High-speed detection of emergent market clustering via an unsupervised parallel genetic algorithm

AUTHORS:
Dieter Hendricks[1]
Tim Gebbie[1]
Diane Wilcox[1]

AFFILIATION:
[1]School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa

CORRESPONDENCE TO:
Dieter Hendricks

EMAIL:
dieter.hendricks@students.wits.ac.za

POSTAL ADDRESS:
School of Computer Science and Applied Mathematics, University of the Witwatersrand, Private Bag, Wits 2050, South Africa

DATES:
Received: 01 Oct. 2014
Revised: 12 May 2015
Accepted: 24 June 2015

KEYWORDS:
unsupervised clustering; genetic algorithms; parallel algorithms; financial data processing; maximum likelihood clustering

HOW TO CITE:
Hendricks D, Gebbie T, Wilcox D. High-speed detection of emergent market clustering via an unsupervised parallel genetic algorithm. S Afr J Sci. 2016;112(1/2), Art. #2014-0340, 9 pages. http://dx.doi.org/10.17159/sajs.2016/20140340

We implement a master-slave parallel genetic algorithm with a bespoke log-likelihood fitness function to identify emergent clusters within price evolutions. We use graphics processing units (GPUs) to implement a parallel genetic algorithm and visualise the results using disjoint minimal spanning trees. We demonstrate that our GPU parallel genetic algorithm, implemented on a commercially available general purpose GPU, is able to recover stock clusters in sub-second speed, based on a subset of stocks in the South African market. This approach represents a pragmatic choice for low-cost, scalable parallel computing and is significantly faster than a prototype serial implementation in an optimised C-based fourth-generation programming language, although the results are not directly comparable because of compiler differences. Combined with fast online intraday correlation matrix estimation from high frequency data for cluster identification, the proposed implementation offers cost-effective, near-real-time risk assessment for financial practitioners.

Introduction

Advances in technology underpinning multiple domains have increased the capacity to generate and store data and metadata relating to domain processes. The field of data science is continuously evolving to meet the challenge of gleaning insights from these large data sets, with extensive research in exact algorithms, heuristics and meta-heuristics for solving combinatorial optimisation problems. The primary advantage of using exact methods is the guarantee of finding the global optimum for the problem. However, a disadvantage when solving complex (NP-hard) problems is the exponential growth of the execution time proportional to the problem instance size.[1] Heuristics tend to be efficient, but solution quality cannot be guaranteed and techniques often are not versatile.[2] Meta-heuristics attempt to consolidate these two approaches and deliver an acceptable solution in a reasonable time frame. A large number of meta-heuristics designed for solving complex problems exist in the literature and the genetic algorithm (GA) has emerged as a prominent technique, using intensive global search heuristics that explore a search space intelligently to solve optimisation problems.

Although the algorithms must traverse large spaces, the computationally intensive calculations can be performed independently. Compute Unified Device Architecture (CUDA) is Nvidia's parallel computing platform which is well suited to many computational tasks, particularly those for which data parallelism is possible. Implementing a GA to perform cluster analysis on vast data sets using this platform allows one to mine through the data relatively quickly and at a fraction of the cost of those of large data centres or computational grids.

A number of authors have considered parallel architectures to accelerate GAs.[3-10] While the work of Kromer et al.[7] is conceptually similar to the implementation proposed in this paper, a key difference is our choice of fitness function for the clustering scheme.

Giada and Marsili [11] propose an unsupervised, parameter-free approach to finding data clusters, based on the maximum likelihood principle. They derive a log-likelihood function, where a given cluster configuration can be assessed to determine whether it represents the inherent structure for the data set: cluster configurations which approach the maximum log-likelihood are better representatives of the data structure. This log-likelihood function is thus a natural candidate for the fitness function in a GA implementation, where the population continually evolves to produce a cluster configuration which maximises the log-likelihood. The optimal number of clusters is a free parameter, unlike in traditional techniques where the number of clusters needs to be specified *a priori*. While unsupervised approaches have been considered (see Omran et al.[12] and references therein), the advantage of the Giada and Marsili approach is that it has a natural interpretation for clustering in the application domain explored here.

Monitoring intraday clustering of financial instruments allows one to better understand market characteristics and systemic risks. While GAs provide a versatile methodology for identifying such clusters, serial implementations are computationally intensive and can take a long time to converge to a best approximation. In this paper, we introduce a maintainable and scalable master-slave parallel genetic algorithm (PGA) framework for unsupervised cluster analysis on the CUDA platform, which is able to detect clusters using the Giada and Marsili likelihood function. Applying the proposed cluster analysis approach and examining the clustering behaviour of financial instruments, offers a unique perspective to monitor the intraday characteristics of the stock market and the detection of structural changes in near real time. The novel implementation presented in this paper builds on the contribution of Cieslakiewicz[13]. While we provide an overview and specific use-case for the algorithm in this paper, we also are investigating aspects of adjoint parameter tuning, performance scalability and the impact on solution quality for varying stock universe sizes and cluster types.

Cluster analysis

Cluster analysis groups objects according to metadata describing the objects or their associations.[14] The goal is to ensure that objects within a group exhibit similar characteristics and are unrelated to objects in other groups. The greater the homogeneity within a group, and the greater the heterogeneity between groups, the more pronounced the clustering. In order to isolate clusters of similar objects, one needs to utilise a data clustering approach that will recover inherent structures efficiently.

The correlation measure of similarity

The correlation measure is an approach to standardise data by using the statistical interdependence between data points. Correlation indicates the direction (positive or negative) and the degree or strength of the relationship between two data points. The most common correlation coefficient which measures the relationship between data points is the Pearson correlation coefficient, which is sensitive only to a linear relationship between points. The Pearson correlation is +1 in the case of a perfect positive linear relationship, -1 in the case of a perfect negative linear relationship and some value between -1 and +1 in all other cases, with values close to 0 signalling negligible interdependence.

Clustering procedures

Any specific clustering procedure entails optimising some kind of criterion, such as minimising the within-cluster variance or maximising the distance between the objects or clusters.

Cluster analysis based on the maximum likelihood principle

Maximum likelihood estimation is a method of estimating the parameters of a statistical model. Data clustering on the other hand deals with the problem of classifying or categorising a set of N objects or clusters, so that the objects within a group or cluster are more similar than objects belonging to different groups. If each object is identified by D measurements, then an object can be represented as a tuple, $\bar{x}_i = (x_i^{(1)},\ldots,x_i^{(D)})$, $i=1,\ldots,N$ in a D-dimensional space. Data clustering is used to identify clusters as more densely populated regions in this vector space. Thus, a configuration of clusters is represented by a set $S=\{s_1,\ldots,s_N\}$ of integer labels, where s_i denotes the cluster to which object i belongs and N is the number of objects[11] (if $s_i=s_j=s$, then object i and object j reside in the same cluster), and if s_i takes on values from 1 to M and $M=N$, then each cluster is a singleton cluster constituting one object only.

Analogy to the Potts model

One can apply super-paramagnetic ordering of a q-state Potts model directly for cluster identification.[15] In a market Potts model, each stock can take on q-states and each state can be represented by a cluster of similar stocks.[15-17] Cluster membership is indicative of some commonality among the cluster members. Each stock has a component of its dynamics as a function of the state it is in and a component of its dynamics influenced by stock specific noise. In addition, there may be global couplings that influence all the stocks, i.e. the external field that represents a market mode.

In the super-paramagnetic clustering approach, the cost function can be considered as a Hamiltonian function whose low energy states correspond to cluster configurations that are most compatible with the data sample. Structures are then identified with configurations $S = \{s_i\}_{i=1}^N$ for the cluster indices s_i which represent the cluster to which the ith object belongs. This allows one to interpret s_i as a Potts spin in the Potts model Hamiltonian with J_{ij} decreasing with the distance between objects.[15,16] The Hamiltonian takes on the form:

$$H_g = -\sum_{s_i,s_j\in S} J_{ij}\,\delta\,(s_i, s_j) - \frac{1}{\beta}\sum_i h_i^M s_i, \qquad \text{Equation 1}$$

where the spins s_j can take on q-states and the external magnetic fields are given by h_i^M. The first term represents common internal influences and the second term represents external influences. We ignore the second term when fitting data, as we include shared factors directly in later sections when we discuss information and risk and the influence of these on price changes.

In the Potts model approach, one can think of the coupling parameters J_{ij} as being a function of the correlation coefficient.[16,17] The coupling parameters are used to specify a distance function that decreases with distance between objects. If all the spins are related in this way then each pair of spins is connected by some non-vanishing coupling, $J_{ij} = J_{ij}(c_{ij})$. In this model, the case where there is only one cluster can be thought of as a ground state. As the system becomes more excited, it could break up into additional clusters and each cluster would have specific Potts magnetisations, even though net magnetisation may remain zero for the complete system. Generically, the correlation would then be both a function of time and temperature in order to encode both the evolution of clusters as well as the hierarchy of clusters as a function of temperature. In the basic approach, one is looking for the lowest energy state that fits the data. In order to parameterise the model efficiently one can choose to make the Noh[18] ansatz and use this to develop a maximum-likelihood approach of Giada and Marsili[17] rather than explicitly solving the Potts Hamiltonian numerically[15,16].

Giada and Marsili clustering technique

Following Giada and Marsili[17], we assume that price increments evolve under Noh[18] model dynamics, whereby objects belonging to the same cluster should share a common component:

$$\bar{x}_i = g_{s_i}\bar{\eta}_{s_i} + \sqrt{1-g^2_{s_i}}\,\bar{\varepsilon}_i. \qquad \text{Equation 2}$$

Here, $\bar{x}_i$ represents the features of object i and s_i is the label of the cluster to which the object belongs. The data have been normalised to have zero mean and unit variance. $\bar{\varepsilon}_i$ is a vector describing the deviation of object i from the features of cluster s and includes measurement errors, while $\bar{\eta}_{s_i}$ describes cluster-specific features. g_s is a loading factor that emphasises the similarity or difference between objects in cluster s. In this research the data set refers to a set of the objects, denoting N assets or stocks, and their features are prices across D days in the data set. The variable i is indexing stocks or assets, whilst d is indexing days.

If $g_s=1$, all objects with $s_i=s$ are identical, whilst if $g_s=0$, all objects are different. The range of the cluster index is from 1 to N in order to allow for singleton clusters of one object or asset each.

If one takes Equation 2 as a statistical hypothesis and assumes that both $\bar{\eta}_{s_i}$ and $\bar{\varepsilon}_s$ are Gaussian vectors with zero mean and unit variance, for values of $i,s=1,\ldots,N$, it is possible to compute the probability density $P(\{\bar{x}_i\} \mid G,S)$ for any given set of parameters $(G,S)=(\{g_s\},\{s_i\})$ by observing the data set $\{x_i\},i,s=1,\ldots,N$ as a realisation of the common component of Equation 2 as follows[11]:

$$P(\{\bar{x}_i\} \mid G,S) = \prod_{d=1}^{D}\left\langle \prod_{i=1}^{N}\delta(x_i\,(d) - g_{s_i}\,\bar{\eta}_{s_i} + \sqrt{1-g^2_{s_i}}\,\bar{\varepsilon}_i)\right\rangle. \qquad \text{Equation 3}$$

The variable δ is the Dirac delta function and $\langle\ldots\rangle$ denotes the mathematical expectation. For a given cluster structure S, the likelihood is maximal when the parameter g_S takes the values

$$g_S^* = \begin{cases} \sqrt{\dfrac{c_S - n_S}{n_S^2 - n_S}} & \text{for } n_s > 1, \\ 0 & \text{for } n_s \leq 1. \end{cases} \qquad \text{Equation 4}$$

The quantity n_s in Equation 4 denotes the number of objects in cluster s, i.e.

$$n_s = \sum_{i=1}^{N}\delta_{s_i,s}. \qquad \text{Equation 5}$$

The variable c_s is the internal correlation of the sth cluster, denoted by:

$$c_s = \sum_{i=1}^{N}\sum_{j=1}^{N} C_{i,j}\,\delta_{s_i,s}\,\delta_{s_j,s}. \qquad \text{Equation 6}$$

The variable $C_{i,j}$ is the Pearson correlation coefficient of the data, denoted by:

$$C_{i,j} = \frac{\bar{x}_i\,\bar{x}_j}{\sqrt{||\,\bar{x}_i^2\,||\,||\,\bar{x}_j^2\,||}}. \qquad \text{Equation 7}$$

The maximum likelihood of structure S can be written as $P(G^*,S\,|\,\overline{x}_i) \propto \exp^{DL(S)}$ (see Sornette[19]), where the resulting likelihood function per feature L_c is denoted by:

$$L_c(S) = \frac{1}{2}\sum_{s:n_s>1}\left(\log\frac{n_s}{c_s} + (n_s-1)\log\frac{n_s^2-n_s}{n_s^2-c_s}\right). \qquad \text{Equation 8}$$

From Equation 8, it follows that $L_c=0$ for clusters of objects that are uncorrelated, i.e. where $g_s^*=0$ or $c_s=n_s$ or when the objects are grouped in singleton clusters for all the cluster indexes ($n_s=1$). Equation 8 illustrates that the resulting maximum likelihood function for S depends on the Pearson correlation coefficient $C_{i,j}$ and hence exhibits the following advantages in comparison to conventional clustering methods:

- It is *unsupervised*: The optimal number of clusters is unknown *a priori* and not fixed at the beginning
- The interpretation of results is *transparent* in terms of the model, namely Equation 2.

Giada and Marsili[11] propose that $\max_s L_c(S)$ provides a measure of structure inherent in the cluster configuration represented by the set $S=\{s_1,...,s_n\}$. The higher the value, the more pronounced the structure.

Parallel genetic algorithms

In order to localise clusters of normalised stock returns in financial data, Giada and Marsili made use of a simulated annealing algorithm,[11,17] with $-L_c$ as the cost function for their application on real-world data sets to substantiate their approach. This simulated annealing algorithm was then compared to other clustering algorithms, such as K-means, single linkage, centroid linkage, average linkage, merging and deterministic maximisation.[11] The technique was successfully applied to South African financial data by Mbambiso[20], using a serial implementation of a simulated annealing algorithm[20,21].

Simulated annealing and deterministic maximisation provided acceptable approximations to the maximum likelihood structure, but were inherently computationally expensive. We promote the use of PGAs as a viable approach to approximate the maximum likelihood structure. L_c will be used as the fitness function and a PGA algorithm will be used to find the maximum for L_c, in order to efficiently isolate clusters in correlated financial data.

GA principle and genetic operators

One of the key advantages of GAs is that they are conceptually simple. The core algorithm can be summarised into the following steps: (1) initialise population, (2) evolve individuals, (3) evaluate fitness and (4) select individuals to survive to the next generation. GAs exhibit the trait of broad applicability,[22] as they can be applied to any problem whose solution domain can be quantified by a function which needs to be optimised.

Specific genetic operators are applied to the parents, in the process of reproduction, which then give rise to offspring. The genetic operators can be classified as follows:

Selection: The purpose of selection is to isolate fitter individuals in the population and allow them to propagate in order to give rise to new offspring with higher fitness values. We implemented the *stochastic universal sampling selection operator*, in which individuals are mapped to contiguous segments on a line in proportion to their fitness values.[23] Individuals are then selected by sampling the line at uniformly spaced intervals. Although fitter individuals have a higher probability of selection, this technique improves the chances that weaker individuals will be selected, allowing diversity to enter the population and reducing the probability of convergence to a local optimum.

Crossover: Crossover is the process of mating two individuals, with the expectation that they can produce a fitter offspring.[22] The crossover genetic operation involves the selection of random loci to mark a cross site within the two parent chromosomes, copying the genes to the offspring. A bespoke *knowledge-based crossover* operator[13] was developed for our implementation, in order to incorporate domain knowledge and improve the rate of convergence.

Mutation: Mutation is the key driver of diversity in the candidate solution set or search space.[22] It is usually applied after crossover and aims to ensure that genetic information is randomly distributed, in order to avoid convergence to local minima. It introduces new genetic structures in the population by randomly modifying some of its building blocks and enables the algorithm to traverse the search space globally.

Elitism: Coley[24] states that fitness-proportional selection does not necessarily favour the selection of any particular individual, even if it is the fittest. Thus, the fittest individuals may not survive an evolutionary cycle. Elitism is the process of preserving the fittest individuals by direct promotion to the next generation, without any genetic transformations due to crossover or mutation.[22]

Replacement: Replacement is the last stage of any evolution cycle, in which the algorithm replaces old members of the current population with new members.[22] This mechanism ensures that the population size remains constant, while the weakest individuals in each generation are dropped.

Although GAs are very effective for solving complex problems, this positive trait can unfortunately be offset by long execution times caused by the traversal of the search space. GAs lend themselves to parallelisation, provided the fitness values can be determined independently for each of the candidate solutions. While a number of schemes have been proposed in the literature to achieve this parallelisation,[8,22,25] we have chosen to implement the *master-slave* model.

Master-slave parallelisation

Master-slave PGAs, also denoted as global PGAs, involve a single population, distributed amongst multiple processing units for determination of fitness values and the consequent application of genetic operators. They allow for computation on shared-memory processing entities or any type of distributed system topology, for example grid computing.[8]

Ismail[25] provides a summary of the key features of the master-slave PGA: the algorithm uses a single population (stored by the master) and the fitness evaluation of all of the individuals is performed in parallel (by the slaves). Communication occurs only as each slave receives an individual (or subset of individuals) to evaluate and when the slaves return the fitness values, sometimes after mutation has been applied with the given probability. The particular algorithm implemented in this paper is synchronous, i.e. the master waits until it has received the fitness values for all individuals in the population before proceeding with selection and mutation. The synchronous master-slave PGA thus has the same properties as a conventional GA, except evaluation of the fitness of the population is achieved at a faster rate. The algorithm is relatively easy to implement and a significant speed-up can be expected if the communications cost does not dominate the computation cost. The whole process has to wait for the slowest processor to finish its fitness evaluations until the selection operator can be applied.

A number of authors have used the Message Parsing Interface (MPI) paradigm to implement a master-slave PGA. Digalakis and Margaritis[26] implement a synchronous MPI PGA and shared-memory PGA, whereby fitness computations are parallelised and other genetic operators are applied by the master node only. They demonstrate a computation speed-up which scales linearly with the number of processors for large population sizes. Zhang et al.[27] use a centralised control island model to concurrently apply genetic operators to sub-groups, with a bespoke migration strategy using elite individuals from sub-groups. Nan et al.[28] used the MATLAB parallel computing and distributed computing toolboxes to develop a master-slave PGA, demonstrating its efficacy on the image registration problem when using a cluster computing configuration.

For our implementation, we made use of the Nvidia CUDA platform to achieve massive parallelism by utilising the graphical processing unit (GPU) streaming multiprocessors (SM) as slaves, and the central processing unit (CPU) as master.

Computational platform and implementation

CUDA is Nvidia's platform for massively parallel high-performance computing on the Nvidia GPUs. At its core are three key abstractions: a hierarchy of thread groups, shared memories and barrier synchronisation. Full details on the execution environment, thread hierarchy, memory hierarchy and thread synchronisation schemes have been omitted here, but we refer the reader to Nvidia technical documentation[29,30] for a comprehensive discussion.

Specific computational environment

The CUDA algorithm and the respective testing tools were developed using Microsoft Visual Studio 2012 Professional, with the Nvidia Nsight extension for CUDA-C projects. The configurations shown in Table 1 were tested to determine the versatility of the CUDA clustering algorithms.

We had the opportunity to test two candidate graphics cards for the algorithm implementation: the Nvidia GTX Titan Black and the Nvidia TESLA C2050. Both cards offer double-precision calculations and a similar number of CUDA cores and TFLOPS (tera floating point operations per second); however, the GTX card is significantly cheaper than the TESLA card. The primary reason for this cost difference is the use of ECC (error check and correction) memory on the TESLA cards, where extra memory bits are present to detect and fix memory errors.[31] The presence of ECC memory ensures consistency in results generated from the TESLA card, which is critical for rigorous scientific computing. In further investigations, we will explore the consistency of the solution quality generated from the GTX card, and whether the resultant error is small enough to justify the cost saving compared to the TESLA card.

Implementation

The following objectives were considered in this study: (1) to investigate and tune the behaviour of the PGA implementation using a pre-defined set of 40 simulated stocks, featuring four distinct and disjoint clusters, (2) to identify clusters in a real-world data set, namely high-frequency price evolutions of stocks and (3) to test the efficiency of the GPU environment.

Representation

We used integer-based encoding for the representation of individuals in the GA:

$$\text{Individual} = S = \{s_1, s_2, \ldots, s_{i-1}, s_i, \ldots, s_N\} \qquad \text{Equation 9}$$

where $s_i = 1, \ldots, K$ and $i = 1, \ldots, N$. Here s_i is the cluster to which object i belongs. In terms of the terminology pertaining to GAs, the ith gene denotes the cluster to which the ith object or asset belongs. The numbers of objects or assets is N; thus to permit the possibility of an all-singleton configuration, we let $K=N$. This representation was implemented by Gebbie et al.[21] in their serial GA and was adopted in this research.

Fitness function

The Giada and Marsili maximum log-likelihood function L_c, as shown in Equation 8, was used as the fitness function. This function is used to determine whether the cluster configuration represents the inherent structure of the data set, i.e. it is used to detect if the GA converges to the fittest individual, corresponding to the cluster configuration which best explains the structure amongst correlated assets or objects in the data set.

Master-slave implementation

The unparallelised MATLAB GA implementation of the clustering technique by Gebbie et al.[21] served as a starting point. In order to maximise the performance of the GA, the application of genetic operators and evaluation of the fitness function were parallelised for the CUDA framework.[13] A summarised exposition is presented here.

Emphasis was placed on outsourcing as much of the GA execution to the GPU and using GPU memory as extensively as possible.[32] The master-slave PGA uses a single population, for which evaluation of the individuals and successive application of genetic operators are conducted in parallel. The global parallelisation model does not predicate anything about the underlying computer architecture, so it can be implemented efficiently on a shared-memory and distributed-memory model platform.[22] Delegating these tasks to the GPU and making extensive use of GPU memory minimises the data transfers between the host and device. These transfers have a significantly lower bandwidth than data transfers between shared or global memory and the kernel executing on the GPU. The algorithm in Gebbie et al.[21] was modified to maximise the performance of the master-slave PGA and to have a clear distinction between the master node (CPU) and slave nodes (GPU streaming multiprocessors). The CPU controls the evolutionary process by issuing the commands for the GA operations to be performed by

Table 1: Development, testing and benchmarking environments

Environment	Configuration	Framework
WINDOWS_GTX_CUDA	Windows 7 Professional Service Pack 1 (64-bit) Core i7-4770K CPU@3.50 GHz x 8, 32 GB RAM Nvidia GTX Titan Black with 6 GB RAM, CC: 3.0, SM: 3.5	CUDA-C 5.5 (parallel)
WINDOWS_GTX_MATLAB	Windows 7 Professional Service Pack 1 (64-bit) Core i7-4770K CPU@3.50 GHz x 8, 32 GB RAM Nvidia GTX Titan Black with 6 GB RAM, CC: 3.0, SM: 3.5	MATLAB 2013a (serial)
WINDOWS_TESLA_CUDA	Windows 7 Professional Service Pack 1 (64-bit) Intel Core i7-X980 CPU@3.33 GHz x 12, 24 GB RAM Nvidia TESLA C2050 with 2.5 GB RAM, CC: 2.0, SM: 2.0	CUDA-C 5.5 (parallel)
WINDOWS_TESLA_MATLAB	Windows 7 Professional Service Pack 1 (64-bit) Intel Core i7-X980 CPU@3.33 GHz x 12, 24 GB RAM Nvidia TESLA C2050 with 2.5 GB RAM, CC: 2.0, SM: 2.0	MATLAB 2013a (serial)

CPU, central processing unit; RAM, random access memory; CC, compute capability; SM, streaming multiprocessor.

the GPU streaming multiprocessors. The pseudo-code for the algorithm implemented is shown in Algorithm 1.

Algorithm 1: Master-slave parallel genetic algorithm for cluster identification

```
Initialise ecosystem for evolution
Size the thread blocks and grid to achieve greatest parallelisation
ON GPU: Create initial population
while TRUE do
    ON GPU: Evaluate fitness of all individuals
    ON GPU: Evaluate state and statistics
    ON GPU: Determine if termination criteria are met
    if YES then
        Terminate ALGO; exit while loop;
    else
        Continue
    end if
    ON GPU: Isolate fittest individuals
    ON GPU: Apply elitism
    ON GPU: Apply scaling
    ON GPU: Apply genetic operator: selection
    ON GPU: Apply genetic operator: crossover
    ON GPU: Apply genetic operator: mutation
    ON GPU: Apply replacement (new generation created)
end while
Report on results
Clean-up (de-allocate memory on GPU/CPU; release device)
```

GPU, graphics processing unit; CPU, central processing unit.

To achieve data parallelism and make use of the CUDA thread hierarchy, we mapped individual genes onto a two-dimensional grid. Using the representation shown in Equation 9, assuming a population of 400 individuals and 18 stocks:

$$Individual_1 = \{1,2,4,5,7...,6\}$$

$$Individual_2 = \{9,2,1,1,1...,2\}$$

$$Individual_3 = \{3,1,3,4,6...,2\}$$

$$\vdots$$

$$Individual_{400} = \{8,1,9,8,7...,3\}$$

would be mapped to grid cells, as illustrated in Figure 1. The data grid cells are mapped to threads, where each thread executes a kernel processing the data cell at the respective *xy*-coordinate.

Given the hardware used in this investigation (see Table 1), Table 2 outlines the restrictions on the permissible stock universe and population sizes imposed by the chosen mapping of individual genes to threads. A thread block dimension of 32 is chosen for larger problems, because this number ensures that the permissible population size is larger than the number of stocks to cluster.

We note that the efficiency of the algorithm may be compromised near the physical limits outlined in Table 2, as the CUDA memory hierarchy would force threads to access high-latency global memory banks more often. However, for the particular domain problem we are considering here, the Johannesburg Stock Exchange consists of about 400 listed companies on its main board, which represents an upper limit on the number of stocks of interest for local cluster analysis. This is well within the physical limits of the algorithm, while still providing scope to extend the application to multiple markets.

The details on the full implementation, as well as specific choices regarding initialisation, block sizes and threads per block, are given in Cieslakiewicz[13].

	Individual 1	**Individual 2**	**Individual 3**	**Individual 4**	**Individual 5**	**...**	**Individual 400**
	GRID						
	Block (0,0)	Block (1,0)	Block (2,0)	Block (3,0)	Block (4,0)	...	Block (399,0)
Stock 1	**1**	**9**	**3**	**2**	**5**		**8**
	Block (0,1)	Block (1,1)	Block (2,1)	Block (3,1)	Block (4,1)	...	Block (399,1)
Stock 2	**2**	**2**	**1**	**1**	**7**		**1**
	Block (0,2)	Block (1,2)	Block(2,2)	Block (3,2)	Block (4,2)	...	Block (399,2)
Stock 3	**4**	**1**	**3**	**3**	**3**		**9**
	Block (0,3)	Block (1, 3)	Block (2,3)	Block (3,3)	Block (4,3)	...	Block (399,3)
Stock 4	**5**	**1**	**4**	**3**	**4**		**8**
	Block (0,4)	Block (1,4)	Block (2,4)	Block (3,4)	Block (4,4)	...	Block (399,4)
Stock 5	**7**	**1**	**6**	**7**	**8**		**7**
⋮	⋮	⋮	⋮	⋮	⋮		⋮
	Block (0,17)	Block (1,17)	Block (2,17)	Block (3,17)	Block (4,17)	...	Block (399,17)
Stock 18	**6**	**2**	**2**	**1**	**2**		**3**

Figure 1: Mapping of individuals onto the Compute Unified Device Architecture (CUDA) thread hierarchy.

Table 2: Restrictions on the number of stocks and population size

Graphics card	Compute capability	Number of streaming multiprocessors	Maximum threads / thread block	Thread block dimension	Maximum thread blocks / multiprocessor	Maximum number of stocks	Maximum population size
Nvidia GTX Titan Black	3.5	15	1024	32	16	3840	17 472
Nvidia Tesla C2050	2.0	14	1024	32	8	3584	18 720

*For the Tesla card, the maximum number of stocks = (14) * (1024/32) * 8 = 3584 and maximum population size = (65 535 / (3584/32)) * 32 = 18 720.*

Key implementation challenges

A key challenge in CUDA programming is adapting to the single-program multiple-data (SPMD) paradigm, where multiple instances of a single program use unique offsets to manipulate portions of a block of data.[33] This architecture suits data parallelism, whereas task parallelism requires a special effort. In addition, because each warp (a group of 32 threads) is executed on a single SPMD processor, divergent threads in a warp can severely impact performance. In order to exploit all processing elements in the multi-processor, a single instruction is used to process data from each thread. However, if one thread needs to execute different instructions as a result of a conditional divergence, all other threads must effectively wait until the divergent thread re-joins them. Thus, divergence forces sequential thread execution, negating a large benefit provided by SPMD processing.

The CUDA memory hierarchy contains numerous shared memory banks which act as a common data cache for threads in a thread block. In order to achieve full throughput, each thread must access a distinct bank and avoid bank conflicts, which would result in additional memory requests and reduce efficiency. In our implementation, bank conflicts were avoided by using padding, in which shared memory is padded with an extra element such that neighbouring elements are stored in different banks.[34]

CUDA provides a simple and efficient mechanism for thread synchronisation within a thread block via the *__syncthreads()* barrier function; however, inter-block communication is not directly supported during the execution of a kernel. Given that the genetic operators can be applied only once the entire population fitness is calculated, it is necessary to synchronise thread blocks assigned to the fitness computation operation. We implemented the CPU implicit synchronisation scheme.[35,36] As kernel launches are asynchronous, successive kernel launches are pipelined and thus the executions are implicitly synchronised with the previous launch, with the exception of the first kernel launch. Given the latency incurred on calls between the CPU and GPU, and the consequent drag on performance, GPU synchronisation schemes were explored which achieve the required inter-block communication. In particular, GPU simple synchronisation, GPU tree-based synchronisation and GPU lock-free synchronisation were considered.[35]

Ultimately, the GPU synchronisation schemes were too restrictive for our particular problem, because the number of thread blocks would have an upper bound equal to the number of SMs on the GPU card. If the number of thread blocks is larger than the number of SMs on the card, execution may deadlock. This deadlock could be caused by the warp scheduling behaviour of the GPU, whereby active thread blocks resident on a SM may remain in a busy waiting state, waiting for unscheduled thread blocks to reach the synchronisation point. While this scheme may be more efficient for smaller problems, we chose the CPU synchronisation scheme in the interest of relative scalability.

Data pre-processing

To generate the *N*-stock correlation matrices to demonstrate the viability of the algorithm on real-world test data, data correlations were computed on data, where missing data was addressed using zero-order hold interpolation.[37] The market mode was removed using the method suggested by Giada and Marsili[17] using a recursive averaging algorithm. A covariance matrix was then computed using an iterative online exponentially weighted moving average filter with a default forgetting factor of $\lambda=0.98$. The correlation matrix was computed from the covariance matrix and was cleaned using random matrix theory methods. In particular, Gaussian noise effects were reduced by eliminating eigenvalues in the Wishart range in a trace-preserving manner.[37] This process enhanced the clusters and improved the stability of estimated sequence of correlation matrices.

Data post-processing

Computed cluster configurations are read from the CUDA output flat file. Successively, an adjacency matrix is constructed by using data values from the correlation matrix in conjunction with computed cluster configuration of the respective data set. The adjacency matrix is then used to construct a disjoint set of minimal spanning trees (MSTs), with each tree capturing the inter-connectedness of each cluster. Each MST exhibits $n_s - 1$ edges, connecting the n_s stocks of the cluster in such a manner that the sum of the weights of the edges is a minimum. Kruskal's algorithm was used to generate the MSTs, which depict the linkages between highly correlated stocks, providing a graphical visualisation of the resultant set of disjoint clusters.[38]

Data and results

Data

In this investigation we used two sets of data: the training set and the test set. The training set consisted of a simulated time series of 40 stocks which exhibit known distinct, disjoint clusters. The recovery of these induced clusters was used to tune the PGA parameters. The test set consisted of actual stock quoted midprice ticks aggregated into 3-min bars from 28 September 2012 to 10 October 2012, viz. approximately 1800 data points for each stock. Stocks chosen represent the 18 most liquid stocks on the Johannesburg Stock Exchange for that period, according to traded volumes. For both data sets, correlation matrices were constructed from the time series data to serve as inputs for the clustering algorithm. The test set results below show the summary statistics from a set of 1760 correlation matrices of 18 Johannesburg Stock Exchange stocks.

Results

We show a sample set of results here. Further discussion regarding aspects of the analysis are given in Cieslakiewicz[13].

Optimal algorithm settings

Various investigations were undertaken to identify optimal adjoint parameters for the PGA. In each case, the algorithm was successively applied to the training set, with known disjoint clusters. Settings were varied until the rate of convergence was maximised. Once the optimal value for each adjoint parameter had been determined from the training set, the optimal algorithm configuration was deployed on the test set. In further investigations, we will study the effect of various adjoint parameter choices on the rate of convergence and algorithm efficiency for varying stock universe sizes.

The optimal configuration shown in Table 3 for the PGA was deployed on the test set, given a population size of 1000.

Table 3: Optimal adjoint parameter values for given training set

Adjoint parameter	Value
Number of generations	400
Crossover probability (*Pc*)	0.9
Mutation probability (*Pm*)	0.1
Error tolerance	0.00001
Stall generations (*Gstall*)	50
Elite size	10
Crossover operator	Knowledge-based operator
Mutation operator	Random replacement
Knowledge-based crossover probability	0.9

Benchmark timing results

Table 4 illustrates the efficiency of the CUDA PGA implementation, compared to the MATLAB serial GA. Direct comparison between the MATLAB serial GA and CUDA PGA may be biased by the fundamental architecture differences of the two platforms. Nevertheless, we immediately observe a significant 10–15 times performance improvement for the test set cluster analysis run. This improvement can be attributed to the utilisation of a parallel computation platform, a novel genetic operator and the algorithm tuning techniques employed. On the GTX platform, the CUDA PGA takes 0.80 s to identify residual clusters inherent in a single correlation matrix of 18 real-world stocks, demonstrating its potential as a near-real-time risk assessment tool. The outperformance of the GTX card is likely explained by the card's relative faster core speed, memory speed, larger memory and memory bandwidth compared to the TESLA card. Although this may justify the use of the more cost-effective GTX card, it is not clear that this performance differential will persist as the size of the stock universe increases, or whether the GTX card preserves solution quality. We note that the scale of the performance improvement over the serial algorithm is not as important as the absolute result of obtaining sub-second computation time. The CUDA PGA thus serves the objective of near-real-time risk assessment, whereby interesting phenomena from emerging stock cluster behaviour can be identified and acted upon to mitigate adverse scenarios. The scalability of these results should be investigated in further research; in particular, the impact of the CUDA memory hierarchy on computation time as global memory accesses increase should be investigated.

In these results we assume correlation matrices are readily available as inputs for the cluster analysis algorithm. Further research to investigate computationally efficient correlation estimation for high-frequency data is a separate problem in the objective of developing a robust and practical near-real-time risk assessment tool.

Although the results are promising, it is not clear that the SPMD architecture used by CUDA is well suited for the particular problem considered. The required data dependence across thread blocks restricts the assignment of population genes to threads and results in a large number of synchronisation calls to ensure consistency of each generation. An MPI island model with distributed fitness computation and controlled migration is perhaps a more well-posed solution[39]; however, it is important to consider the cost of the set-up required to achieve the equivalent speed-up provided by CUDA. This cost should be explored in further research.

Interpretation of real-world test set results

In this section, we illustrate a sample of the resultant cluster configurations which were generated from our model, represented graphically as MSTs.[13,20] This illustration serves as a particular domain application which provides an example of resulting cluster configurations which have meaningful interpretations. The thickness of the edges connecting nodes gives an indication of the strength of the correlation between stocks.

The South African equity market is often characterised by diverging behaviour between financial/industrial stocks and resource stocks and strong coupling with global market trends.

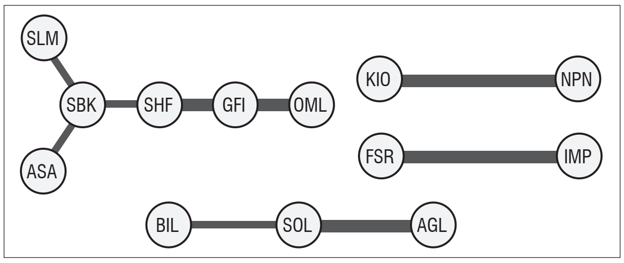

Figure 2: Morning trading residual clusters (on 28 September 2012 at 09:03).

In Figure 2, we see four distinct clusters emerge as a result of the early morning trading patterns, just after market open: most notably, a six-node financial/industrial cluster (SLM, SBK, ASA, SHF, GFI, OML) and a three-node resource cluster (BIL, SOL, AGL). At face value, these configurations would be expected; however, we notice that GFI, a gold mining company, appears in the financial cluster and FSR, a banking company, does not appear in the financial cluster. These examples are of short-term decoupling behaviour of individual stocks as a consequence of idiosyncratic factors.

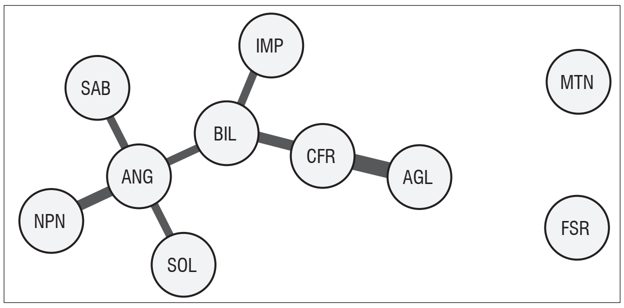

Figure 3: Morning trading (after UK market open) residual clusters (on 28 September 2012 at 10:21).

Table 4: Benchmark computational speed results

Environment	Framework	Benchmark	Median time (s)	Minimum time (s)	Maximum time (s)
WINDOWS_GTX_CUDA	CUDA-C 5.5	18-stock test set (optimal configuration)	0.80	0.73	3.17
WINDOWS_GTX_MATLAB	Serial	18-stock test set (optimal configuration)	7.77	6.72	13.27
WINDOWS_TESLA_CUDA	CUDA-C 5.5	18-stock test set (optimal configuration)	1.39	1.36	5.51
WINDOWS_TESLA_MATLAB	Serial	18-stock test set (optimal configuration)	15.91	13.41	26.22

Figure 3 illustrates the effect of the UK market open on local trading patterns. We see a clear emergence of a single large cluster, indicating that trading activity by UK investors has a significant impact on the local market. When examining the large single cluster, all of the stocks have either primary or secondary listings in the USA and UK. In particular, SAB and ANG have secondary listings on the London Stock Exchange, whereas BIL and AGL have primary listings on the London Stock Exchange.[40] It is also unusual to see such a strong link (correlation) between AGL, a mining company, and CFR, a luxury goods company. This correlation may be evidence that significant UK trading in these two stocks can cause a short-term elevated correlation, which may not be meaningful or sustainable.

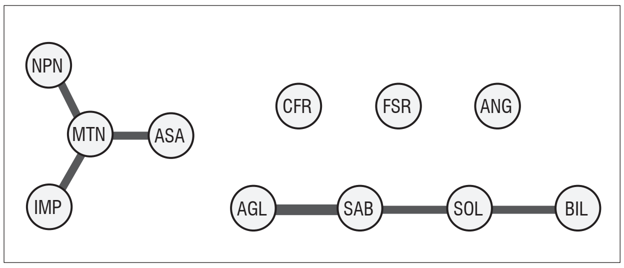

Figure 4: Midday trading residual clusters (on 28 September 2012 at 12:21).

In Figure 4, we consider midday trading patterns. We see that the clustering effect from UK trading has dissipated and multiple disjoint clusters have emerged. CFR has decoupled from AGL in the 2 h after the UK market open, as we might expect. We see a four-node financial/industrial cluster (NPN, MTN, ASA, IMP) and a four-node resource cluster (AGL, SAB, SOL, BIL); IMP, a mining company, appears in the financial/industrial cluster.

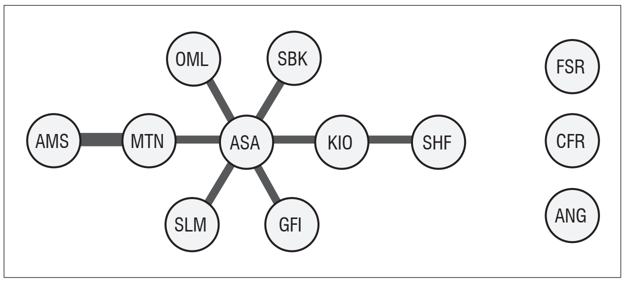

Figure 5: Afternoon trading (after US market open) residual clusters (on 28 September 2012 at 15:33).

Figure 5 illustrates the effect of the US market open on local trading patterns. Similar to what we observed in Figure 3, we see the emergence of a large single cluster, driven by elevated short-term correlations amongst constituent stocks. This observation provides further evidence that significant trading by foreign investors in local stocks can cause a material impact on stock market dynamics.

Conclusion

In this paper, we have verified that the Giada and Marsili[11] likelihood function is a viable, parallelisable approach for isolating residual clusters in data sets on a GPU platform. Key advantages of this function compared with conventional clustering methods are that: (1) the method is unsupervised and (2) the interpretation of results is transparent in terms of the model.

The implementation of the master-slave PGA showed that efficiency depends on various algorithm settings. The type of mutation operator utilised has a significant effect on the algorithm's efficiency to isolate the optimal solution in the search space, whilst the other adjoint parameter settings primarily impact the convergence rate. According to the benchmark test results, the CUDA PGA implementation runs 10–15 times faster than the serial GA implementation in MATLAB for detecting clusters in 18-stock real-world correlation matrices. Specifically, when using the Nvidia GTX Titan Black card, clusters are recovered in sub-second speed, demonstrating the efficiency of the algorithm.

Provided intraday correlation matrices can be estimated from high frequency data, this significantly reduced computation time suggests intraday cluster identification can be practical, for near-real-time risk assessment for financial practitioners.

Detecting cluster anomalies and measuring persistence of effects may provide financial practitioners with useful information to support local trading strategies. From the sample results shown, it is clear that intraday financial market evolution is dynamic, reflecting effects which are both exogenous and endogenous. The ability of the clustering algorithm to capture interpretable and meaningful characteristics of the system dynamics, and the generality of its construction, suggests the method can be successful in other domains.

Further investigations will include adjoint parameter tuning and performance scalability for varying stock universe sizes and cluster types, quantifying the variability of solution quality on the GTX architecture as a result of non-ECC memory usage and the investigation of alternative cost-effective parallelisation schemes. Given the SPMD architecture used by CUDA, the required data dependence across thread blocks restricts the assignment of population genes to threads and results in a large number of synchronisation calls to ensure consistency of each generation. An MPI island model with distributed fitness computation and controlled migration is perhaps a more well-posed solution to explore[39]; however, the cost of the set-up required to achieve the equivalent speed-up provided by CUDA should be justified.

Acknowledgements

This work is based on research supported in part by the National Research Foundation (NRF) of South Africa (grant numbers 87830, 74223 and 70643). The conclusions herein are those of the authors and the NRF accepts no liability in this regard. We thank Margit Haerting and Michelle Kuttel for their feedback on preceding projects culminating in this work.

Authors' contributions

D.W. and T.G. were responsible for the initial conception of the problem and idea development. T.G. assisted in data provision and cleaning, as well as in implementation of the serial algorithm. D.H. implemented the code, performed simulations, collated results and wrote the manuscript.

References

1. Luque G, Alba E. Parallel genetic algorithms. Berlin: Springer-Verlag; 2011. http://dx.doi.org/10.1007/978-3-642-22084-5_1

2. Colorni A, Dorigo M, Maffioli F, Maniezzo V, Righini G, Trubian M. Heuristics from nature for hard combinatorial optimization problems. Int Trans Oper Res. 1996;3:1–21. http://dx.doi.org/10.1111/j.1475-3995.1996.tb00032.x

3. Bohm C, Noll R, Plant C, Wackersreuther B. Density-based clustering using graphics processors. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. New York: ACM; 2009. p. 661–670. http://dx.doi.org/10.1145/1645953.1646038

4. Brecheisen S, Kriegel HP, Pfeifle M. Parallel density-based clustering of complex objects. In: Ng W-K, Kitsuregawa M, Li J, Chang K, editors. Advances in knowledge discovery and data mining. Proceedings of the 10th Pacific-Asia Conference; 2006 Apr 9–12; Singapore. Berlin: Springer; 2006. p. 179–188. http://dx.doi.org/10.1007/11731139_22

5. Dessel T, Anderson DP, Magdon-Ismail M, Newberg H, Szymanski BK, Varela CA. An analysis of massively distributed evolutionary algorithms. In: Proceedings of 2010 IEEE Congress on Evolutionary Computation; 2010 July 18–23; Barcelona, Spain. IEEE; 2010. p. 1–8. http://dx.doi.org/10.1109/CEC.2010.5586073

6. Jaimes A, Coello Coello C. MRMOGA: A new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. Concurrency Comput Pract Experience. 2007;19(4):397–441. http://dx.doi.org/10.1002/cpe.1107

7. Kromer P, Platos J, Snasel V. Data parallel density-based genetic clustering on the CUDA architecture. Concurrency Comput Pract Experience. 2014;26:1097–1112. http://dx.doi.org/10.1002/cpe.3054

8. Pospichal P, Jaros J, Schwarz J. Parallel genetic algorithm on the CUDA architecture. In: Di Chio C, Cagnoni S, Cotta C, Ebner M, Ekárt A, Esparcia-Alcazar AI, et al., editors. Applications of Evolutionary Computation EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC; 2010 Apr 7–9; Istanbul, Turkey. Berlin: Springer; 2010. p. 442–451. http://dx.doi.org/10.1007/978-3-642-12239-2_46

9. Robilliard D, Marion M, Fonlupt C. High performance genetic programming on GPU. In: Proceedings of the 2009 Workshop on Bio-inspired Algorithms for Distributed Systems. New York: ACM; 2009. p. 85–94. http://dx.doi.org/10.1145/1555284.1555299

10. Tirumalai V, Ricks K, Woodbury K. Using parallelization and hardware concurrency to improve the performance of a genetic algorithm. Concurrency Comput Pract Experience. 2007;19:443–462. http://dx.doi.org/10.1002/cpe.1113

11. Giada L, Marsili M. Algorithms of maximum likelihood data clustering with applications. Physica A. 2002;315:650–664. http://dx.doi.org/10.1016/S0378-4371(02)00974-3

12. Omran M, Salaman A, Engelbrecht A. Dynamic clustering using particle swarm optimization with application in image segmentation. Pattern Anal Appl. 2006;8:332–344. http://dx.doi.org/10.1007/s10044-005-0015-5

13. Cieslakiewicz D. Unsupervised asset cluster analysis implemented with parallel genetic algorithms on the Nvidia CUDA platform [thesis]. Johannesburg: University of the Witwatersrand; 2014.

14. Everitt B, Landau S, Leese M. Cluster analysis. 5th ed. Chichester: Wiley; 2001.

15. Blatt M, Wiseman S, Domany E. Superparamagnetic clustering of data. Phys Rev Lett. 1996;76:3251–3254. http://dx.doi.org/10.1103/PhysRevLett.76.3251

16. Kullmann L, Kertész J, Mantegna RN. Identification of clusters of companies in stock indices via Potts super-paramagnetic transitions. Physica A. 2000; 287(3–4):412–419. http://dx.doi.org/10.1016/S0378-4371(00)00380-0

17. Giada L, Marsili M. Data clustering and noise undressing of correlation matrices. Phys Rev E. 2001;63(6), Art. #061101, 8 pages. http://dx.doi.org/10.1103/PhysRevE.63.061101

18. Noh JD. A model for correlations in stock markets. Phys Rev E. 2000;61(5):5981–5982. http://dx.doi.org/10.1103/PhysRevE.61.5981

19. Sornette D. Critical phenomena in natural sciences: Chaos, fractals, selforganization and disorder: Concepts and tools. Berlin: Springer; 2000. http://dx.doi.org/10.1007/978-3-662-04174-1

20. Mbambiso B. Dissecting the South African equity markets into sectors and states [thesis]. Cape Town: University of Cape Town; 2009.

21. Gebbie T, Wilcox D, Mbambiso B. Spin, stochastic factor models, and a GA. PDF presented at: Southern African Finance Association conference; 2010. http://www.academia.edu/2499934/Spin_Stochastic_Factor_Models_and_a_GA.

22. Sivanandam S, Deepa S. Introduction to genetic algorithms. Berlin: Springer Science & Business Media; 2010.

23. Baker J. Reducing bias and inefficiency in the selection algorithm. In: Grefenstette JJ, editor. Proceedings of the Second International Conference on Genetic Algorithms and their Application. Hillsdale, NJ: L. Erlbaum Associates Inc.; 1987. p. 14–21.

24. Coley D. An introduction to genetic algorithms for scientists and engineers. Singapore: World Scientific Publishing; 1999. http://dx.doi.org/10.1142/3904

25. Ismail M. Parallel genetic algorithms (PGAs): Master-slave paradigm approach using MPI. IEEE e-Tech. 2004;31:83–87. http://dx.doi.org/10.1109/ETECH.2004.1353848

26. Digalakis J, Margaritis K. Parallel evolutionary algorithms on message-parsing clusters: Working paper [document on the Internet]. c2003 [cited 2015 May 12]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.2539&rep=rep1&type=pdf

27. Zhang J, Liu W, Liu G. Parallel genetic algorithm based on the MPI environment. Telkomnika. 2012;10:1708–1715. http://dx.doi.org/10.11591/telkomnika.v10i7.1566

28. Nan L, Pengdong G, Yongquan L, Wenhua Y. The implementation and comparison of two kinds of parallel genetic algorithm using Matlab. In: Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES); 2010 Aug. 10–12; Hong Kong, China. IEEE; 2010. p. 13–17. http://dx.doi.org/10.1109/DCABES.2010.9

29. Nvidia. Nvidia CUDA C programming guide. Santa Clara, CA: Nvidia Corporation; 2011.

30. Nvidia. CUDA dynamic parallelism programming guide. Santa Clara, CA: Nvidia Corporation; 2012.

31. ACT HPC. GTX vs Tesla [blog on the Internet]. c2014 [cited 2014 Sep 25]. Available from: http://www.advancedclustering.com/hpc-cluster-blog-gtx-vs-tesla

32. Zhang S, He Z. Implementation of parallel genetic algorithm based on CUDA. In: Cai Z, Li Z, Kang Z, Liu Y, editors. Advances in Computation and Intelligence. Proceedings of the 4th International Symposium on Advances in Computation and Intelligence; 2009 Oct 23–25; Huangshi, China. Berlin: Springer; 2009. p. 24–30. http://dx.doi.org/10.1007/978-3-642-04843-2_4

33. Darema F. SPMD model: Past, present and future. In: Cotronis Y, Dongarra J. Recent advances in parallel virtual machine and message passing interface: Proceedings of the 8th European PVM/MPI Users' Group Meeting; 2001 Sep 23–26; Santorini, Greece. Berlin: Springer; 2001. p. 1. http://dx.doi.org/10.1007/3-540-45417-9_1

34. Brodtkorb A, Hagen T, Saetra M. Graphics processing unit (GPU) programming strategies and trends in GPU computing. J Parallel Distr Com. 2012;73(1):4–13. http://dx.doi.org/10.1016/j.jpdc.2012.04.003

35. Xiao S, Feng W. Inter-block GPU communication via fast barrier synchronization. In: 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS); 2010 Apr 19–23; Atlanta, GA, USA. IEEE; 2010. p. 1–12. http://dx.doi.org/10.1109/IPDPS.2010.5470477

36. Nvidia. CUDA C best practices guide. Santa Clara, CA: Nvidia Corporation; 2012.

37. Wilcox D, Gebbie T. An analysis of cross-correlations in South African market data. Physica A. 2007;375:584–598. http://dx.doi.org/10.1016/j.physa.2006.10.030

38. Kruskal J. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc Amer Math Soc. 1956;7:48–50. http://dx.doi.org/10.1090/S0002-9939-1956-0078686-7

39. Whitley D, Rana S, Heckendorn R. The island model genetic algorithm: On separability, population size and convergence. J Comput Inform Technol. 1999;7:33–47.

40. JSE. Companies and financial instruments [homepage on the Internet]. c2013 [cited 2013 Sep 25]. Available from: https://www.jse.co.za/current-companies/companies-and-financial-instruments