Implementation of probabilistic forecasts in a GPU for big data problems.

Juan R. Trapero[a,*], Enrique Holgado de Frutos[a], Francisco Ramos[b]

[a] *University of Castilla-La Mancha, Department of Business Administration, 13071 Ciudad Real, Spain*
[b] *University of Castilla-La Mancha, Department of Electrical, Electronics, Control and Communications Engineering, 13071 Ciudad Real, Spain*

Abstract

High Performance Computing via General Purpose Graphical Processing Unit (GPU) is a potential instrument to speed up computational times. In a world where big data is becoming a revolution, GPU could play an important role. This work intends to analyze the performance of GPU by implementing the calculation of probabilistic forecasts based on single exponential smoothing in conjunction with simulated predictive distributions. Essentially, supply chain companies must deal with a high number of forecasts at SKU level. In this context, reducing the computational times can be a source of a competitive advantage. Since the forecasts are usually made independently between SKUs, this problem can be easily parallelized and GPU computing can exploit such parallelization. To the best of authors knowledge, this is the first time GPU is applied to a supply chain demand forecasting context. Firstly, we will show how to adapt the programming of probabilistic forecasts in a parallel fashion. Then, real data coming from a manufacturer company will be used to illustrate the differences between GPU and traditional CPU computing. The results show that GPU can significantly increase the computational speedup ratio more than 30 times with respect to traditional CPU computing.

Keywords: forecasting, GPU, big data, supply chain management

*Corresponding author.
Email addresses: juanramon.trapero@uclm.es (Juan R. Trapero), Enrique.Holgado@alu.uclm.es (Enrique Holgado de Frutos), francisco.ramos@uclm.es (Francisco Ramos)

1. Introduction

The world digitalization is generating a massive amount of data that can be obtained from different sources [1]. In a supply chain context those data sources are generated from enterprise resource planning (ERP) systems, distributed manufacturing environments, orders and shipment logistics, social media feeds, customer buying patterns, product lifecycle operations, and technology-driven data sources such as global positioning systems (GPS), radio frequency based identification (RFID) tracking, mobile devices and surveillance videos, among others [2, 3]. Dealing with such an amount and variety of information has challenged the traditional data analysis methods, [1, 4]. Therefore, organisations have to cope with big datasets characterized by 4Vs: large volume, velocity, variety, and value. Currently, the "Big Data" concept is starting to be defined in terms of the 5Vs model, which added the "Veracity" dimension (related to proper data governance and privacy concerns), [5, 6]. A recent International Data Corporation (IDC) forecast suggests that the Big Data technology will grow at a rate of 26.4% compound annual growth, reaching 41.5 billion through 2018, [2]. Essentially, the bigger the data, the more challenging it becomes to manage and analyse in order to deliver useful business insights. Manyika et al. [7] indicate that 300 billion dollars of potential annual value can be generated in US healthcare if organisations or governments can capture big data's value.

One of the main improvements that big data may bring is more accurate forecasts, which implies improving customer service levels, while lowering inventory costs, waste, and working capital [8]. Focusing on how the forecasts can be improved based on such big data, several articles indicate that predictive analytics is the key to reap the benefits of big data. It seems that predictive analytics comprises all the new techniques able to deal with big data that traditional forecasting methods cannot, [4]. Regarding the differences between predictive analytics and traditional forecasting methods, predictive analytics neglects statistical significance (given that we move from a small sample framework typical from the traditional statistical analysis to a big sample, even the entire population). Predictive analytics also acknowledges the importance of computational efficiency, [4]. Waller and Fawcett [9] classifies predictive analytics as a subset of data science and indicate that whereas forecasting is about predicting the future, predictive analytics adds

questions regarding what would have happened in the past, given different conditions. Although most of the research is focused on how Big Data Predictive Analytics (BDPA) improves corporate financial performance, it is also interesting to note that the impact of BDPA on other aspects of the triple botton-line (environmental and social sustainability) remains still as an open question [10].

Despite the acclaimed benefits of BDPA, there is a lack of empirical works that define at operational level how the required forecasts can be improved, or at least, how the traditional procedures should be changed to adapt them to the predictive analytics context. Nguyen et al. [11] carried out a literature review about big data analytics in supply chain management and concluded that only 3 papers out of 88 are centered on demand forecasting.

Nikolopoulos and Petropoulos [12] intend to shed some light into the problem of forecasting big data focusing on the optimization stage of the forecasting process. In that reference, the authors analyze the influence of the complex optimization routines on the forecasting performance. Their results show that less complex routines more oriented to face big data problems do not reduce significantly the forecasting accuracy, although it does improve the computation speed. Interestingly, they also note that the forecasting discipline has not developed too much progress in the incorporation of the Information and Communication Technologies for improving forecasting performance. The necessity of a compromise between computational speed and forecasting accuracy is also pointed out in a retail industry context in [13]. For instance, Seaman in [13] quantifies the forecasting problem for Walmart with upwards of a trillion forecasts being needed, calculating over 10 millions forecasts a second. In this sense, Seaman [13] indicates the "parallelizability" of the forecasting models used as a key factor to improve the speed of forecasting, as well as, the underlying computational infrastructure to support the parallelization. In the same sense, Zhong et al. [14] indicate that traditional serial algorithms, models, and mechanisms are inefficient for Big Data and new data parallelism and advanced approaches are a current challenge.

Demand planning and inventory management requires probabilistic forecasts to optimize supply chain decisions. For instance, safety stocks and reorder points in order up to level replenishment policies are based on those probabilistic forecasts. Here, probabilistic forecasts refers to forecasts of a certain quantile, i.e., a forecast of the demand mean plus a forecast of the demand variability. In general terms, the variability is a measure of demand

uncertainty. Traditionally, these estimations are done via serial computing.

Although probabilistic forecasts can be based on theoretical models [15], where it is assumed that the underlying demand process is correctly captured by the forecasting model, the uncertainty in the parameter estimates are usually neglected [16]. A possible solution to overcome that limitation is to use simulated prediction distributions [17, p. 77]. In fact, for some complex models, simulation is the only method available to compute predictive distributions [17, p. 79]. However, the main inconvenient is the computational burden.

This work aims at exploring the use of GPU to improve the computational speed of calculating probabilistic forecasts via simulated prediction distributions. Essentially, we show how the forecasting method should be implemented in a parallel fashion to take advantage of the potential benefits of GPU. We particularize our results for a well-known method as single exponential smoothing for point forecasting and simulated prediction distributions through Monte Carlo experiments for variability forecasts. These techniques will be implemented in MATLAB. In this work, we intend to implement the algorithms with minimum changes with respect to traditional MATLAB programming, i.e., without programming in CUDA (Compute Unified Device Architecture). Following that philosophy, we will use some "built-in" functions already available in MATLAB that supports GPU computing, minimizing the programming effort. Real data coming from a manufacturer will be used to illustrate the methodology.

This article is organised as follows: Section 2 explains the main differences of a GPU with respect to traditional CPU. Section 3 describes the exponential smoothing forecasting method expressed as a transfer function and how the smoothing parameter is determined via the grid and search optimization routine. Additionally, in the same section, the simulated probabilistic forecast methodology will be explained. Section 4 shows the evaluation metrics in terms of forecasting accuracy and computational speed. Section 5 is devoted to test those algorithms with real data coming from a manufacturer shipments data. Finally, section 6 summarizes the main conclusions.

2. What can GPUs do for probabilistic demand forecasting?

GPUs were designed for graphical representation purposes characterized by tedious and repetitive, although not complicated, calculations with thousands of millions of them per second. The idea was easy: we divide the

desired image processing in independent units of work (e.g. dividing the whole image in small chunks), separately calculate each of these units of work, and then obtain the final output by composing the results of all units of work.

The introduction of CUDA, which is a general purpose parallel computing architecture that is more user-friendly and mature, has made GPU computing to evolve towards the General Purpose GPU. In general terms, GPUs are recommended when the following criteria are met: i) computationally extensive: the time spent in calculations should highly exceed the time spent on transferring data to and from GPU memory, as memory access becomes a bottleneck when using GPUs, i.e., data must be sent from CPU to GPU before calculations and returned afterwards to main memory. As the communication between CPU and GPU is typically performed through an AGP or PCI Express bus, memory access is slower than with the usual CPU architecture; ii) massively parallel: the calculations must be divisible in hundreds/thousands of independent units of work. These calculations must be the same for every chunk of data that is to be processed in a GPU core at the same time.

Demand forecasting fits both criteria perfectly and, hence, can greatly benefit from the use of GPUS. On the one hand, forecasting algorithms can be individually applied to each SKU, as they are (mainly) independent to each other, what makes them easily parallelizable. On the other hand, each algorithm relies on a Monte Carlo experiments with (many) replications, being computationally demanding. Each of these replications could as well be performed in a different core. Therefore, parallel computing is expected to significantly reduce the computation time, what might become a competitive advantage in a big data driven world.

However, parallel programming is a different programming paradigm from traditional structured programming or object-oriented programming used for most applications. Some libraries, such as CUDA[1] (Computer Unified Device Architecture), were developed to help in the process of shifting this paradigm for a variety of applications, not only graphics processing. Thankfully, some mathematical frameworks, such as MATLAB, provide their own built-in functions to leverage the change of paradigm with slight changes respect to traditional programming.

[1]For the interested reader see `https://developer.nvidia.com/cuda-toolkit`

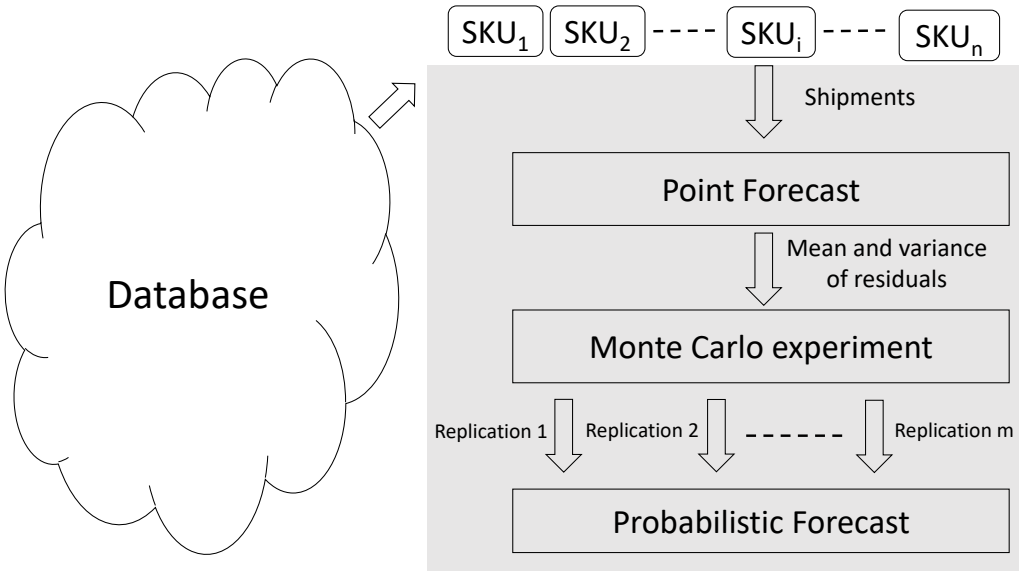

Figure 1: Scheme of probabilistic forecasts calculation.

Currently, GPU is becoming more popular in large-scale, data mining [18], dynamic programming [19, 20], graphs problems [21], discrete location theory [22], genetic algorithms [23], although its application to predictive analytics problems is still scarce [24].

3. Probabilistic Forecasts

In this section, we will explain how to compute the probabilistic forecasts following the scheme showed in Figure 1, which is divided in three stages. First, the shipments information for n SKUs is obtained from the company database, and for each SKU, the point forecast is computed. Second, with the point forecasts, the mean and variance of residuals are computed and fed into the Monte Carlo experiment which generates m replications. Third, the calculation of the probability forecasts is obtained based on those replications. Below, we describe in detail each stage.

3.1. Point forecasting

3.1.1. Single exponential smoothing (SES)

First, we need to calculate the point forecast using SKU shipments as an input, where the point forecasting algorithm yields a measure of the central tendency of the forecast density function. Among the possible forecasting methods, Single Exponential Smoothing (SES) has been chosen for two reasons. Firstly, SES is widely used in business applications [25, 26]. Secondly, it is consistent with common practice, in which a company may not be using the optimal forecasting model due to either a lack of expertise or the availability of only a limited set of forecasting models in their software, [27]. Such a forecasting method can be formulated as, [28]

$$F_{t+h}^{h} = \alpha y_t + (1-\alpha) F_t^h, \tag{1}$$

where $0 \leq \alpha \leq 1$ is the smoothing constant, y_t is the actual value and F_t^h is the h-step ahead forecast, both at time t. h is the forecasting horizon. Given the recursive nature of exponential smoothing, it is necessary to initialize the algorithm. We initialize with $F_1^h = y_1$, which is a simple approach because when the data sample is big, as we expect in a big data framework, the influence of the initialization is minor [28].

In this work, we will use the transfer function form of SES since it is easier to allow its parallelization. Recall that SES expressed as a transfer function [29] is:

$$TF_{SES} = Z\left(\frac{F_{t+h}^{h}}{y_t}\right) = \frac{\alpha}{1-(1-\alpha)Z^{-1}} \tag{2}$$

where Z is the Z-transform, such as $Z^{-1}Y_t = Y_{t-1}$.

3.1.2. Optimization. Grid and search vs. fmincon

The α value in (1) is optimized by using a grid and search algorithm that is also known as parameter sweep or exhaustive search [12]. This algorithm starts a search for values of α ranging from 0 to 1. The parameter that controls the routine is the number of steps (n_a) that we divide the whole interval, where the updating step is $s = \frac{1-0}{n_a}$. For each value of α, the Mean Squared Error (MSE) is calculated as follows:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(y_t - F_t^1)^2 \tag{3}$$

where N is the sample size used for training. The chosen α is the one that minimizes one-step ahead MSE over the training set. Authors in [12] show that this simple method is able to provide a similar forecast accuracy than other sophisticated alternatives with a considerable reduction of computational time.

The GPU optimizes the exponential smoothing parameter by applying grid and search routine previously introduced. To compute the MSE, we need to apply SES for each value of α. This can be done with the "built-in" MATLAB command **filter**, [30], using the SES on the hold-in sample expressed as a transfer function, see (2). As a result of this stage, the optimal α value is obtained.

Note that this optimization can be done either serially or in parallel. Procedure 1 describes the algorithm to obtain the exponential smoothing parameter in serial computing. In turn, Procedure 2 shows how to obtain the same value in a parallel manner. Essentially, by computing in parallel we have removed the outer loop with regards to the serial version. Note that, such an outer loop is bigger than the inner loop because $n >> n_a$

Procedure 1 Grid and search optimization (serial)

1: **for** $i = 1$ **to** n **do**
2: Select sales of SKU i ($y_{i,t}$)
3: **for** $j = 1$ **to** n_a **do**
4: Compute point forecasts of $y_{i,t}$ with SES for a determined α_{ij}
5: Compute MSE_j over the training set
6: **end for**
7: Determine $\alpha_i = \alpha_j$ such as α_j minimizes MSE_j over the train set
8: **end for**

Procedure 2 Grid and search optimization (parallel)

1: **for** $j = 1$ **to** n_a **do**
2: Compute point forecasts with SES for a determined α_j for all SKUs in parallel
3: Compute MSE_j over the training set for all SKUs in parallel
4: **end for**
5: Determine $\alpha_i = \alpha_j$ such as α_j minimizes MSE_j over the train set for all SKUs.

3.2. Monte Carlo experiment

Apart from the point forecasts, another output available is the residuals in the test set, i.e., the difference between forecasts and actual values for a determined SKU. Note that we use the residuals in the test set instead of the training set to avoid underestimating the magnitude of the residuals variance.

The Monte Carlo replications are $x^h_{ij,t}$, with $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$, where n and m are the number of SKUs and replications, respectively. To compute such replications mean and variance of residuals on the test set are calculated and used for generating Gaussian random numbers $(\epsilon^h_{ij,t})$ that are added to the h-step ahead point forecast $(F^h_{i,t})$ for a determined SKU i, such as:

$$x^h_{ij,t} = F^h_{i,t} + \epsilon^h_{ij,t} \tag{4}$$

where m is the number of replications in the Monte Carlo experiment and n is the total number of SKUs.

According to MATLAB 2018b release, the fastest algorithm to generate random normal $\epsilon^h_{ij,t}$ is Philox 4x32 generator with 10 rounds, and for that reason, it is the one chosen in this study. This generator was introduced in 2011, and it was specifically designed for high performance in highly parallel systems such as GPUs [30]. In this study we have used a normal random number generator because other alternatives as using either different statistical distributions or resampling methods as bootstrapping do not have available a built-in function in MATLAB, and thus, they cannot be used directly on a GPU.

3.3. Probabilistic forecast

Finally, the m forecast replications $(x^h_{ij,t})$ for a determined forecasting horizon h can be used to build a predictive distribution of a certain SKU i. Considering that distribution and a target quantile, the MATLAB command **prctile** provides the empirical quantile of interest. In a supply chain context, the quantile of interest is given by the Cycle Service Level (CSL) ([31]), which represents the asymmetry between stock out and holding costs.

In summary, the pseudocode of the methodology presented here is shown in procedures 3 and 4, where procedure 3 explains the calculation of probabilistic forecasts from a serial computing point of view typically used in CPU, whereas procedure 4 shows its parallel implementation thought for a GPU. Note that the parallel version has removed the inner loop required to generate m Monte Carlo replications.

Procedure 3 Calculation of probabilistic forecasts (CPU serial computing)

for $i = 1$ **to** n **do**
  Optimize in serial SES parameter α for the i SKU on the training set (see Procedure 1)
  Compute h-step ahead point forecasts $F_{i,t}^h$ with the parameter optimized in 2
  Compute residuals on the test set: $e_{i,t}^h = F_{i,t}^h - y_{i,t}$
  for $j = 1$ **to** m **do**
    Generate Gaussian random numbers $\epsilon_{ij,t}^h$ from mean and variance of residuals $e_{i,t}^h$
    Build j replication: $x_{ij,t}^h = F_{i,t}^h + \epsilon_{ij,t}^h$
  end for
  Compute h-step ahead probabilistic forecast based on m replications for a given CSL and for a certain SKU i
end for

4. Evaluation metrics

In this work, we evaluate two key aspects in forecasting: accuracy and computational speed. The accuracy is measured by an overall metric as the Average Relative Mean Absolute Error (AvgRelMAE) [32]. The AvgRelMAE can be computed as:

$$AvgRelMAE = \left(\prod_{i=1}^{n} r_i \right)^{1/n} \tag{5}$$

where $r_i = \frac{MAE_i^{SES}}{MAE_i^N}$. MAE_i^{SES} is the Mean Absolute Error (MAE) produced by SES on the test set for SKU i; and MAE_i^N is the MAE of the Naïve method that is used as a benchmark for the same SKU i also in the same test set. n is the total number of SKUs. Note that, the Naïve forecasting method uses the last observation as a forecast for the next period. Thus, if AvgRelMAE is lower than 1, it means that SES is more accurate than Naïve.

The differences in computational speed can be easily measured with the speedup ratio, [22]. Speedups are computed as CPU time required to do the calculations divided by GPU time counterpart, such as: $speedup = CPU_{time}/GPU_{time}$.

Procedure 4 Calculation of probabilistic forecasts (GPU parallel computing)

1: Optimize in parallel SES parameter α for the all SKUs (see Procedure 2)
2: **for** $i = 1$ **to** n **do**
3: Compute h-step ahead point forecasts $F_{i,t}^h$ with the parameter optimized in 2
4: Compute residuals on the test set: $e_{i,t}^h = F_{i,t}^h - y_{i,t}$
5: Generate m Gaussian random numbers ϵ_{ij,t^h} from mean and variance of residuals $e_{i,t}^h$, where $j = 1, \ldots, m$
6: Build m replications: $x_{ij,t} = F_{i,t}^h + \epsilon_{ij,t}^h$
7: **end for**
8: Compute h-step ahead probabilistic forecast based on m replications for a given CSL and for all SKUs

5. Experimental results

In this section we compare the computational time taken to compute probabilistic forecasts for a real dataset that belongs to an international company based on UK devoted to provide personal care products. The dataset has been studied in detail by Barrow and Kourentzes [33]. It comprises 173 SKUs, with 229 observations per SKU. Manufacturer shipments are used to estimate the demand, which is mainly continuous, that is, there are not many zeros in the data. The dataset has been divided in two parts. The hold-in sample or training set that consists of 80% of the dataset (183 observations) and the hold-out sample or test set based on the last part of the data (46 observations). The optimization is carried out in the training set and the AvgRelMAE is assessed in the test set for all the SKUs considered. The forecasting horizon is one step ahead ($h = 1$).

These experiments will be run in a machine with two Intel (R) Xeon processors at 3.00 GHz with 128 GB of RAM. The GPU card is a Quadro GP100, 16 GB of Memory and 3584 CUDA Parallel-Processing Cores.

5.1. Analysis of accuracy

The calculation of the probabilistic forecasts requires the use of the grid and search optimization routine, whose performance depends on the grid size parameter (n_a), which should be determined.

This section analyzes the influence of n_a on the forecasting accuracy perspective. Figure 2 shows the AvgRelMAE provided by the forecasting method SES for all the 173 SKUs and different forecasting horizons $h = 1, 3, 5$. In that graph, it can be observed the forecasting accuracy obtained by the considered optimization methods. On the one hand, the typical optimization function of MATLAB (**fmincon** that finds minimum of constrained nonlinear multivariable function, [30]) in a dashed line and, on the other hand, the grid and search routine, which was previously introduced, in a solid line. That figure shows that the required n_a that achieves the same forecasting accuracy than **fmincon** depends on the forecasting horizon. For instance, when $h = 1$ less than 5 steps are needed to obtain the same forecasting accuracy than **fmincon**. For horizons greater than 1, a grid size (n_a)of 20 offers a similar accuracy to the one obtained by the **fmincon** function. These results agree with those obtained by Nikolopoulos and Petropoulos [12] for a different dataset, where small grid sizes yield similar forecasting accuracies with respect to more complex optimization routines. Therefore, for the experiments in this study, the grid size will be set to a conservative $n_a = 20$.

5.2. Analysis of computational times

Figure 3 depicts an example of shipments and forecasts for a certain SKU. The upper panel of that figure shows the shipments data for that SKU. The lower panel shows the actual shipments and forecasts for the test set. The forecast errors in the test set are used to obtain the mean and standard deviation that will be fed to the random normal number generator to build the m Monte Carlo replications.

In our dataset we have a total of 173 SKUs sales data. Out interest is to analyze the computational times for a bigger number of SKUs. To do that, we have resampled the original dataset to obtain as many SKUs sales data as we need on the basis of the original dataset.

Table 1 shows the speedup ratios calculated for different volumes of SKUs from 100 up to 5000. The first row shows the speed up ratio of the MATLAB function **fmincon** run on a CPU with respect to the grid and search ($n_a = 20$) run in a GPU. Such a ratio varies between 32 and 42, that is, the GPU option was more than 30x times faster than the CPU alternative. The second and third row computes the speed up ratio using the same optimization technique (grid and search) with different values of n_a. For values of $n > 100$ a lower value of n_a provides higher speed up ratios.

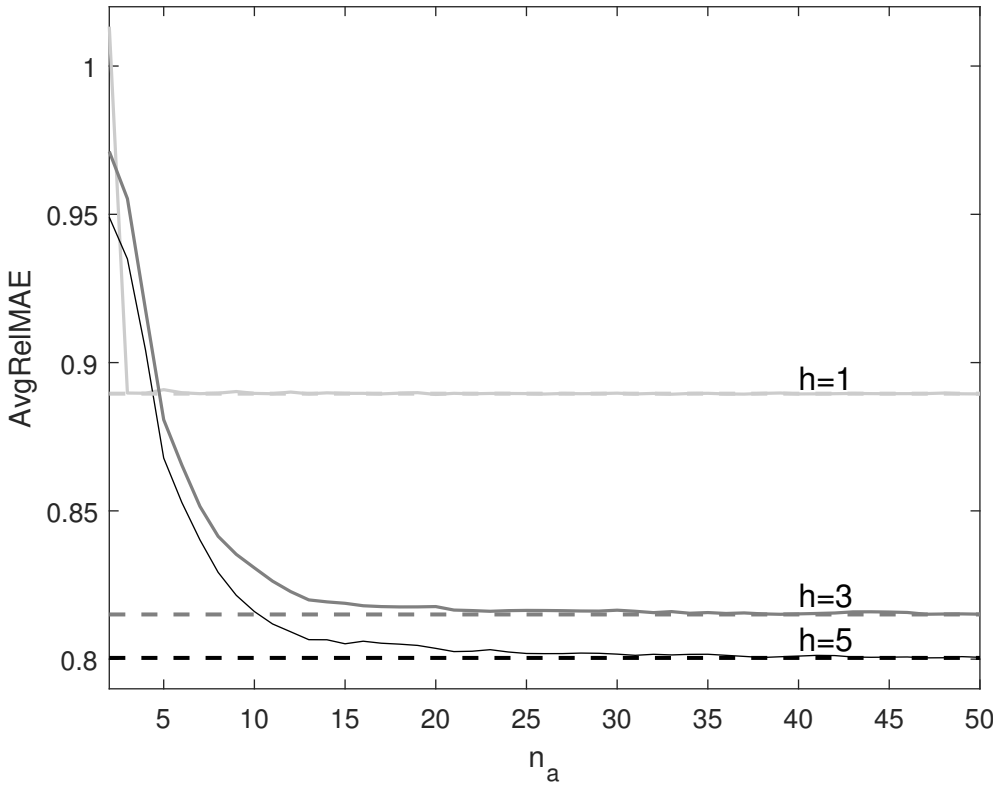

Figure 2: AvgRelMAE versus grid size (n_a) obtained for both fmincon (dashed line) and grid and search (solid line) optimization for all SKUs.

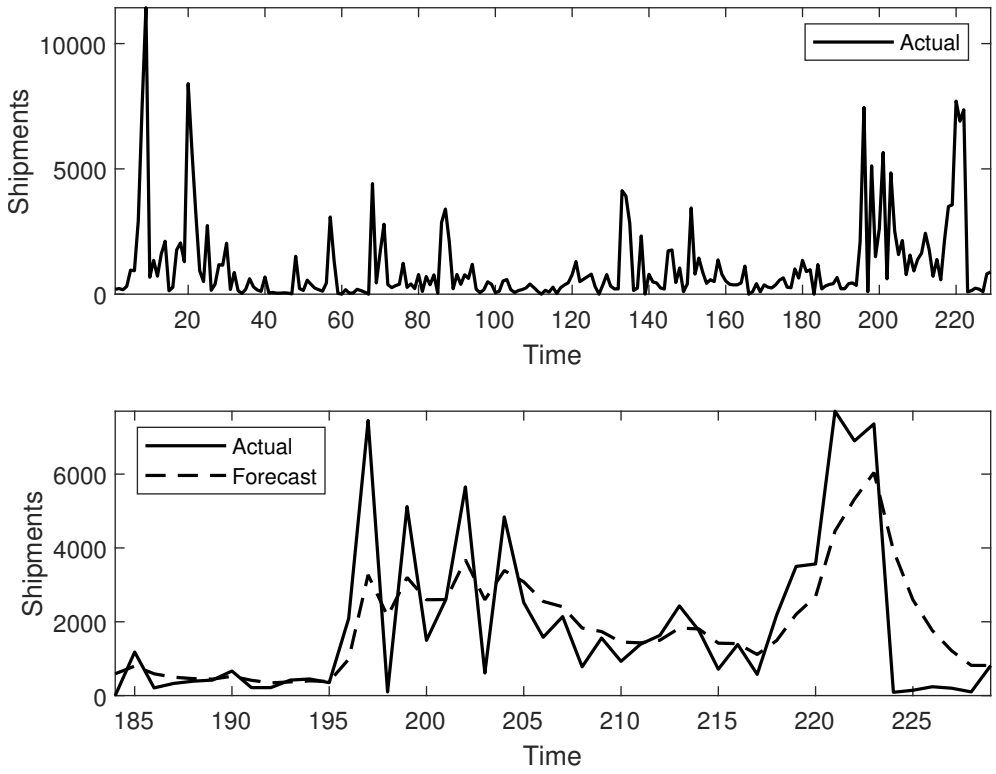

Figure 3: Example of the shipments for a certain SKU. Upper panel shows all the observations. Lower panel shows the forecasts (dashed line) and actual observations (solid line) for the test set.

It is interesting to note that we there is a value of n that maximizes the speed up ratio and it is $n = 1000$. That is an interesting fact, since, a priori, one could wrongly think that the higher the value of n, the higher the speed up ratio of the GPU.

This is an important conclusion that can be utilized when scaling this problem. Essentially, for a big data framework where the number of SKUs may compromise the memory size, MATLAB allows to work with different GPU cards (in parallel[2]) and this experiment shed some light about how many SKUS should be distributed in a GPUs cluster to obtain the maximum speed up ratio, i.e., the lower computational times.

Speed up	n			
	100	1000	2500	5000
fmincon	32	42	40	38
grid and search (n_a=40)	28	39	37	34
grid and search (n_a=20)	29	41	40	36

Table 1: Speed up ratios calculated for different values of n and rounded to the nearest integer.

Figure 4 shows the computational time required to calculate the probabilistic forecasts when using the CPU and GPU. The results show that the GPU with a grid and search optimization is faster than the CPU, regardless of the optimization routine chosen. Moreover, that difference is bigger when the number of SKUs (n) increases. In GPU the optimization routine was grid and search.

6. Conclusions

This article presents a methodology to implement in a GPU the calculation of probabilistic forecasts in a parallel manner, where the point forecasting algorithm is a widely used forecasting technique as SES and the quantile forecasts can be obtained empirically via simulated predictive distributions. This simulation approach to compute qunatile forecasts has two main advantages: i) it incorporates the uncertainty of the parameter estimations that is usually neglected and; ii) for some complex forecasting models is the only

[2]Recent versions of MATLAB permit to use the **parfor** command on GPU cards

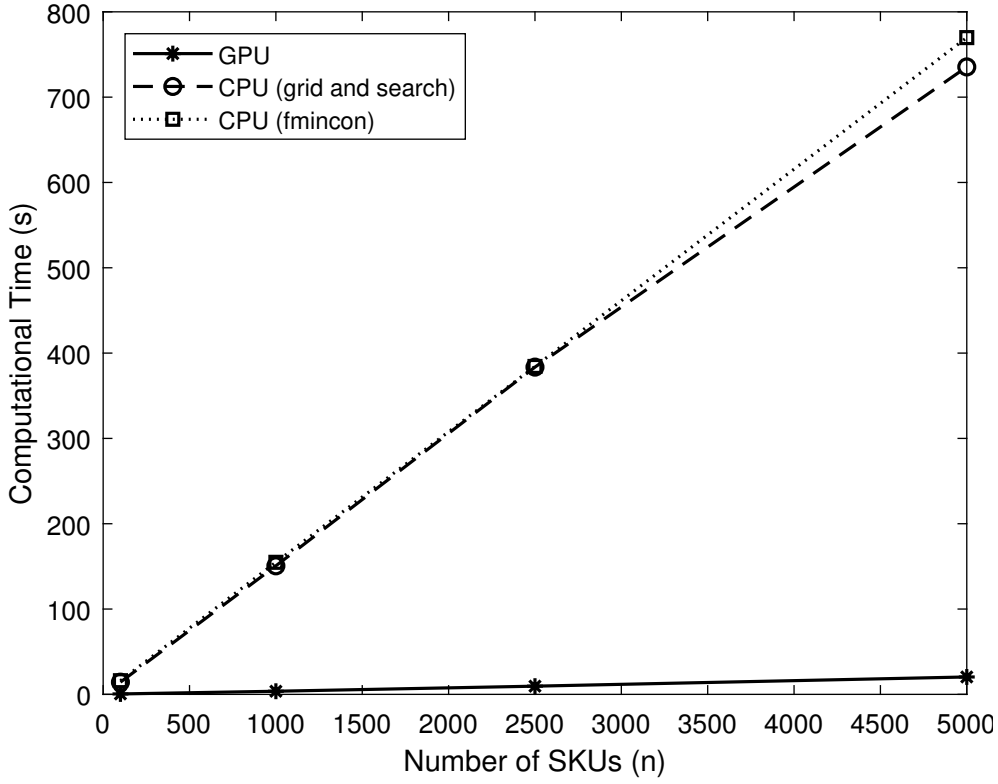

Figure 4: Computational time required for calculating the probabilistic forecast for different number of SKUs (n).

approach to compute predictive distributions. However, its main limitation is the computational burden associated.

In this work, such a limitation is alleviated through the use of GPU computing in parallel implementations of the algorithms involved. This methodology is implemented in MATLAB using "built-in" functions to circumvent the use of CUDA language. Thus, simply usual MATLAB coding is necessary to apply the proposed approach.

To the best of authors' knowledge, this is the first time that GPU computing is used for the problem of supply chain forecasting. The main result show that GPU computing obtains similar forecasting accuracy in a reduced computational time.

The improvements of GPU over CPU is limited for the GPU card memory. Apart from expecting future GPU cards with bigger memories, a possibility to relax that limitation in big data applications is to use a cluster of GPU cards. Recent MATLAB 2018b version is able to use several GPUs in parallel too. Additionally, in this work, we have shown that there is an optimal data size to potentially distributing among GPU cards in a cluster, if it is necessary to scale the problem.

Although SES is a widely used forecasting method, when the data exhibits trend and/or seasonality, more advances exponential smoothing models should be used. Further research should address the application of the proposed procedure with other exponential smoothing forecasting models. The only limitation to apply the procedure proposed here is that the forecasting model must be able to be represented as a filter. For instance, ARIMA models fit perfectly and can also be used.

Acknowledgment

This work was supported by the European Regional Development Fund and Spanish Government (MINECO/FEDER, UE) under the project with reference DPI2015-64133-R. Juan R. Trapero also thanks Raul Fernandez Rodriguez for his help with the GPU installation.

Bibliography

[1] D. Blazquez, J. Domenech, Big data sources and methods for social and economic analyses, Technological Forecasting and Social Change 130 (2018) 99 – 113.

[2] K. Govindan, T. Cheng, N. Mishra, N. Shukla, Big data analytics and application for logistics and supply chain management, Transportation Research Part E: Logistics and Transportation Review (2018).

[3] S. Tiwari, H. Wee, Y. Daryanto, Big data analytics in supply chain management between 2010 and 2016: Insights to industries, Computers & Industrial Engineering 115 (2018) 319 – 330.

[4] A. Gandomi, M. Haider, Beyond the hype: Big data concepts, methods, and analytics, International Journal of Information Management 35 (2015) 137 – 144.

[5] G. Bello-Orgaz, J. J. Jung, D. Camacho, Social big data: Recent achievements and new challenges, Information Fusion 28 (2016) 45 – 59.

[6] R. Addo-Tenkorang, P. T. Helo, Big data applications in operations/supply-chain management: A literature review, Computers & Industrial Engineering 101 (2016) 528 – 543.

[7] J. Manyika, M. Chui, P. Groves, D. Farrell, S. Kuiken, E. Doshi, Open data: unlocking innovation and performance with liquid information, McKinsey Global Institute. Available on-line., 2013.

[8] C. W. Chase, Using big data to enhance demand-driven forecasting and planning, The Journal of Business Forecasting 32 (2013) 27–32.

[9] A. M. Waller, E. S. Fawcett, Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management, Journal of Business Logistics 34 (2013) 77–84.

[10] B. T. Hazen, J. B. Skipper, J. D. Ezell, C. A. Boone, Big data and predictive analytics for supply chain sustainability: A theory-driven research agenda, Computers & Industrial Engineering 101 (2016) 592 – 598.

[11] T. Nguyen, L. Zhou, V. Spiegler, P. Ieromonachou, Y. Lin, Big data analytics in supply chain management: A state-of-the-art literature review, Computers & Operations Research 98 (2018) 254 – 264.

[12] K. Nikolopoulos, F. Petropoulos, Forecasting for big data: Does suboptimality matter?, Computers & Operations Research (2017).

[13] B. Seaman, Considerations of a retail forecasting practitioner, International Journal of Forecasting (2018).

[14] R. Y. Zhong, S. T. Newman, G. Q. Huang, S. Lan, Big data for supply chain management in the service and manufacturing sectors: Challenges, opportunities, and future perspectives, Computers & Industrial Engineering 101 (2016) 572 – 591.

[15] J. R. Trapero, M. Cards, N. Kourentzes, Empirical safety stock estimation based on kernel and GARCH models, Omega 84 (2019) 199 – 211.

[16] D. Prak, R. Teunter, A general method for addressing forecasting uncertainty in inventory models, International Journal of Forecasting 35 (2019) 224 – 238. Special Section: Supply Chain Forecasting.

[17] R. J. Hyndman, A. B. Koehler, J. K. Ord, R. D. Snyder, Forecasting with Exponential Smoothing: The State Space Approach, Springer-Verlag, Berlin, 2008.

[18] C. H. Nadungodage, Y. Xia, J. J. Lee, M. Lee, C. S. Park, GPU accelerated item-based collaborative filtering for big-data applications, in: 2013 IEEE International Conference on Big Data, pp. 175–180.

[19] V. Boyer, D. E. Baz, M. Elkihel, Solving knapsack problems on GPU, Computers & Operations Research 39 (2012) 42 – 47. Special Issue on Knapsack Problems and Applications.

[20] V. Curtis, C. Sanches, A low-space algorithm for the subset-sum problem on GPU, Computers & Operations Research 83 (2017) 120 – 124.

[21] B. Nogueira, R. G. Pinheiro, A CPU-GPU local search heuristic for the maximum weight clique problem on massive graphs, Computers & Operations Research 90 (2018) 232 – 248.

[22] G. J. Lim, L. Ma, GPU-based parallel vertex substitution algorithm for the p-median problem, Computers & Industrial Engineering 64 (2013) 381 – 388.

[23] J. R. Cheng, M. Gen, Accelerating genetic algorithms with GPU computing: A selective overview, Computers & Industrial Engineering 128 (2019) 514 – 525.

[24] I. M. Coelho, V. N. Coelho, E. J. da S. Luz, L. S. Ochi, F. G. Guimares, E. Rios, A GPU deep learning metaheuristic based model for time series forecasting, Applied Energy 201 (2017) 412 – 418.

[25] E. S. Gardner, Exponential smooothing: The state of the art, Journal of Forecasting 4 (1985) 1–28.

[26] E. S. Gardner, Exponential smoothing: The state of the art, Part II, International Journal of Forecasting 22 (2006) 637–666.

[27] R. Fildes, Research into Forecasting Practice, Foresight: The International Journal of Applied Forecasting (2017) 39–46.

[28] J. K. Ord, R. Fildes, N. Kourentzes, Principles of Business Forecasting, Wessex Press Publishing Co., 2nd edition, 2017.

[29] S. M. Disney, D. R. Towill, On the bullwhip and inventory variance produced by an ordering policy, Omega 31 (2003) 157 – 167.

[30] MATLAB, Release 2018a, The MathWorks Inc., Natick, Massachusetts, 2018.

[31] E. Silver, D. Pyke, D. Thomas, Inventory and Production Management in Supply Chains. Fourth Edition, CRC Press. Taylor and Francis Group, 2017.

[32] A. Davydenko, R. Fildes, Measuring forecasting accuracy: The case of judgmental adjustments to sku-level demand forecasts, International Journal of Forecasting 29 (2013) 510 – 522.

[33] D. Barrow, N. Kourentzes, Distributions of forecasting errors of forecast combinations: implications for inventory management, International Journal of Production Economics 177 (2016) 24–33.