus solution A.

6.1.3. 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 from 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% in 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 the GPU, which did improve the crossover/fitness ratio as displayed in Fig. 18. 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 draw a conclusion.

The speedup of solution C (Fig. 18) 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.

As for the speedup difference between solution A and C (Fig. 19), 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.

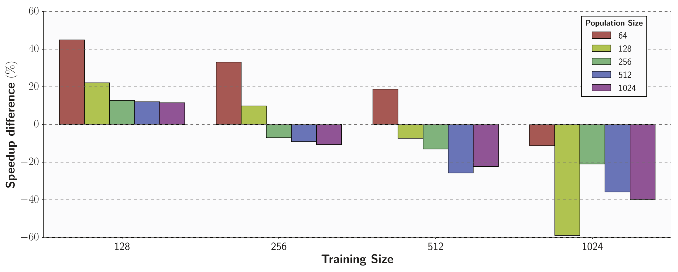


Fig. 19. Difference between the speedup of Solution C and A.

The most noticeable improvements in solution C are with training sizes of 128 where all populations were 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.

6.2. *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 optimised by each implementation are identical to the original version of SAX/GA with the main difference being the type of optimisation used. The similarity level between an original CPU SAX/GA and a GPU SAX/GA solution is not determined by the chromosome structure and if they have an identical SAX sequence but by the performance of both solutions. Since the GA does not converge to a unique global solution with only one possible configuration, the performance is measured during the test period using the ROI indicator.

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 (January 1998 until April 2010) 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 of time 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 optimisation process, solution A is used as the continuous optimisation solution and implementation C for the parallel training. For each solution, the average and best ROI were calculated.

Considering the average case, SAX/GA was able to converge to different individuals that were capable of performing trading actions that led to a positive ROI. On average, the vast majority of test cases obtained a positive ROI with a few negative ROI's spread across all tests (Fig. 20) 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 trading strategy obtained by each solution is influenced by multiple parameters and it is almost guaranteed that such solution is different from any other. Observing Fig. 21, solution A was the only one that achieved only positive ROI's 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 optimisation strategy, the continuous optimisation, however, solution C presents overall higher ROI's compared to A even though there is a test case that leads to losses for the user.

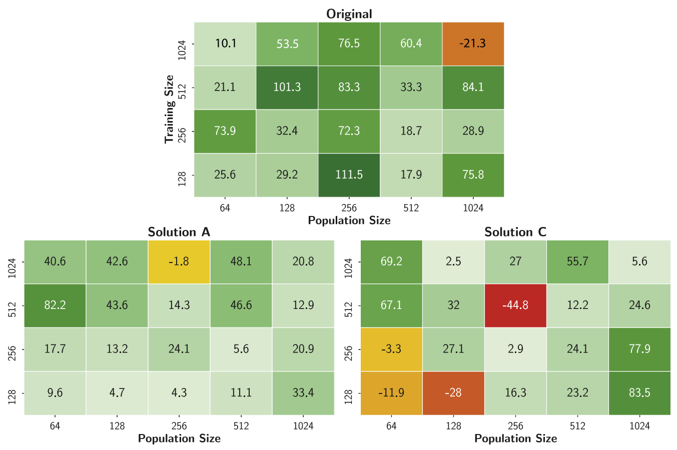


Fig. 20. Average ROI (%) for all test cases.

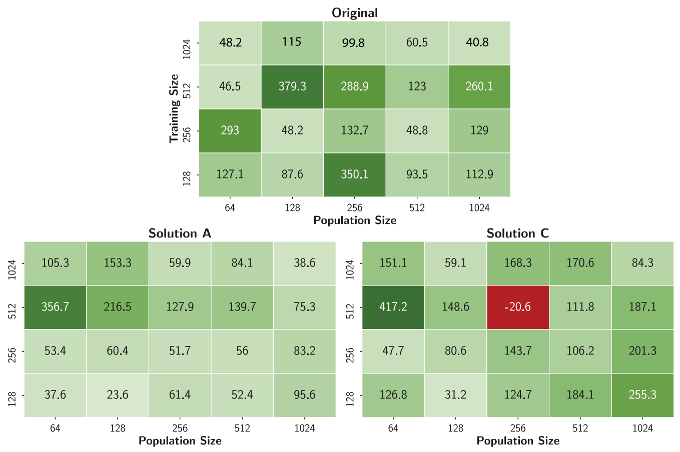


Fig. 21. Best ROI (%) for all test cases.

7. Conclusions

In conclusion, the SAX/GA approach was initially designed to be an evolutionary algorithm that could explore a vast search space with relatively small populations of individuals. To accomplish the implementation, genetic operators modify the information of an individual in far more aggressive manners than conventional GA's, however, their complex implementation leads to partially inefficient executions in the GPU. In this work, a benchmark was produced for the SAX/GA approach considering a CPU execution, which allowed the identification of critical bottlenecks and, based on that, three GPU based solutions were implemented and tested, resulting in gains of 30 times to 180 times. Optimising solely the fitness function in solution A provided the most consistent results with a near linear gain with the increase of threads up until the saturation of the GPU. Even though solution B and C failed to meet the baseline speedup of solution A, both solutions show that the remaining GA operators still have a saying the performance of the algorithm but the implemented solutions suffered from a highly complex design that needs further research to either simplify the operator workflow with a detailed study of the impact of the SAX parameters or improve the current optimisation. Nevertheless, the presented results show not only a significant improvement in performance but also an advance in the state-of-the-art of published computational finance solutions.

Acknowledgments

This work was supported in part by Fundação para a Ciência e a Tecnologia (Project UID/EEA/50008/2013).

References

Bakhach, A., Tsang, E. P. K., & Jalalian, H. (2016a). Forecasting directional changes in the fx markets. *Computational intelligence in financial engineering*. IEEE.

Bakhach, A., Tsang, E. P. K., Ng, W. L., & Chinthalapati, V. L. R. (2016b). Backlash agent: A trading strategy based on directional change. *Computational intelligence in financial engineering*. IEEE.

Canelas, A., Neves, R., & Horta, N. (2013a). Multi-dimensional pattern discovery in financial time series using sax-ga with extended robustness. *GECCO*.

Canelas, A., Neves, R., & Horta, N. (2013b). A sax-ga approach to evolve investment strategies on financial markets based on pattern discovery techniques. *Expert Systems with Applications, 40*(5), 1579–1590.

Chang, K. W., Deka, B., Hwu, W. M. W., & Roth, D. (2012). Efficient pattern-based time series classification on GPU. In *Proceedings - IEEE international conference on data mining, ICDM* (pp. 131–140).

Chen, C. H., Tseng, V., Yu, H. H., & Hong, T. P. (2013). Time series pattern discovery by a pip-based evolutionary approach. *Soft Computing, 17*(9), 1699–1710.

Chen, W. S., Hsieh, L., & Yuan, S. Y. (2004). High performance data compression method with pattern matching for biomedical ecg and arterial pulse waveforms. *Computer Methods and Programs in Biomedicine, 74*(1), 11–27.

NVIDIA C. (2015). Nvidia cuda compute unified device architecture programming guide. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, Accessed: 15-11-2015.

Fu, T. c., Chung, F. l., Luk, R., & Ng, C. m. (2007). Stock time series pattern matching: Template-based vs. rule-based approaches. *Engineering Applications of Artificial Intelligence, 20*(3), 347–364.

Iglesias, F., & Kastner, W. (2013). Analysis of similarity measures in times series clustering for the discovery of building energy patterns. *Energies, 6*(2), 579.

Li, H., Guo, C., & Qiu, W. (2011). Similarity measure based on piecewise linear approximation and derivative dynamic time warping for time series mining. *Expert Systems with Applications, 38*(12), 14732–14743.

Lkhagva, B., Suzuki, Y., & Kawagoe, K. (2006). Extended sax: Extension of symbolic aggregate approximation for financial time series data representation. *DEWS*.

tak Zhang, B., & jib Kim, J. (2000). Comparison of selection methods for evolutionary optimization. *Evolutionary Optimization, 2*, 55–70.

Wei, L. (2006). Sax: N/n not equal an integer case.

Ye, L., & Keogh, E. (2009). Time series shapelets. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining - KDD '09* (p. 947).

Zapranis, A., & Tsinaslanidis, P. (2010). Identification of the head-and-shoulders technical analysis pattern with neural networks. In K. Diamantaras, W. Duch, & L. Iliadis (Eds.), *Artificial neural networks ICANN 2010, volume 6354 of lecture notes in computer science* (pp. 130–136). Berlin Heidelberg: Springer.