FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Improving electrical production forecasting using GPU

António dos Santos Monteiro Borges Pires

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui PedroAmaral Rodrigues (PhD)

July 17th, 2013

Improving electrical production forecasting using GPU

António dos Santos Monteiro Borges Pires

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

July 17th, 2013

Abstract

Wind power penetration in worldwide energy markets has been rapidly growing over the last few years. Therefore, and due to the integration restrictions of this type of energy, the need for short-term wind power output predictions has been rapidly increasing.

Ever more the reliability of the forecast is a factor to consider due to the growing economic impact that its errors can cause. However, the main objective of the actual forecasting is diverging from purely improving accuracy to increasing the economic value of the obtained results.

Although several forecasting models have been developed throughout the last decades, its majority is focused on single-value (point) predictions. Such predictions are normally not enough when the aim is decision-making.

Therefore, the use of additional information on the uncertainty of the forecasts is of utmost importance for instance for system operators that manage networks with high wind power integration or for bidding in the electrical market. Thus, surging the need to associate the traditional point forecast to a new dimension of information, its uncertainty.

This implies treating a high volume of data, making the process very time consuming, which sometimes is enough to make this type of prediction prohibitive.

In this thesis, we propose the use of a method based on the kernel density estimator (KDE) for computing both the complete predictive probability density function (PDF) as well as the expected value of the distribution for each time-step of the prediction horizon. Thus, we start by presenting two variations of the algorithm, one for deterministic and other for probabilistic forecasting. We also discuss two possible extensions to the basic methodology in order to improve the quality of its results.

The main problem of this type of method is the huge computational requirements, as it involves dealing with large data sets and relatively complex models, therefore having to cope with a large number of computations.

In order to accelerate this process, we developed a parallelized version for each forecasting algorithm, proposing a parallel computing solution that takes advantage of graphic processing units (GPU) capabilities using the Compute Unified Device Architecture (CUDA) programming model.

Later, after briefly discussing the benefits and drawbacks of the proposed extensions, we evaluate the implemented algorithms both in terms of accuracy and time performance using real data from two operating Portuguese wind farms.

Finally and in order to test and validate the results, we make a comparison between the proposed methodology and a prediction system based on neural network technique.

Resumo

Nos últimos anos tem-se verificado, a nível mundial, uma elevada penetração de energia eólica nos mercados eléctricos. Consequentemente, e devido às restrições de integração deste tipo de energia, tem aumentado a necessidade de realizar uma previsão de produção eólica de curto prazo, com maior volume de informação.

Cada vez mais, a fiabilidade da previsão é um factor a ter em conta, devido ao impacto económico que os erros desta provocam.

Ao longo das últimas décadas foram estudadas e desenvolvidas um conjunto de técnicas de previsão de produção eólica, contudo, a sua maioria tem como principal objetivo fornecer uma previsão pontual. Com a alteração do paradigma, este tipo de previsão, não se revela suficiente, na maior parte das situações. Quando um operador se encontra a gerir uma rede com elevada integração de energia eólica, este necessita de uma ferramenta não de previsão pontual, mas sim de apoio à decisão. Surge assim a necessidade de associar à tradicional previsão pontual, uma nova dimensão de informação relacionada com a incerteza da previsão. Para que tal seja possível, é necessário o tratamento de um elevado volume de dados, o que torna os processos morosos, e muita das vezes inviabiliza este tipo de previsão.

Nesta tese, propomos a utilização de um método baseado no "Kernel density estimator" (KDE), de forma a calcular tanto a função densidade probabilidade como o valor esperado da distribuição, para cada horizonte temporal. Assim, começamos por apresentar dois algoritmos, um para previsão determinística e outro para probabilística. Também será discutindo a introdução de duas possíveis extensões que podem ser adicionadas à metodologia básica, a fim de melhorar a qualidade dos resultados.

A principal desvantagem deste método de previsão deve-se à elevada necessidade computacional que acarreta, uma vez que envolve elevados volumes de dados, bem como um consideravel número de cálculos relativamente complexos.

Para acelerar este processo, desenvolvemos uma versão que tem a capacidade de paralelizar cada algoritmo de previsão, de modo a retirar partido das capacidades do GPU (Graphics Processing Units). Para que isto seja possível foi utilizado o modelo de programação CUDA (Compute Unified Device Architecture).

Após uma discussão sobre as vantagens e desvantagens das extensões propostas, procedemos à avaliação dos algoritmos implementados, tanto em termos de precisão como de

desempenho, utilizando dados reais de dois parques eólicos situados ao longo de Portugal Continental.

Para finalizar e com o objectivo de testar e validar os resultados obtidos, realizamos uma comparação, com um sistema de previsão baseado em técnicas de redes neuronais para os mesmos parques. Este tipo de técnicas são as mais utilizadas para realizar previsão de produção eólica actualmente.

Aknowledgements

I would like to begin by thanking my parents, sisters and the rest of my family not only for their support during the realization of this thesis but for always being there for me.

I am also really thankful to my supervisor Rui Rodrigues for giving me the opportunity to perform this thesis work and for backing me up during its course.

Furthermore, I am also grateful to have been able to carry out my master thesis in Smartwatt. It was very interesting and instructive to get the experience of working in a real company environment and for that I thank Claudio Monteiro, Tiago Santos and Ruben Costa.

I have really enjoyed writing my thesis there and I would therefore like to thank all the people that have contributed to making it a pleasure to go to work, especially Bruno Santos for always helping me when needed.

I must also acknowledge António Cardoso for helping me reading and correcting my writings.

Last but not the least I would like to thank all my classmates, friends and teachers at FEUP for the fantastic time there. It was an amazing journey during which I have learned a lot.

António Pires

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ANN	Artificial Neural Network
ARMA	Autoregressive Moving Average
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FIS	Fuzzy Inference System
GPGPU	General-Purpose computing using GPU
GPU	Graphics Processing Unit
KDE	Kernel Density Estimator
LQR	Local Quantile Regression
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MSE	Mean Square Error
NMAE	Normalized Mean Absolute Error
NRMSE	Normalized Root Mean Square Error
NW	Nadaraya-Watson
NWP	Numerical Weather Prediction
PDF	Probability Density Function
QRF	Quantile Regression Forest
RMSE	Root Mean Square Error
SCADA	Supervisory Control And Data Acquisition
SVM	Support Vector Machine
WPF	Wind Power Forecasting

Chapter 1

Introduction

In a world increasingly dependent on electrical power and concerned about the escalating pollution levels, several countries have greatly stimulated the use of green energy solutions. Being one of the fastest growing renewable electricity technologies [1], wind power has become an important part of the global energy production. Countries like China, USA and Germany have already attained a considerable rate of installed wind power capacity. Moreover, great investments have been made in the last few years which resulted in a worldwide annual growth rate of cumulative wind power capacity over the period from end 2006 to end 2011 averaging 26% [2, 3].

Wind power is a renewable energy with little environmental impact and with reduced operating costs when compared to fossil-fueled plants, making this type of energy extremely competitive, having an increasingly greater impact on global market's energy average prices.

However, as with other renewable energy sources, the power output from wind turbines is a stochastic variable highly dependent on weather conditions, mainly the wind speed, which results in great variations and uncertainty in the actual power generation.

The work presented was conducted in the scope of the Smartwatt's research and development effort and aims at providing an approach to wind power production forecasting, increasing both the data processing speed and amount of available information about the forecast, therefore, adding value to the obtained predictions and providing greater confidence in company's solutions, namely in renewable and energy efficiency areas.

1.1 Motivation and goals

Wind power intermittence brings new challenges and needs to several players in the energy market, which need to reduce the additional costs caused by it.

The reduction of these costs is increasingly dependent on models which are capable not only of accurately predict the expected wind power output for the coming hours and days, but also of providing useful information to support decision making.

This information can be very useful for instance for transmission system operators (TSO), which are responsible for efficiently integrating the generated power into the power system and for balancing the energy on the grid at any moment. It can also be very valuable for wind power producers who need to offer energy in the market, or to control and regulate energy transport lines.

Many forecasting models have been developed throughout the years, aiming to mitigate the drawbacks of this unreliable production. Nevertheless, its majority concentrates on the prediction of one value for each horizon, i.e. look-ahead time, limiting their use in risk analysis. Therefore, as discussed in [4], the use of additional information on the uncertainty of the predictions is a great added value when performing risk assessments. Indeed, as wind power penetration grows, more and more information is required in order to optimize resource usage and adjust behaviors, as costs associated with its uncertainty also scale up.

This increase on the quantity and quality of the available information about the predictions comprises a great amount of additional data involved on the process as well as more complex computations, which in turn leads to an increase of the necessary processing time. In an environment in which the results have to be accurate and obtained in real time, the reduction of the processing time is of the essence.

In order to approach these two problems, first we propose the implementation of a forecasting method based on a kernel density estimator (KDE), which is used to compute single expected values and then modified to obtain probabilistic forecasts of wind power production. Moreover, some extensions to the base methodology are introduced and evaluated.

The main problem of using this type of method is the huge computational requirements, as it involves dealing with large datasets as well as with relatively complex models, based on sophisticated estimation techniques.

Thus, in order to cope with the computational demands of this kind of estimation, we propose the use of a parallel computing solution using a graphics processing unit (GPU).

Accordingly, in this paper, besides presenting two sequential algorithms, one for computing a point forecast and other for a probabilistic one, we propose one GPU parallelized version for each algorithm using Compute Unified Device Architecture (CUDA) programming model.

1.2 Structure

The remainder of the thesis proceeds as follows. In chapter 2, the main research areas are introduced, reviewing existent literature and related work, identifying possible existing solutions to the problems we intend to address. First an overview is given about forecasting and different existent models and techniques for point and probabilistic forecasting are discussed.

The second section of this chapter presents an introduction to GPU and general purpose computing on GPU (GPGPU), referring possible platforms to be used and describing in more detail the CUDA programming model and architecture. Finally, we make a brief reference to several papers that document the use of GPGPU for speeding up algorithms applied on wind power forecasting.

Chapter 3 describes the used methodology and approach taken, starting by detailing the forecasting problem in which the work is focused. Then, the KDE and the actual proposed forecasting model are presented and explained, first aiming point forecasts and then referring some modifications in order to obtain the probabilistic version. This chapter also includes a section where some extensions are addressed in order to overcome some model limitations.

The actual development and implementation phase is described in chapter 4. This chapter addresses the various algorithm implementations, which are split in separated steps for ease of comprehension. First, the point forecasting model, both in its sequential and parallelized versions, is explained, followed by the presentation of its probabilistic counterparts. This chapter also addresses the implementation modifications underlying the discussed extensions to the basic methodology.

Chapter 5 addresses the validation process, drawing results of accuracy and time performance from several tests performed for both types of forecast. First the point forecast algorithms are tested and a comparison is made between the implemented methodology and an artificial neural network. This is followed by an assessment of the probabilistic forecast results, which are also evaluated and discussed.

Finally, chapter 6 draws general conclusions from the final algorithms implementation and their results, also presenting possible ways to further benefit from the work performed in the scope of the thesis.

Chapter 2

State of the art

This chapter is divided in two major sections, which provide an overview of the main topics involved on the developed work.

In section 1, we present a general introduction to the forecasting theme, followed by a general classification of the prediction models, mentioning the problems for which each one can be applied. Then, we present a more focused description about the wind power forecasting problem, its specific characteristics as well as several methods for evaluating the predictions quality. Finally a review of wind power forecasting state of the art is presented, addressing several documented methods and discussing the obtained results for different forecasting horizons and time-steps.

In section 2 we explore the use of GPU for general-purpose programming (GPGPU). We start by introducing the GPU and the GPGPU main advantages and principles, addressing different existent models and frameworks. Then we describe in more detail the CUDA model and architecture, and finally we make a brief reference to documented wind power forecasting models implemented using this technology.

2.1 Forecasting

Forecasting can be shortly defined as the analysis of current and past information in order to determine the direction of future trends. Predictions are required in many situations and can actually help to plan in advance what to do and how to do it in order to obtain the best outcome. They can provide very useful information and are present in many daily life situations, as weather forecasts, election forecasts or products future demand estimation. Forecasting has been extensively used in the energy field for instance to predict electric demand (i.e. how much energy has to be generated to fulfill a certain group of consumers' needs) [5, 6].

However, forecasting does not apply to all kinds of problems. First the problem has to be analyzed in order to conclude whether or not it is possible to obtain a valid prediction, since it implies the existence of a causal relationship between the result and some factor, i.e. the event to be predicted cannot be fully independent, otherwise the results are not relevant, as in the case of completely random events like the next lottery winning numbers.

When a valid forecast can be obtained, an appropriate forecasting method needs to be selected in order to solve the problem [7]. This selection is dependent in great extent on what data is available, what is known about the problem behavior model (i.e. the relationship between the factors that lead to a certain occurrence) and the prediction horizon (i.e. how far into the future we are trying to forecast). Besides, a comparison between the prediction error obtained using different methods can be an immediate selection criterion.

Forecasting methods can be classified in two categories, qualitative forecasting when no data are available, and quantitative forecasting when relevant information in which the prediction can be based is accessible. These categories are described in more detail in the following sections.

2.1.1 Qualitative vs. quantitative forecasting

Qualitative forecasting methods are usually used when there are no past data available or if the data available is of no relevance to the forecast and normally target medium-long term horizons. Although predicting an event without historical data may seem pure guessing, there are many developed approaches to obtain a good forecast without the access to past data. These methods are subjective and based on consumers or experts' opinion and judgment. As examples we can consider forecast by analogy, which assumes that two different events share the same behavior model and base the forecast on the already known event's past data (ex. predict a new product's expected demand based on already existing product's data), statistical survey, that base the forecast on data obtained from surveys conducted on a sample of a target population (ex. Predict whose candidate is going to win prime-minister election based on voting population' answers) and the Delphi method [9], which involves relying on a group of experts whose opinions are compiled, considered and discussed until a group consensus is reached as to what is going to happen in the future.

Quantitative forecasting methods are normally adopted for short-medium range predictions and, unlike qualitative forecasting, can only be applied in cases where historical data are available. Moreover it must be possible to assume that some kind of past pattern exists and that it will continue into the future. Thus, future values can be obtained from a function of past data.

Quantitative forecasting methods can be further divided into two separate categories, depending on what data is being analyzed and how: causal models and time series models.

Causal models are used when it is possible to understand the dynamic between causes and corresponding effect, i.e. what factors may influence the variable that is being forecasted and what effect are they causing in the resulting value. With these models, any change on the influencing factors will alter the output value in a more or less predictable fashion. On the other hand, time series models only use information on the variable to be forecasted, not taking into consideration any factors that can affect its behavior, assuming that the variable changes over time depending only on the time and previous values. This data can exhibit several patterns that can be considered in order to improve the forecast results. Three types of patterns can be extracted when decomposing time-series' data: trend, seasonal and cyclic patterns. [11]

A trend exists when the value of the variable to be forecasted increases or decreases over long periods of time. Seasonal patterns exist when the data is affected by seasonal occurrences in fixed periods of time. Cyclic patterns are characterized by recurrent upward and downward movements with no fixed time periods. These components can be plotted and analyzed separately so that the problem can be better understood.

Quantitative forecasting encompasses a wide range of different methods, including regression methods, autoregressive moving average (ARMA), artificial intelligence models, exponential smoothing, etc.

In the scope of wind power forecasting only quantitative methods are of interest, thus, qualitative methods will be no further discussed.

2.1.2 Wind power forecasting

WPF data

Three data sources can be used when forecasting wind power production: historic data, numerical weather predictions (NWP) and terrain information.

Historical data is obtained from the Supervisory Control And Data Acquisition (SCADA) system, generally installed in every wind turbine, which is responsible for regularly collect measures of power output and sometimes meteorological variables as wind speed and direction.

NWPs are provided by meteorological agencies and are obtained using models based in series of equations that try to define physical laws of the atmosphere. However, due to the atmospheric variability, the obtained results are very sensitive to initial conditions, which consist in a large set of measurements made by meteorological stations, balloons and radars. Indeed, this high error probability on the NWPs can lead to an amplification of the error obtained on WPF models that rely on this type of data.

At last, terrain information, that consists solely on information about the location where the wind farm is located such as roughness, orography or obstacles, which help to model the wind farm's site.

WPF general model classification

In the particular case of WPF, models can be classification based on the input data they use. Therefore, WPF models can be divided into three categories: time-series, statistical and physical models.

Time-series models, as ARMA and ARIMA can be applied to sets of past measured power outputs, retrieved from the SCADA system, in order to produce power predictions. This type of methods are capable of obtaining relatively accurate forecasts up to 6 hours ahead, however its effectiveness decreases considerably when used for longer horizons due to the non-linearity characteristics of the atmosphere.

Statistical models try to establish a relation between historical power output and a set of explanatory variables, and then replicate this pattern in the future. Such models are based for example on ANNs which after mapping this relation over a training set of historic values, the forecasts can be made using meteorological predictions of the explanatory variables, obtained from a NWP model.

Physical models are based on modeling the wind flow around the wind farm in order to refine the NWPs for different locations in the farm, which are then used to forecast wind power output. However, in order to do this the model needs wind farm's site information. Moreover, despite being the most accurate models for long term predictions the simulations can demand large computational power.

Finally, note that these different forecasting models can be used independently with different data inputs, although a combined approach can also be implemented to improve the model's results.

Temporal scales

In terms of temporal dimension, WPF can consider different time scales, which are normally classified as: very-short, short and long term forecasts.

Very-short term encompass predictions from a few milliseconds up to 9 hours ahead. This type of forecasts is used for turbine active control and for forecasting needed adjustments in the interdaily energy market.

Short term predictions go from 9 to 72 hours horizons, and are required for power system management when scheduling power plants production (dispatch) and for market trading, as bids are normally required on the morning of day d for the next day $d + 1$.

Finally, long term forecasts include all predictions for more than 72 hours, normally for 5 to 7 days ahead, which are used for planning maintenance operations on the wind farms.

It is also noteworthy that these different time scales can have various time steps, defining the sampling density of the forecasting series, which could vary from 10 or 15 minutes for short-term predictions to 1 or 3 hours for longer horizons.

2.1.3 Deterministic vs. probabilistic forecasts

Deterministic forecasts, also called point or spot predictions are the most commonly used. This type of forecast provides a single expected value $P_{t+k|t}$, computed at time t, for each horizon k. Although this type of predictions are simpler to use and analyze, its support in decision making and risk assessment is limited, as it gives no information about the possible deviation of the predicted value.

This need can be fulfilled by probabilistic forecasts which present the uncertainty of the forecasted value, i.e. a range of values associated with a probability of containing the correct future value. For instance, while a point forecast would inform that for certain time a wind turbine is expected to produce a given power, a probabilistic forecast would present for the same time an interval of expected produced values with a correspondent probability for its occurrence (e.g. between 100kW and 200kW with 90% probability).

Probabilistic forecasts can be expressed in many forms: probability density function (PDF), probability mass function (PMF), moments of distribution (e.g. mean, variance, skewness, kurtosis) or as point predictions with prediction intervals computed from quantiles.

The most commonly used representations are the interval and quantile forms. However, of all, PDFs can provide clearer information and are a generic form representation, meaning that can be transformed in all of the other probabilistic forms.

Two representations of probabilistic WPFs are presented in figure 1, a PDF on the right and a set of prediction intervals with a point forecast on the left.

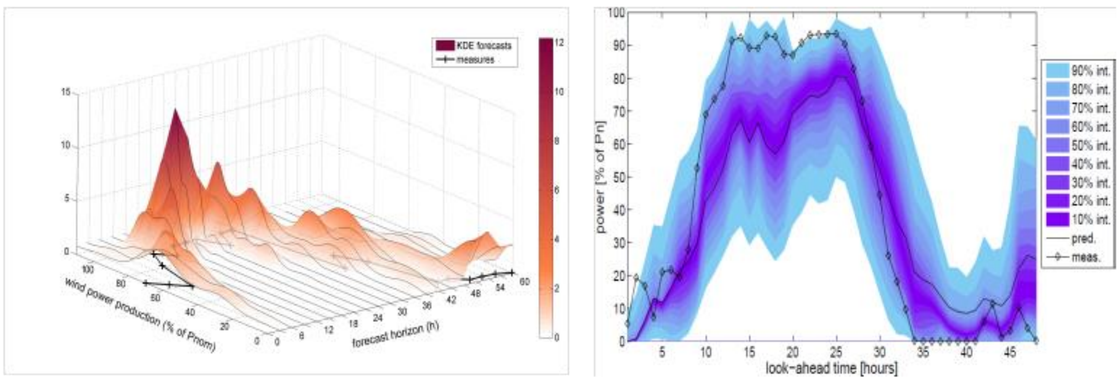

Figure 1: Example of two probabilistic forecasts. These figures are taken from [36] and [12].

Essentially two main methodologies are defined for wind power probabilistic forecasting: statistical and ensemble forecasts.

The statistical approach calculates uncertainty based on estimates done inside the forecasting method for a single NWP.

On the contrary the ensemble forecasting basic idea is to obtain a set of different forecasting results which are then analyzed in order to assess the forecast uncertainty. This different results can be obtain from two different ways, or by running the same model using a set of different NWPs (i.e. using different initial conditions) or by using a group of different forecasting models to predict a single NWP.

2.1.4 Determining errors and evaluating models

Deterministic models

In order to quantitatively evaluate a point forecasting model several error measurements can be employed. The error observed at a given time $t+k$ for a prediction made at time t is $e_{t+k|t}$ and can be defined as the difference between the real value, p_{t+k}, measured at $t+k$, and the value predicted at time t for the horizon k, $F_{t+k|t}$. This results in the following equation:

$$e_{t+k|t} = p_{t+k} - F_{t+k|t} \quad (2.1)$$

This prediction error can be normalized dividing $e_{t+k|t}$ by $P_{max} - P_{min}$, representing the maximum and minimum values that the real measurement p can adopt.

There are several error measurements that are commonly used, based on this simple error calculation, which can give some information about how each method behaves for a given test set.

The mean absolute error (MAE) is used to measure the average magnitude of the errors over the test set and is defined as:

$$MAE = \frac{1}{N}\sum_{t=1}^{n}\left|e_{t+k|t}\right| \quad (2.2)$$

When using the normalized prediction error, MAE becomes NMAE, and dividing the prediction error for the real measured value at time $t+k$, and multiplying its summation over the test set by 100 gives the mean absolute percentage error (MAPE), defined as:

$$MAPE = \frac{1}{N}\sum_{t=1}^{n}\left|\frac{e_{t+k|t}}{p_{t+k}}\right| * 100 \quad (2.3)$$

Other important information about a forecast is the variation of the errors, which can be analyzed using the root mean square error (RMSE) that is calculated from the mean square error (MSE):

$$MSE = \frac{1}{N}\sum_{t=1}^{n}\left(e_{t+k|t}\right)^2 \quad (2.4)$$

Therefore the RMSE can be defined as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{n}\left(e_{t+k|t}\right)^{2}} \quad (2.5)$$

As in the case of the MAE, both RMSE and MSE can be calculated using the normalized prediction error becoming in that case NRMSE and NMSE respectively, which makes it easier to analyze large errors.

When evaluating the performance and the advantages of using more complex and advanced models, normally, a comparison is made with a reference model. The simplest one is *persistence*, which assumes that the future value of the variable to forecast is equal to the real measurement at the time the forecast was made:

$$P_{t+k|t} = p_t \quad (2.6)$$

Where $P_{t+k|t}$ is the prediction done for the horizon k at time t and p_t is the real measurement obtained for time t. Due to atmosphere suffering constant but slow changes over time, *persistence* can be used with relatively good accuracy in very-short-term wind power predictions, from 0 to 6 hours ahead.

Probabilistic models

Regarding probabilistic models evaluation, in [13] a systematic framework to make this assessment was proposed, being adopted in this thesis.

The evaluation relies on three metrics: reliability, sharpness and skill. Although in this work only the former two are used as skill is a junction of both.

Reliability represents the ability of the model to obtain matching quantile probabilities (nominal proportions) and the real percentage of values actually under the considered quantile (empirical proportions). For instance an 85% quantile should contain 85% of the real measured values below or equal to it.

Hence, the evaluation of the reliability of the probabilistic model is achieved by verifying the reliability of different quantiles. In this work quantiles are evaluated ranging from 10% to 90% with 10% increments.

Given a quantile $\hat{q}^{\alpha}_{t+k|t}$ forecasted at time t for the horizon $t+k$, with nominal proportion α and the actual measurement p_{t+k} we have:

$$\varepsilon^{\alpha}_{t,k} = \begin{cases} 1, & if\ p_{t+k} < \hat{q}^{\alpha}_{t+k|t} \\ 0, & otherwise. \end{cases} \quad (2.7)$$

Thus $\{\varepsilon_{t,k}^{\alpha}\}$ $(t = 1, \ldots, N)$ is a binary sequence that represents the hits (values of p_{t+k} below the quantile) and misses (otherwise) for a given horizon k with N time-steps. The empirical proportion, represented by $\hat{a}_k^{\alpha}$ is obtained by dividing the number of hits by the total number of observations:

$$\hat{a}_k^{\alpha} = \frac{1}{N}\sum_{t=1}^{N} \varepsilon_{t,k}^{\alpha} \quad (2.8)$$

After obtaining this value, we can compute the bias of the probabilistic forecast for each quantile, which is calculated as the difference between the nominal and empirical proportions.

$$b_k^{\alpha} = \alpha - \hat{a}_k^{\alpha} \quad (2.10)$$

Sharpness corresponds to the model ability to concentrate the probabilistic information about the forecast. Quantiles are paired up in order to obtain intervals with different nominal coverage. Let $\delta_{t,k}^{\beta}$ be the size of the central interval forecast with nominal coverage rate of $1 - \beta$, estimated at time t for $t + k$. A measure of sharpness is given by the mean distance between the quantiles' values:

$$\bar{\delta}_{t,k}^{\beta} = \frac{1}{N}\sum_{t=1}^{N} \delta_{t,k}^{\beta} = \frac{1}{N}\sum_{t=1}^{N} \hat{q}_{t+k|t}^{(1-\frac{\beta}{2})} - \hat{q}_{t+k|t}^{\frac{\beta}{2}} \quad (2.11)$$

2.1.5 Wind power forecasting review

In this section a review of several documented wind power forecasting methods is performed, for both deterministic and probabilistic forecasts, using different time horizons as well as time-steps. For each one, accuracy performance results are assessed and shortly discussed.

Deterministic models

The first presented algorithms use time-series analysis in order to predict either wind speed or power and as previously referred, normally used for very short-term horizons.

The first model used to solve the wind forecasting problem was a Kalman filter in 1985 by *Bossany* [15] that used the last 6 observed values for one-step-ahead predictions with different horizons, obtaining for the 1 minute prediction 10% RMSE improvement over persistence. A

Kalman Filter method was also employed by *Babazadeh et. al.* [16] for wind speed predictions with different time horizons (but using a Gaussian markov system to calculate the state transition matrix). The results showed a RMSE of 0.1494 for the 2 minutes ahead prediction whereas that for the 60 minutes was 3.5711. *Morrison and Pike* [17] also documented the use of this method but for electrical power demand forecasting.

Other possible solution is the use of a Box-Jenkins based model. Both ARMA and ARIMA have been extensively used on time-series analyses and several applications of these methods for wind speed and power forecast have been documented in literature.

Milligan et. al. [18] used an ARMA model to forecast hourly power up to 6 hours ahead. The results showed that the RMSE improved on persistence by about 3% for the first hours forecast whereas those improve for the 6 hours forecast was of 11%. The authors also document the use of ARMA models for 10 minutes intervals forecast up to 8 periods ahead. This model achieved 16% RMSE improvement over persistence for one period ahead forecast, falling to 7.5% four periods ahead and obtaining the same performance as persistence eight periods ahead. The authors also suggested the use of an ensemble of different ARMA models, depending on the performance of different model variations for different horizons.

Torres et.al. [19] employed an ARMA model to forecast hourly wind speed up to 10 hours horizon in five different locations. The results showed that this model, as opposed to the persistence method, obtained a percentage improve of the RMSE, averaged over all the forecasts, between 2% and 5% for one hour ahead forecast, reaching 12% to 20% improvement in 10 hours in advance forecasts.

Kavasseri et al. [20] proposed a fractional-ARIMA (f-ARIMA) model, that used records from four North Dakota's wind monitoring sites, to forecast hourly average wind speeds up to two days ahead, obtaining an accuracy improve of 42% when compared to the persistence method.

Chen et al. [21] presented a limited-ARIMA (LARIMA) model, introducing a limiter in the standard ARIMA model to represent the upper and lower bounds of wind power generations.

Another alternative type of solution for this problem is the use of artificial intelligence models, namely neural networks, fuzzy inference systems (FIS) and support vector machines (SVM). This type of methods have been widely studied and developed in the forecasting context especially the neural network solution, which is one of the most documented for solving the wind power forecast problem, being a simple to implement method and obtaining relatively good results for very short and short term predictions.

Cadenas and Rivera [22] compared a seasonal-ARIMA (s-ARIMA) and an Adeline neural network for forecasting monthly average wind speed in Mexico. The s-ARIMA model performed better, presenting a 13.4% MAPE whereas for the ANN this value was 20.7%. However, the ANN performance is expected to improve if the number of training vectors increases.

In *Castro and Gomes* [23] three case-studies are presented and a performance comparison is made between an ARMA model and feed-forward neural network with different configurations. The results showed that, although both methods performed better than persistence, the ARMA model achieved slightly better forecasts. For one hour ahead forecasts both models obtained a minimum of 0.78 RMSE in the best case although the ARMA presented a smaller MAE of 0.50. In the second case the forecasts were performed with four to nine hours in advance during a five days period. The results showed that the ARMA model obtained a 2.20 RMSE and 1.80 MAE whereas the best neural network obtained 2.27 and 1.85 respectively.

Cadenas and Rivera [24] tested several neural network configurations for hourly wind speed forecasting. The results showed that the simplest model, with two layers and three neurons, outperformed the more complex ones. This approach was compared to a simple exponential smoothing method in *Cadenas et.al.* [25], where the latter showed slightly better results when forecasting wind speed for 10 minutes horizon, only relying on last period forecast and current real measurement.

Kojabadi et. al. [26] used statistical properties, such as standard deviation, average, and slope, obtained from historical data, as inputs of a fuzzy system and a neural network with a backpropagtion algorithm.

Catalão et. al. [27] studied the performance of a three layered feed-forward neural network using the Levenberg-Marquardt algorithm for training. The resulting values had a MAPE of 7.26% for a 24 hour horizon forecast with hourly resolution. Besides, the average computation time was less than 5 seconds.

Li and shi [28] compared the performance of three different neural networks, a back-propagation (BP), a radial basis function (RBF) and an adaptive linear element (ADELINE). Each neural network was tested with data of two different locations and the effect of different learning rates, inputs and model structures were studied. The results obtained for hourly average wind speed forecast showed that different this three variables directly influence the forecast accuracy, besides different models obtained the better solution in different locations, therefore, the authors suggested the implementation of a method that combine the results of different ANN models.

Potter et. al. [29] described an adaptive neural fuzzy inference system (ANFIS), which combine two artificial intelligence models (neural networks and fuzzy inference system). This method was used to forecast wind speed for 2.5 minutes horizons. Obtained results showed that this model produced a MAPE of 3.5% in the worse situation, whereas the persistence method obtained no less than 38.4%. Despite these results, periods of no wind proved to be difficult to predict, thus presenting a greater error.

For greater time horizons forecasting (up to 72 hours), NWP results are normally used as input variables to improve the forecasts' accuracy. Several methods with such behavior have been evaluated and documented in literature. Although this sort of methods can be used for even longer term forecasts, in general its accuracy is highly dependent on the NWP quality.

Georgilakis et. al. [30] proposed a method that combines a neuro-fuzzy system and an artificial neural network, which uses the former method outputs to improve the forecast accuracy. This method is used to make forecasts up to 24 hours ahead with one hour time steps in two test cases and its performance is compared with a simple neuro-fuzzy system. The proposed method obtained better results in both tests, showing significant improves.

Fugon et. al. [31] employed six data mining methods (2 linear and 4 non-linear methods) to make forecasts for 60 hours horizons with 3 hours resolution using a combination of meteorological variables' predictions and historical data as input. The linear methods were a simple regression model and a regression model with interaction, in which crossed products between variables are added to the inputs. The non-linear models considered were a neural network with back-propagation, a support vector machine, regression trees with bagging and a random forests model. The test runs were conducted with data of three different locations, concluding that the forecasting accuracy was dependent on the terrain complexity, as for flatter terrains the error obtained by all the methods was smaller. Finally the results showed that, although all the methods performed better than the persistence, the non-linear ones outperformed both regression models, being the best results achieved by the random forests and SVM methods.

Bessa et. al. [32] presented two forecasting systems, FORECAS and SGP. The first one is a short-term forecasting system that combines inputs of historical production and information about wind speed and direction obtained by a NWP forecasting model. The second system encompasses a set of different forecasting models specialized in different time horizons, this way the system can choose the best for each situation. Besides, for each time horizon three models are selected so that their outputs were the inputs of a fuzzy inference system that provided the forecasted value as a non-linear combination of the outputs of the three selected models. For very short term forecasts historical information was used and there were 11 forecasting models implemented that included one ARMA(1,1) and nine ANN, one for each horizon (0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4 and 4.5 h). For longer term forecasts NWP results were the system inputs and 3 models were implemented, also based on neural networks (responsible for 5 to 24h, 24.5 to 48 and 48.5 to 72h horizons respectively). The results showed the RMSE for both systems ranged between 10% and 25% with a mean 17% for 72 ahead with a time step of 30 minutes. Both systems outperformed the persistence model obtaining improvements in the RMSE over persistence ranging from a minimum of 7% for the first hour to a maximum of 65%.

Zheng et. al. [33] purposed a forecasting method that uses an empirical mode decomposition (EMD) for decomposing wind power into different components and a radial basis function neural network (RBFNN) used to forecast each component. After the forecast, the components are again aggregated to obtain the final predictive value. In the results, this method performance is compared with a direct RBFNN forecast for one hour ahead predictions, obtaining the former a MAPE of 17.582% whereas the latter got 29.543%.

Probabilistic models

In [34] *Bremmes* proposes a method, local quantile regression (LQR), in order to estimate different quantiles of energy production based on data obtained from NWPs. The methodology is tested with data obtained from two different NWP models for a wind farm in Norway. In another paper by *Bremmes* [35] three probabilistic forecasting methods are tested, a LQR, a Nadaraya-Watson (NW) estimator for conditional cumulative distribution functions and a local Gaussian model, which assumes that the distribution around the forecasted value can be approximated with a Gaussian. Before applying these methods to a wind farm in Norway, the author does no show a clear preference for one of the methods. However, the local Gaussian model produced slightly more uncertain results.

In *Juban et. al.* [36] two models for probabilistic wind power forecasting are introduced and compared to a B-spline quantile regression model. The first model is based on a KDE technique and aims to produce the complete PDF for each time step of the forecasting horizon. The second method is a quantile regression forest (QRF), proposed by *N. Meinshausen* [37], a generalization of random forests, which provide a non-parametric way of estimating conditional quantiles. A deterministic comparison with a point forecasting model was made as well as an evaluation of different performance criteria of a derived quantile prediction, showing similar results to those found on literature.

Juban et. al. [38] compared four probabilistic models, KDE, spline quantile regression (SQR), QRF and a LQR used as reference model. Two different comparisons were made, the first using a point NWP prediction and the second using an ensemble of NWPs as inputs. The results showed that all the three methods performed similarly, obtaining better results than the reference model. Also, in the considered tests, the use of NWP ensembles showed relatively small improvements over the point NWP solutions.

Yang et.al. [39] propose an approach based on Sparse Bayesian Learning (SBL) algorithm, which produces probabilistic forecasts by estimating the probabilistic density of the weights of Gaussian kernel functions. In this paper a componential forecasting strategy is used, decomposing the wind generation into several more predictable series in order to improve model results.

In *bessa et.al.* [40] and [41] two time-adaptative KDE based models are presented. On both papers a comparison is made to a quantile regression (QR) model, concluding on both cases that the KDE with the NW estimator present better reliability values, while the QR model as tendency to present better sharpness. Beside the time-adaptative novelty, other alteration is introduced to the typical model, namely the adoption of specific kernels for each variable. Finally an important conclusion is drawn, stating that density estimation models, such as the one presented offer significant advantages for the wind power industry when uncertainty estimation is required.

2.1.6 Concluding remarks

To conclude this overview of the most common wind power forecasting methods, table 1 presents a summary of the addressed point forecast papers with the results obtained for each method. However, these results have to be analyzed with some care, as a fair comparison between methods used in different forecast environments is quite difficult since system performance is closely related to various characteristics that are different depending on the situation. For instance, terrain complexity, wind farm size, turbine installed power, geographical location, climatology conditions, NWP quality or historical data measurements errors.

From the analysis of the table it is possible to conclude that both the Kalman filters and the Box-Jenkins based models (ARMA and ARIMA) are well fitted for very short term forecasts, especially for one-step-ahead predictions, On the other hand the artificial intelligence models, as the ANNs, are more versatile, obtaining relatively good results for a wide range of horizons and time-steps.

Regarding the probabilistic forecast methods, it is not trivial to make comparison between methods. This happens because in addition to the care that must be taken when analyzing forecasts for different locations, it is difficult to translate the quality of a probabilistic forecast to a single value that can be used to compare methods. Thus, for a better evaluation, a more detailed study would be necessary, something that is beyond the scope of this work.

Table 1: Forecasting algorithms resume table.

	Criteria				
Paper	*Algorithm*	*Horizon*	*Time-step*	*Metric*	*Improve over persistence*
Babazadeh et. al. (2012)	Kalman Filter	2 min	2 min	0.1494 RMSE	
		1 hour	1 hour	3.5711 RMSE	
Milligan et. al. (2004)	ARMA	6 hours	1 hour		11% RMSE
		10 min	10 min		16% RMSE
		80 min	10 min		0% RMSE
Torres et.al. (2005)	ARMA	1 hour	1 hour		2-5% RMSE
		10 hours	1 hour		12-20% RMSE
Kavasseri et al. (2009)	f-ARIMA	48 hours	1 hour		42% RMSE
Cadenas and Rivera (2007)	s-ARIMA	1 month	1 month	13.4% MAPE	
	ADALINE	1 month	1 month	20.7% MAPE	
Castro and Gomes (2012)	ARMA	1 hour	1 hour	0.78 RMSE 0.50 MAE	
	ANN	1 hour	1 hour	0.78 RMSE 0.53 MAE	
Cadenas and Rivera (2009)	ANN	1 hour	1 hour	0.0016 RMSE	
Cadenas et.al. (2010)	Exponential Smoothing	10 min	10 min	0.5303 MSE	
	ANN	10 min	10 min	0.6403 MSE	
Catalão et. al. (2009)	ANN	24 hours	1 hour	7.26% MAPE	
Potter et. al. (2006)	ANFIS	2,5 min	2,5 min	3.5% MAPE	
Georgilakis et.al. (2006)	neuro-fuzzy and ANN	24 hours	1 hour	12.79%, 10.22% MAPE	
	neuro-fuzzy	24 hours	1 hour	24.32%, 17.52% MAPE	

Paper	Criteria				
	Algorithm	*Horizon*	*Time-step*	*Metric*	*Improve over persistence*
Fugon et. al. (2008)	Linear regression	60 hours	3 hours	~16-18%, ~13-14%, ~10-11% NMAE*	
	Linear regression w/ iterations	60 hours	3 hours	~16-18%, ~13-14%, ~10-11% NMAE*	
	Random Forests	60 hours	3 hours	~16-18%, ~13-14%, ~9-10% NMAE*	
	Regression trees w/ bagging	60 hours	3 hours	~16-18%, ~13-14%, ~9-10% NMAE*	
	ANN	60 hours	3 hours	~16-18%, ~13-14%, ~9-10% NMAE*	
	SVM	60 hours	3 hours	~16-18%, ~13-14%, ~9-10% NMAE*	
Bessa et.al. (2008)	FORECAS	12h to 24h	30 min	14% RMSE	
		24h to 48h	30 min	17.6% RMSE	
		48h to 72h	30 min	19.7% RMSE	
	SGP	12h to 24h	30 min	14.1% RMSE	
		24h to 48h	30 min	16.2% RMSE	
		48h to 72h	30 min	18.8% RMSE	
Zheng et. al. (2013)	EMD and ANN	1 hour	1 hour	17.582% MAPE	
	ANN	1 hour	1 hour	29.543% MAPE	

*These are approximated values as the paper does not refer explicitly the error.

2.2 Graphic processor Units

Graphic processor units (GPUs) are specialized processing units designed with the aim of reducing the workload on the central processing unit (CPU) when performing graphics intensive tasks, for example in video games, interactive simulations or just 3D rendering. In the last few years the GPU performance have largely overcome common CPUs in terms of speed (floating point operations per second) and memory bandwidth.

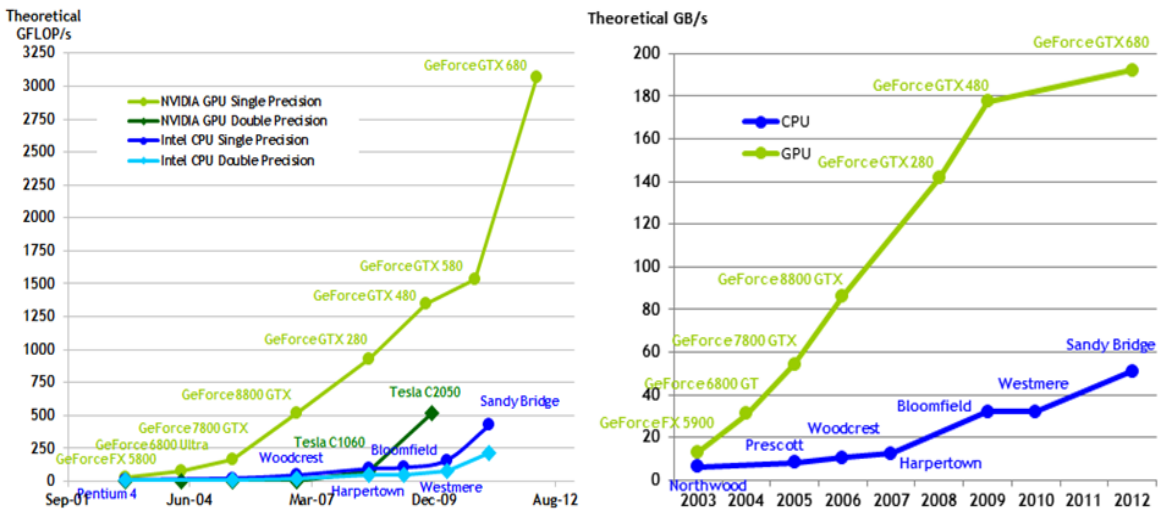

Figure 2: GPUs and CPUs speed performance and memory bandwidth. This figure is taken from [44].

Nevertheless, GPUs have not been created to replace the CPU or to make it obsolete, as they have a rather different purpose. While CPUs are designed to run serial code in an efficient way, GPUs are optimized for performing massive amounts of parallel computations, as they possess hundreds of cores capable of running thousands of threads at the same time. Thus, a heterogeneous execution model can be employed, in order to enjoy the best of both worlds, implementing the sequential algorithm parts on CPU and the parallelizable critical parts on GPU. Indeed, properly harnessed, the GPU can lead to a significant decrease of computation time when compared to a simple CPU implementation, providing huge computational performance for a relatively low price when compared to conventional supercomputing platforms such as CPU clusters.

However, not all the algorithms are suitable for this type of model, since the possible performance improvement is dependent on several factors, as the ratio between sequential and parallel parts of the algorithm, number of global memory accesses, branching, threads synchronization and data transfer overhead.

2.2.1 General purpose computation on GPU

Over the last few years, attempts to utilize GPUs to improve other kind of processes, rather than only computer graphics manipulation, lead to the creation of a new research field called general purpose computation on GPU (GPGPU).

In early days, GPGPU was not as popular as it is nowadays as it required the use of shading languages as Cg, High level shading language or openGL, using its texture units in order to map data and operations, greatly limiting the access to the hardware capabilities, which was not fully exposed or documented.

This is not an obstacle anymore, as GPUs became a fully programmable processing unit easily accessible to programmers and of more common use, due to the release of several complete sets of development tools that facilitate this type of general purpose computations.

The three mainly used frameworks for GPGPU computing are CUDA (Compute Unified Device Computing), ATI stream and openCL.

Developed by nVidia and first released in 2007, CUDA, is one of the most widely used parallel programming models. Besides being easy to program, it provides a great framework, including a compiler, CUDA SDK with several code examples, a debugger and several useful libraries as CUBLAS. However, its utilization is limited to nVidia graphic cards.

The ATI stream, maybe the least used of the three, is also a vendor platform, formerly developed by AMD (formerly ATI) specifically for its own GPUs. The hardware is architecturally different from nVidia allowing a better double precision performance.

Finally, openCL is a vendor independent framework, developed by the Khronos group, which can be used for parallelizing both CPUs and GPU and also supports both major GPU vendors' architectures.

Although in terms of portability, openCL is far better, when it comes to performance, CUDA outperforms on the majority of the cases, as can be seen in *Bhuiyan et. al.* [47], where several spiking neural networks were compared using both technologies.

According to *Fang et. al.* [48] and *Arai et.al.* [49], openCL can obtain a similar performance if the kernel codes are optimized and tuned properly, nevertheless, in other comparison work, *Dickson et al.* [50] concluded that CUDA, besides being consistently faster that openCL, also performed better when transferring data to and from the GPU. This study also showed that porting a CUDA kernel to an openCL kernel when using NVIDIA tools required minimal modifications.

In this work we focus our GPU parallelization efforts on the CUDA programming model for nVidia GPUs, due to the advantages presented in literature and to the large developer community.

2.2.2 CUDA architecture and model

In terms of architecture, the nVidia graphics card consists of a number of streaming multiprocessors (SM), each one composed several stream processors (SP), also known as cuda cores, and a local memory, shared by all SPs. The new KEPLER architecture introduced new SMs with 192 SPs and 64KB registers that can be configured between shared memory and L1 cache [51].

From the programming point of view, every program as two different parts, the code run on CPU (host) and a kernel code run on GPU (device) in a parallelized manner using a Singe Instruction Multiple Thread (SIMT) programming model. This means that each thread running on the GPU executes the same instruction at the same time, but operates on different data. Nevertheless, threads are allowed to diverge at a performance cost. When this occurs for instance due to an "if" statement, creating code branches, the threads that diverge become inactive until the compliant threads complete their separate execution and finally merge again.

Besides, CUDA allows defining the number of threads that are going to be execute on the device, which is normally several orders of magnitude greater than the number of processing units available to process the program. This allows the GPU to hide the latency of communicating with the device memory, by quickly changing active threads until the memory request completes.

Threads executed on GPU are organized in three-dimensional structures called blocks, which can hold a maximum of 512 threads. The blocks are further arranged in a two-dimensional grid layout, having a unique id. Thus, each thread can be uniquely identified by its block and its location inside it, so that, each can operate on its own data. Note that each block is executed in warps of 32 threads, inside a unique SM, and cannot be split.

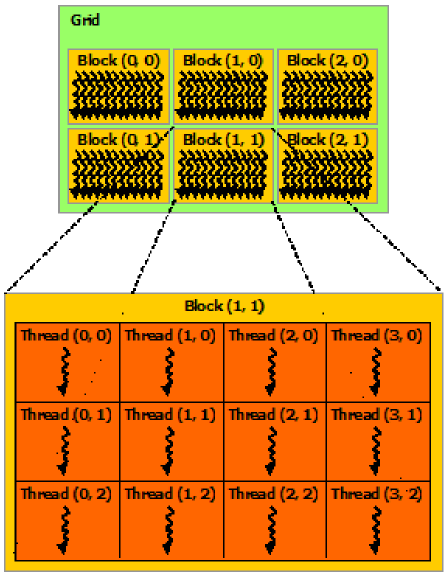

Figure 3: CUDA thread organization. This figure is taken from [44].

Regarding the memory organization, the device has four main types of memory organized in a hierarchy and as usual upper levels provide much larger storage but with slower access times, while lower levels are smaller and faster.

In the lowest level we have local memory, which are fast non-shared registers, i.e. only accessible within a specific thread. Then each block as shared memory, that can provide communication between threads inside the same block. Another important memory space is the constant memory, which only allows reading operations. Finally, in the highest level there is a large but very slow global memory, accessible by all the threads.

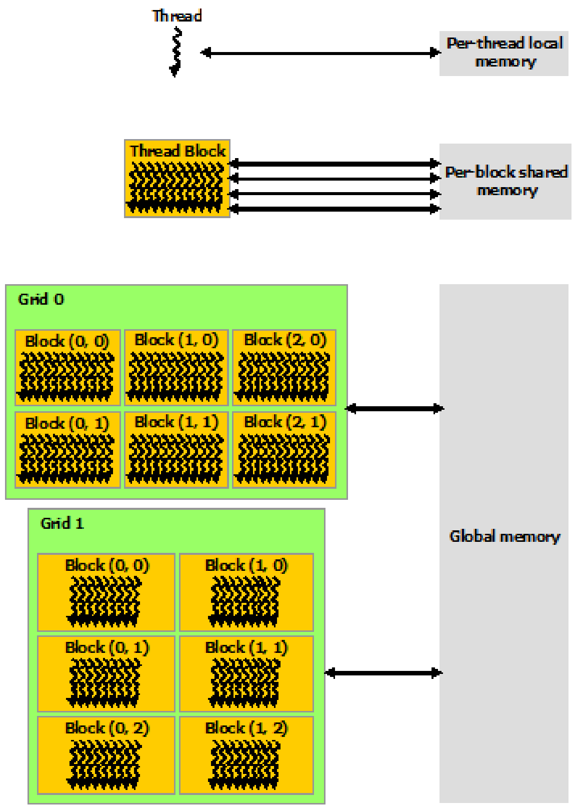

Figure 4: CUDA memory hierarchy. This figure is taken from [44].

2.2.3 GPU usage on forecasting methods

This section presents a brief reference to several papers that document the use of GPGPU for speeding up some of the algorithms that can be applied on wind power forecasting and which were previously addressed.

In Chang et.al. [52] as well as in P. Trebatický and J. Pospíchal [53] a Kalman filter is implemented on GPU, providing high speedups in matrix operations. The obtained results show improvements of 6 times over the one threaded implementation, in the best case.

Parallelized implementations of artificial neural networks have been widely documented over the last years as it is a very suitable algorithm for parallelization. C. E. Davis [54] showed the potential of GPU on several ANNs, comparing its results with the ones obtained on CPU. Moreover, several papers evaluated the computational performance improvements of a parallel implementation of back-propagation learning algorithm, as in F. Madera-Ramírez et. al. [55], where this approach was tested with two data-sets obtaining speedups of the order of 46 times to 63 times for neural networks with one hidden layer. Another case was presented in S.Cumani et. al. [56] where this method was also tested for training acoustic models for speech recognition, obtaining a 6 times training time reduction. Other GPU implementations of ANNs were also documented showing very interesting results [57, 58, 59].

Other algorithms that can benefit from parallelization are random forests and SVMs. T. Sharp [60] and D. Slat [61] implemented a random forest algorithm on GPU, obtaining improvements on the training and evaluation of decision trees of around 100 times over the CPU version and 9.2 times over Librf (random forests library).

In another paper, T. Do and V. Nguyen [62] implemented a parallelized SVM on GPU with the aim of gaining high performance at low cost. Results showed that the parallel algorithm was 65 times faster than the CPU version and over 1000 times faster than LibSVM (SVM library). In another study, V. Nguyen et. al. [63] obtained improvements of about 70 times over the sequential counterpart and 100 times over LibSVM. Other SVM implementations on GPU were documented, also obtaining very interesting results [64, 65, 66].

In [67], a basic KDE was optimized using a CUDA implementation, being tested for different kernel configurations, obtaining performance speedups of more than 2000 times the CPU implementation. However, it only implements a basic version of the KDE, thus being not adequate for the problem proposed in this thesis, as a series of extensions need to be implemented as will be later explained in chapter 3. Thus, although some of the computations are equivalent, the complete process cannot be compared.

Chapter 3

Prediction Model

In chapter 2 some models based on kernel density estimators (KDEs) were introduced and used to compute a probability density function as response to the necessity of probabilistic forecasts. This method presented good results, although having a major drawback, huge computational requirements, especially for large data sets. However, when analyzed, it proved to have a great potential for parallelization. For these reasons, this method was further studied and used as the basis for the proposed estimator.

This chapter describes in more detail the forecasting problem and the used methodology based on the KDE, identifying its limitations and proposing some extensions that can reduce them.

A table for quick reference of the parameters used to describe the method is presented in annex A.

3.1 Forecasting problem

The proposed forecasting problem consists of estimating an expected point value and a probabilistic density function (PDF) for each time-step ahead $t+k$ of a certain horizon, given a learning set D_t with N samples composed by pairs of (X_t, y_t), summarizing all the past information known up to time t. Each pair consists of a set of explanatory variables X_t (e.g. wind speed and direction in case of wind power forecasts) and the correspondent measured value of the variable to predict y_t (wind power). Additionally, we assume that at time t a set of explanatory variables, x (normally variables resulting from NWP models), is also known for each time-step ahead we want to evaluate.

This problem can be expressed as the conditional probability [40] of the variable to predict y for time $t+k$, knowing D_t and a given x for the instant $t+k$:

$$f_{y_{t+k}|D_t}(x) = \frac{f_{y_{t+k},D_t}(x)}{f_{D_t}(x)} \quad (3.1)$$

Since the actual expressions $f_{y_{t+k},D_t}(x)$ and $f_{D_t}(x)$ are not known and only discrete samples are available, the density functions are estimated using a well-known non-parametric technique called kernel density estimator (KDE). The KDE methodology is described below.

3.2 Kernel Density Estimator (KDE)

The proposed forecasting problem consists of estimating an expected point value and a probabilistic density function (PDF)

As shown in equation 3.2, the kernel density estimator returns a smoothed density estimation, obtained through the sum of the results of a function K, known as kernel function, that computes the contribution to the density of each data sample X_i.

$$\hat{f}_{D_t}(x) = \frac{1}{N \cdot h} \sum_{i=1}^{N} K\left(\frac{x - X_i}{h}\right) \quad (3.2)$$

Where, x is the explanatory variable of the evaluation point, N is the number of historic data samples and h is the bandwidth, which controls the smoothing of the estimation.

When extending this process to a multivariate case, i.e. using multiple explanatory variables, the total contribution of each sample u is obtained simply by computing the multiplicand of the kernel results for each variable.

$$K(u) = \prod_{j=1}^{d} K(u_j) \quad (3.3)$$

Applying this to equation (3.2) results in:

$$\hat{f}_{D_t}(x) = \frac{1}{N \cdot h_1 \cdot \ldots \cdot h_d} \sum_{i=1}^{N} \prod_{j=1}^{d} K\left(\frac{x_j - X_{ij}}{h_j}\right) \quad (3.4)$$

In this process two parameters need to be defined: the kernel function and the bandwidth. Concerning the kernel selection, it is possible to use different kernels for each variable as presented in [41], although it is referred in [36] that the kernel selection has minor impact on the estimate's quality. In this thesis, we simplified this step and a simple Gaussian kernel was selected for all the variables, being defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} \quad (3.5)$$

By contrast, the smoothing parameter h has a relatively strong influence on the obtained results [68]. In order to improve the model results, a simple optimization method was used to estimate the best bandwidth for each variable. This method that minimizes the squared deviation between forecast and measured values for historical data will be detailed below.

3.3 Model Formulation

Our purpose is to compute, based on historic data and future explicable variables, the forecast of given variable in the form of a distribution expected value as well as a future conditional probability density. In this section our model is detailed based on the presented KDE, first for deterministic and then for probabilistic forecasting, detailing the modifications to the original model.

3.3.1 Point forecast

For this type of forecast we have to estimate the expected value for the variable y at time $t + k$, resulting from the function $f_{y_{t+k}|D_t}$ (x) presented in (3.1). This can be achieved using a Nadaraya-Watson (NW) kernel regression estimator [69], defined as:

$$f_{y_{t+k}|D_t}(x) = \sum_{i=1}^{N} w(x, X_i) \cdot y_i \quad (3.6)$$

Where,

$$w(x, X_i) = \frac{K\left(\frac{x - X_i}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{x - X_i}{h}\right)} \quad (3.7)$$

This method estimates y_{t+k} as a weighted average of the historic observed data of y, using the kernel K as the weighing function, which calculates the contribution of each group of explanatory variables X_i for the estimation. Indeed, this weighing function estimates a point forecast value based in past combinations of (X_t, y_t), giving more importance to past y values that have been obtained under similar conditions to those forecasted for $t + k$ (explanatory variables x). The variable x is, in this case, a NWP, which can lead to an amplification of the error obtained by the model, as already referred on chapter 2.

3.3.2 Probabilistic forecast

In the probabilistic prediction case, instead of getting an array of points representing the group of expected values for the set of different time-steps ahead, a PDF is computed for each $t+k$, obtaining a distribution representing the relative probabilities for the realization of different values of y, given the explanatory variables x, therefore obtaining a prediction uncertainty.

Introducing the value y and making a small adaptation to the NW estimator (3.6) we can obtain a PDF:

$$f_{y_{t+k}|D_t}(x,y) = \frac{1}{h_y}\sum_{i=1}^{N} w(x,X_i)\cdot K\left(\frac{y-y_i}{h_y}\right) \quad (3.8)$$

Here, K is the Gaussian kernel function and h_y is a smoother used to control the smoothing of the resulting distribution.

This equation is then solved for several different possible values of y, which are calculated by dividing the difference between maximum and minimum possible value by a given number of bins, obtaining a histogram that is then normalized in order to obtain an accumulated probability function that sums one.

This normalization can be translated into the following equation:

$$f_{y_{t+k}|D_t}(x,y) = \frac{\sum_{i=1}^{N} w(x,X_i)\cdot K\left(\frac{y-y_i}{h}\right)}{\sum_{j=1}^{B}\sum_{i=1}^{N} w(x,X_i)\cdot K\left(\frac{y_j-y_i}{h}\right)} \quad (3.9)$$

Where B is the selected number of bins and $y_j(j=0\ldots B)$ is the value of each bin, representing a possible output.

This equation can be simplified in order to reduce the number of operations needed to obtain the normalized PDF, as follows:

$$f_{y_{t+k}|D_t}(x,y) = \frac{\sum_{i=1}^{N} K\left(\frac{x-X_i}{h}\right)\cdot K\left(\frac{y-y_i}{h}\right)}{\sum_{j=1}^{B}\sum_{i=1}^{N} K\left(\frac{x-X_i}{h}\right)\cdot K\left(\frac{y_j-y_i}{h}\right)} \quad (3.10)$$

After obtaining this discrete distribution, an approximation to a continuous parametric PDF is made as described below.

3.3.3 Fitting a discrete PDF to a beta distribution

After obtaining the PDF using the methodology described in the previous section, an approximation to a parametric distribution is made, as presented in [36], where the authors used a methodology called PIT to approximate the discrete PDF to a uniform one. However, in our case, the approximation is done with a beta distribution, which was found to be a good approximation for modeling variables with minimum and maximum limits as wind power [70], and is defined by:

$$f(x, \alpha, \beta) = \left(\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B[\alpha, \beta]} \right) \quad (3.11)$$

Where B[α,β] is a normalization constant to ensure that the total probability integrates to one, $0 \leq x \leq 1$ and shape parameters $\alpha > 0$ and $\beta > 0$.

The performance of this approximation can be observed in figure 5 that shows three different usual cases of the beta distribution fitted to wind power forecasting histograms resulting from KDE technique. As we can see this is a good parametric distribution to model this type of variables.

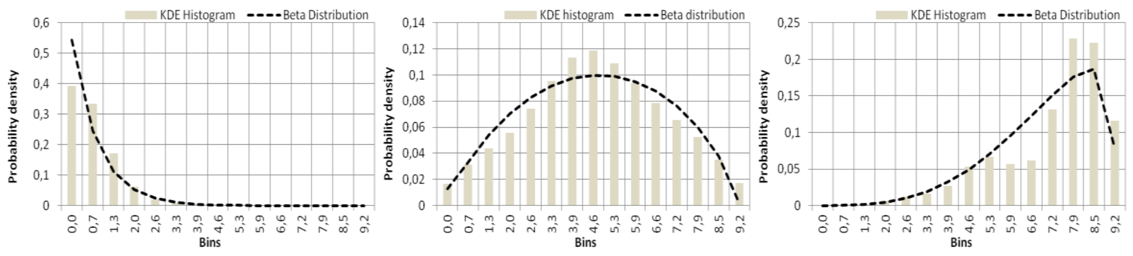

Figure 5: Wind power PDF fitted to a beta distribution.

This distribution fitting, besides allowing the evaluation of the distribution in a continuous way, also makes possible to approximately simulate the obtained distribution using only two parameters, in this case alpha and beta, thus not being necessary to keep all the calculated values of the initial discrete distribution.

The parameters alpha and beta, which have to be calculated for every time-step of the forecast, are related to the mean and variance of the distribution, thus being possible to obtain them using the equations also presented in [70], shown below:

$$\alpha = \mu \left(\frac{\mu(1-\mu)}{\vartheta} - 1 \right) \quad (3.12)$$

$$\beta = (1-\mu) \cdot \left(\frac{\mu(1-\mu)}{\vartheta} - 1 \right) \quad (3.13)$$

Where, μ is the histogram sample mean and ϑ its sample variance.

This is the main methodology which is going to be implemented in the chapter 4.

3.4 Extensions to the original model

Although the basic methodology, as described above, seem to behave as expected, it presents a noticeable limitation in the intended application context, namely when it comes to handling circular variables, which sometimes can have significant impact in forecasting results. To overcome this drawback, we propose an extension to the method in order handle this kind of variables in a more correct way.

We also discuss the possibility of introducing other improvement to the initial algorithm, which basically consists on giving different importance values to different past observations.

These two extensions will potentially improve the quality of the forecasts and are described in more detail in the following sections.

3.4.1 Circular variables

This improvement was introduced because circular variables must be handled differently from linear ones. A variable is considered circular if it is bounded between two values, being the last value equivalent to the first one. For instance, the wind direction that varies between 0 and 360 degrees or the hours of the day that varies between 0 and 24 being the 24th hour correspondent to the initial hour, i.e. the same as having the hour 0.

There are many situations in which sample values are not given the due importance for the estimation of the density, because linearly they are distant from the explicatory variables, although when treated circularly they are adjacent, being rightly considered to the estimation of the density.

As an example, we can consider a wind forecasting example. Assuming a sample of past wind direction with value 4 ^{o} and an expected wind direction for the time to be forecasted of 357^{o}, we are able to recognize that, when handling the values as circular, the sample is going to contribute with a greater importance to the estimation than the obtained when treating it as a linear variable.

Thus, a change was introduced to the initial base methodology, in order to obtain this effect when calculating the contribution of this type of variables to the forecast.

The introduced modification consists in mapping the variable into a trigonometric circle. Then, the bandwidth is summed and subtracted to the given variable, achieving the angles represented in figure 6.

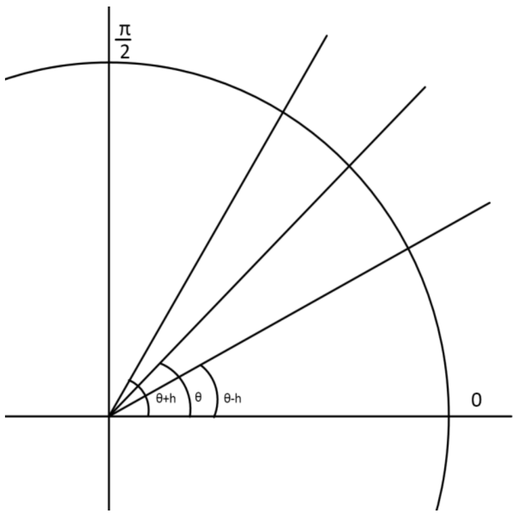

Figure 6: Figure Circular variable calculated angles.

As we can see there is a great range of possible values between angles [θ-h, θ+h] that should contribute to the estimation of the density. So, in this case, we calculate the range of values of sine and cosine that combined, results in angles that should contribute to the density as represented in figure 7:

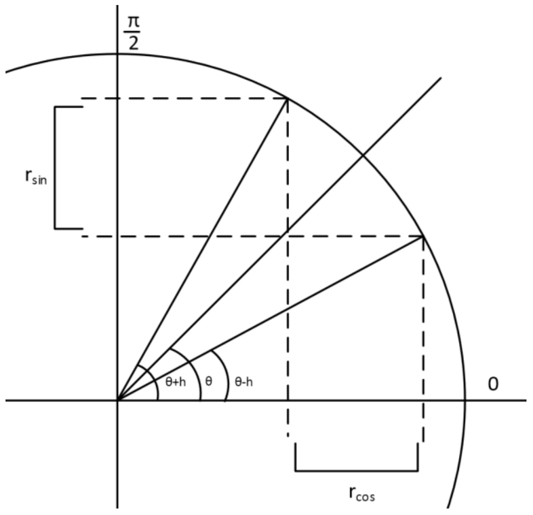

Figure 7: Circular variables bandwidths calculation.

With this methodology we can construct two different Gaussian kernels centered in $\cos\theta$ and $\sin\theta$ with a value of bandwidths of $rcos/2$ and $rsin/2$ respectively. This to kernels results are multiplied to obtain the final contribution result.

This transformation can improve results especially in cases where a circular explanatory variable value is close to the boundary. Therefore, the benefits achieved by using this extension are highly dependent on the data sets used, as a low percentage of values near the limits will result in a small improvement.

3.4.2 Weighed historical data

Another alteration to the algorithm is proposed with the aim of improving the weighing function and thereby the quality of the model results. Basically the main idea is to weigh the past knowledge, giving more importance to samples that yield better results when forecasted. This additional sample weight is calculated before the actual algorithm for each data sample, D_i and then used as an additional input in the actual forecasting computation.

This weight is inversely proportional to the error and is calculated removing the sample from the historic and using its explanatory variables to make a prediction, and then calculating the obtained error, which is used to compute the sample weigh. This expressed as an equation stands:

$$w(e) = \frac{1 - \frac{e}{\max(y)}}{\sum_{i=1}^{N} 1 - \frac{e_i}{\max(y)}} \quad (3.14)$$

Where $\max(y)$ is the maximum possible value for the variable to predict (the turbine installed power in the wind power forecast case) and the error value e for a given sample i is obtained by:

$$e_i = y_i - \sum_{j=1, j\neq i}^{N} \frac{K\left(\frac{X_i - X_j}{h}\right)}{\sum_{k=1,k\neq i}^{N} K\left(\frac{X_i - X_k}{h}\right)} \cdot y_j \quad (3.15)$$

This weight is then applied to the forecasting function, altering the sample contribution to the density for all the forecasted time-steps as follows:

$$\hat{f}_{D_t\, weighted}(x) = \frac{1}{N \cdot h} \sum_{i=1}^{N} w(e_i) \cdot K\left(\frac{x - Xi}{h}\right) \quad (3.16)$$

This addition to the method was implemented and tested, being its results presented in chapter 6.

3.5 Parameter selection (optimization)

As previously mentioned in this chapter, the selection of the variables' bandwidth parameters, i.e. the method's h variables, is very important, having a significant impact on the quality of the forecasts. Indeed, small bandwidth values lead to an over-fitted forecast, while high values could result in flat (over-smoothed) forecasts. So, in order to obtain a set of appropriate parameters to each situation, an algorithm was developed to try to improve the initial bandwidth parameters which were selected from pure observation.

The chosen method is similar to the one presented in [71] and tries to minimize the following error function:

$$e = \frac{1}{N}\sum_{i=1}^{I}\left(y_i - f_{y_i|D_{t-i}}(X_i)\right)^2 \quad (3.17)$$

$$e = \frac{1}{I}\sum_{i=1}^{I}\left(y_i - \sum_{j=1,j\neq i}^{N}\frac{K\left(\frac{X_i - X_j}{h}\right)}{\sum_{k=1,k\neq i}^{N}K\left(\frac{X_i - X_k}{h}\right)}\cdot y_j\right)^2 \quad (3.18)$$

As previously stated, the main idea is to minimize the squared deviation between forecast and measured values of historical data. Thus, we predict, for each historic sample (X_i, y_i), the expected value y using the variables X_i as explanatory variables and leaving out the ith sample contribution to the estimate.

A basic hill climbing technique was used to make a local search for a better group of smoothing parameters. In each iteration of the algorithm the values of bandwidth were changed, and saved if a better solution (with a smaller error e) was found.

Chapter 4

Model implementation

In this section we present two forecasting algorithms, a deterministic and a probabilistic, implementing the methodology described in the previous section. Both algorithms are presented, first, in a sequential version and then parallelized using the CUDA programming model.

The implemented forecast algorithms can be roughly divided in four steps, five in the probabilistic case:

- **Variable initialization**, where the needed data structures are created.
- **Sample activation**, which consists in calculating the importance of each sample to the forecast.
- **Sample weighing**, where the measured power output for each sample is weighed using the sample importance.
- **Normalization**
- **Beta fitting** is only applicable for the probabilistic forecast and consists of making an approximation of the obtained PDF to a beta distribution.

The parameter optimization is not considered as it is an optional pre-processing step that has to be run only once for each problem domain.

It must also be noticed that in the parallelized version of both the probabilistic and point forecast algorithms the weighing step is divided in three sub-steps, a first weight calculation followed by a reduction step and a final block result sum.

In this chapter, besides addressing the differences between the sequential and parallelized implementation, we also refer the adaptations needed when doing a point or a probabilistic forecast for each case.

A table for quick reference of the variables used in the algorithms implementation is presented in Annex A.

4.1 Sequential implementation

In this section we show how the sequential version of the algorithm was implemented. Presenting all the steps, first for the point forecast case and then the modifications needed for obtaining probabilistic results.

4.1.1 Point forecast (base implementation)

Initialization

We start by creating three main data structures that are used to hold the data used throughout the algorithm:

- A matrix of historic data consisting of N samples composed of a group of explicatory variables (var v, $v = 1 \dots d$) and the corresponding measured output p.
- A matrix of I inputs composed of a set of explanatory variables obtained for the forecasting point we want to predict.
- An array of bandwidths, one for each variable v.

Sample activation

After completing the first step, we can start the sample activation. This step is performed for each input, and consists of iterating through the samples of the historic matrix, H, and its variables, calculating the importance of each one for the forecast of that input.

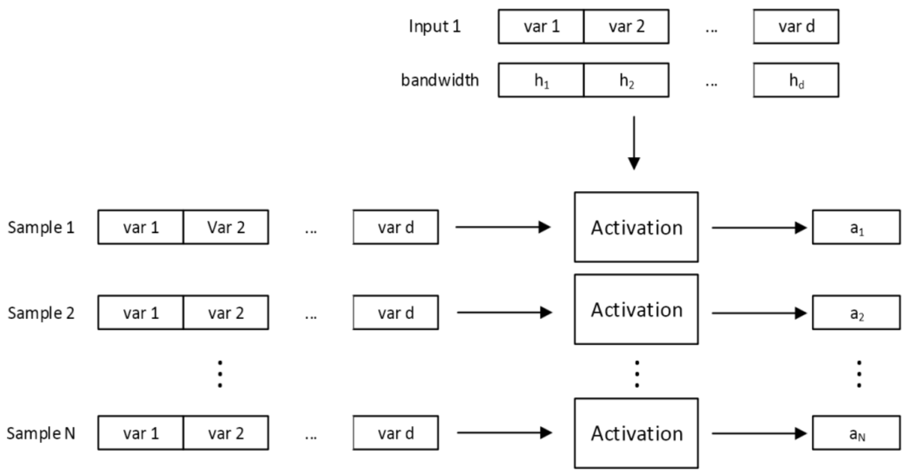

Figure 8: Sequential sample activation step.

This sample importance is calculated using the kernel K, in this case a Gaussian distribution, centered in the input variable value and with a deviation h. This kernel function is calculated for each sample variable using the correspondent bandwidth value as the distribution deviation. The results are aggregated by multiplying every variable contribution obtaining the final sample activation value a_i. A simple diagram presenting this step is shown in figures 9 and 4.1.3.

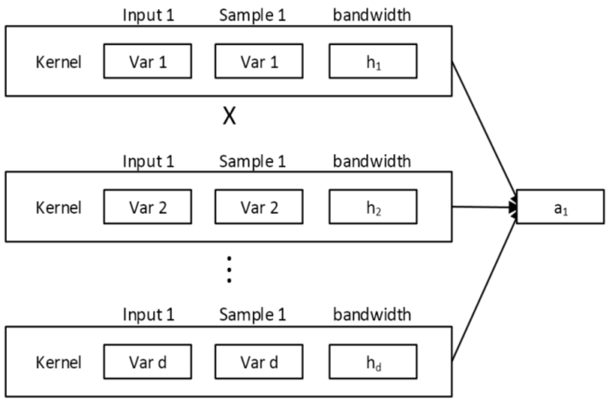

Figure 9: Activation calculation.

This step is the translation of the equation (3.4), and is presented in algorithm 1 in pseudo-c.

Algorithm 1: (Sequential) Sample activation

```
//for each input
for i=0 to I-1

    //for each sample
    for j=0 to N-1
        float activation = 1.0;

        //for each variable
        for k=0 to d-1
            activation *= K(H[j][k], I[i][k], h[k]);
```

Sample weighing

With this activation we can now start the weighing step, which can actually be computed inside the same for loop as the activation. It consists of weighing each sample power output value, referred here as w, which is obtained by multiplying the p value of each sample by the calculated activation of that same sample.

This is presented in figure 10 in a graphical manner.

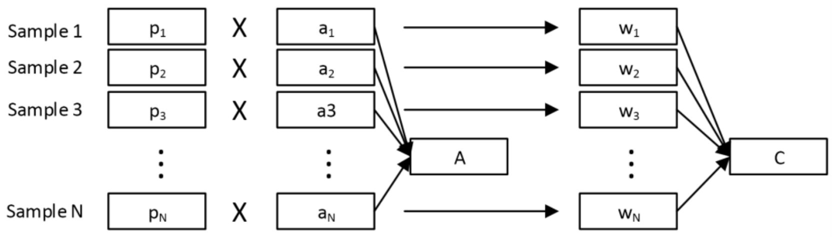

Figure 10: (Sequential) Deterministic sample weighing step.

As we can perceive from this diagram, in every iteration of the loop that goes through the historic samples, the values of the obtained activation, a, as well as the values of the calculated w are being accumulated in two different variables, A and C respectively representing the sum of all the activations and the sum of all the weighed p values, w.

Normalization

Finally, in the last step, to obtain the expected point forecast of a given input, the variable C_i divided by A_i, being the resulting value stored in the corresponding index of the final array of expected values.

The additions to the algorithm introduced in the last two steps are presented in algorithm 2 highlighted.

Algorithm 2: (Sequential) Determinitic forecast

```
//for each input
for i=0 to I-1
    float colapse = 0.0;
    float sumActivation = 0.0;

    //for each sample
    for j=0 to N-1
        float activation = 1.0;

        //for each variable
        for k=0 to d-1
            activation *= K(I[i][k], H[j][k], h[k]);

            colapse += activation * H[j][d+1];
            sumActivation += activation;

    point[i] = colapse / sumActivation;
```

4.1.2 Probabilistic forecast

Initialization

In the probabilistic version of the base algorithm several different values of possible outcomes, referred as bins, are used to compute the relative probabilities of its actual realization. This implies the need to define some additional variables in the initialization step:

- A bandwidth h_p, which is added to the bandwidths vector.
- The limits of the variable to predict (max and min).

- The number of bins B in which the range of the variable to predict is going to be divided.

bandwidth
h1
h2
...
hd
hp

Figure 11: Probabilistic bandwidth initialization.

After defining these variables we need to calculate the bin values. As every bin is equally separated from one another, we only have to calculate the difference between each one, which is defined by:

$$step = (max - min)/(B - 1)$$

This allows iterating through the bin values without the need to create a new array to store them, since it is possible to obtain the values of v_b, by multiplying the index of the bin we want, b, by the step and summing it to the minimum limit, min.

$$v_b = min + b \times step$$

Sample activation

The second step proceeds exactly in the same way as in the basic implementation, thus we will move on to analyze the changes in the following steps.

Sample weighing

As can be observed in figure 12, now instead of having one weighed p for each sample, we have a weight, w, for each bin for each one of the tested samples. These values are added to a density vector that accumulates all the bin weights obtained in each sample iteration for a given input.

Unlike the base implementation, the calculation of the w values is obtained by multiplying the sample activation by a kernel function, which in this case is the same Gaussian distribution used in the activation step, although centered in the historic measured value p with a standard deviation h_p, and thereby calculating the contribution of each sample to the relative probability for the realization of each bin value.

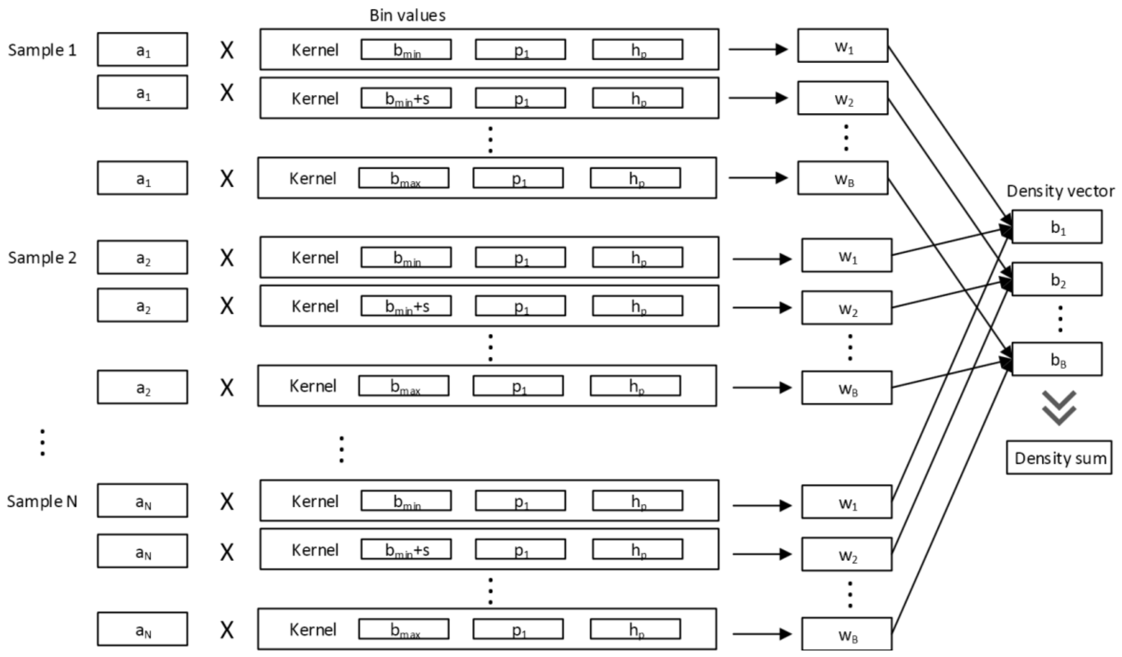

Figure 12: (Sequential) Probabilistic sample weighing step.

Normalization

After obtaining the complete density vector for one input iteration, the obtained values stored in the density vector have to be normalized. This is performed by dividing each one of this density values by the sum of all the elements of the vector, which result in a normalized discrete distribution.

It is noteworthy that the density values are not divided by the sum of all the activation values, as in the base implementation, due to the equation simplification discussed above and defined by equation (3.10).

The complete method is translated to the algorithm 3 presented below.

Beta fitting

After obtaining this normalized discrete distribution an approximation to a beta distribution is made, which consists in calculating the alpha and beta parameters that shape the distribution.

These two values can be obtained from the equations (3.12) and (3.13). Thus, for each input the mean and variance of the distribution are calculated and then used as arguments to solve both equations, being the resulting values stored in a matrix with *I* lines and 2 columns, for the correspondent alpha and beta values.

Algorithm 3: (Sequential) Probabilistic forecast

```
//for each input
for i=0 to I-1

    //density sum
    float collapse[B];
    float sumBin = 0.0;

    //for each sample
    for j=0 to N-1
        float activation = 1.0;
        float density = 0.0;

        //for each variable
        for k=0 to d-1
            activation *= K(I[i][k], H[j][k], h[k]);
            float bin = minBin;

            //for each bin
            for m=0 to B-1
                w = activation * K(bin, H[j][d+1], h[d+1]);
                collapse[m] += w;
                sumBin += w;
                bin += step;

    //for each bin, normalization
    for m=0 to B-1
        pdf[i][m] = collapse[m] / sumBin;
```

4.2 Parallelization on GPU

In this section we show how the probabilistic version of the algorithm was implemented. First we present how the CUDA kernel was structured, and then how all the steps were implemented both in the point forecast as well as in the probabilistic one.

As previously pointed, in the parallelized version of both the probabilistic and point forecast algorithms the weighing step is split in three sub-steps, a weigh calculation followed by a reduction step and finally a block results sum.

4.2.1 Kernel structure

When analyzing the diagrams of the sequential implementation, presented in the previous section, it is easy to understand that the computations performed for each input are completely independent, being a possible starting point for our parallelized implementation process. However, with a closer look, we may see that it is possible to take best advantage from the GPU capabilities as the activation and weighing steps, presented above, can be performed individually by an independent thread without the need to wait for other computations, thus being suitable for parallelization.

As discussed in section (4.1.1) and clearly shown in algorithm 4.1.2, we can observe that the sequential algorithm is composed of two main 'for' loops: an outer loop that iterates over the inputs and an inner loop that covers all the historic samples, in which the activation and weigh are calculated, independently of any other data. Thus, for implementing this forecasting algorithm in GPU we can organize our kernel as shown in figure 13 displayed below.

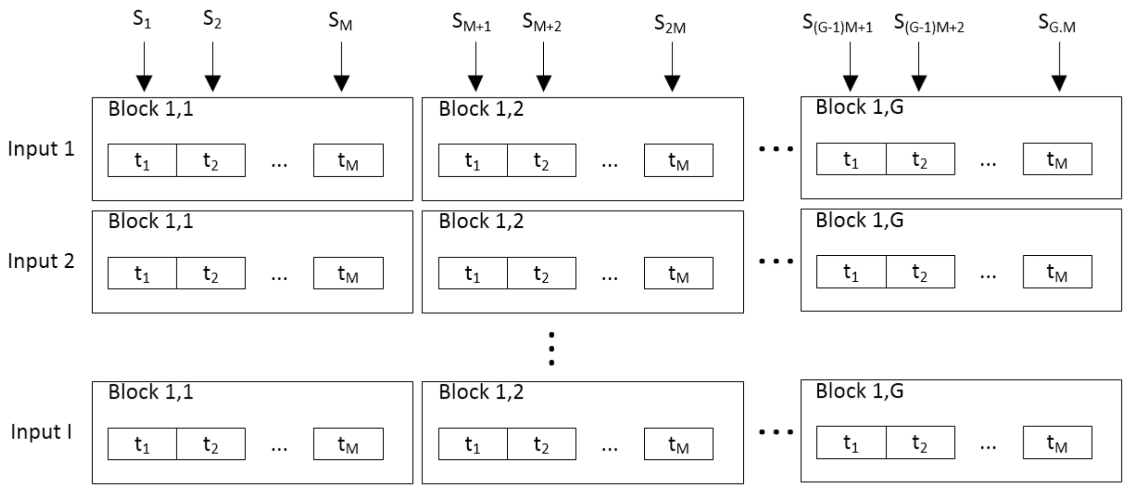

Figure 13: Kernel structure.

Looking at this diagram it is possible to see that our kernel consists of a two-dimensional grid partitioned in one-dimensional blocks, equally divided in several threads. Each one of these threads is responsible for the computation of the activation step and the first two weighing sub-steps, relative to a given input and historic sample. Indeed, as we can see, the grid lines are considered for each input data while the columns for each sample. For this reason an iteration of the outer loop in conjunction to one of the inner loop is replaced by an independent thread, with the identifier (idx, idy) on the grid of threads.

The number of threads per block has to be previously defined, normally to a multiple of 32 as this is the warp size, i.e. the maximum number of threads that may be executed concurrently inside a SM as explained in section 2.2. The number of blocks per grid line is obtained as the ceiling of the division of the number of historic samples N by the number of threads per block, M. It is noteworthy that when the number of samples is not a multiple of the block size, which is normally the case, there are a group of threads that become idle during the execution.

4.2.2 Point forecast

Initialization

In this case the variables initialization is similar to the sequential implementation, although, a small difference may be noticeable: variables initialized as two dimensional arrays in the sequential implementation are in this case created as one-dimensional, as exemplified in figure 14.

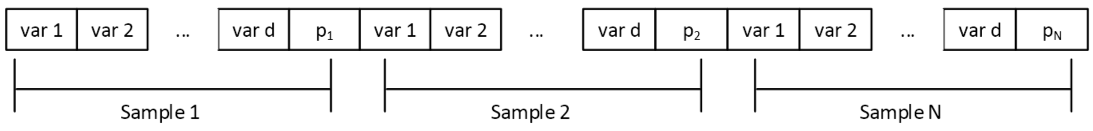

Figure 14: GPU historic initialization.

In this first phase, the needed memory space is allocated both on the host and on device, and the data is copied from the former to the latter.

After this initializing step the kernel function is called, being its instructions computed for each thread, which one is responsible for the following two steps of the algorithm.

Sample activation

The activation step proceeds similarly to what occurs in one iteration of the sequential algorithm activation, i.e. for a given input and historic sample as can be observed in figure 15. The resulting activation value is then stored on shared memory, based on the thread index inside the block, referred as x.

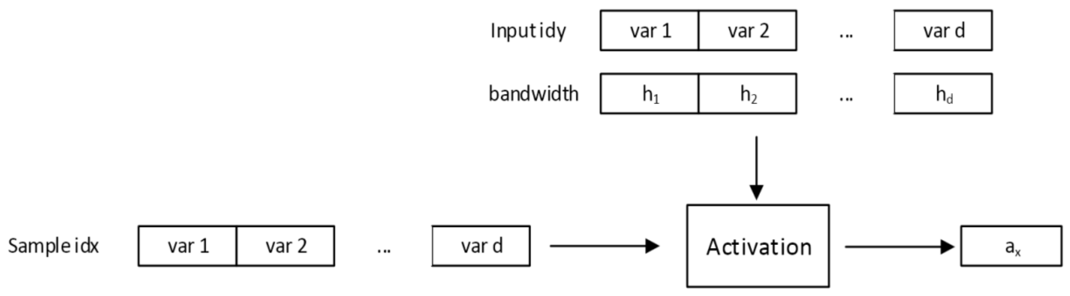

Figure 15:Thread sample activation step.

This step is also followed by a weighing step, which in this case is divided in three sub-steps, as already referred.

First weighing sub-step

This first sub-step is responsible for the w value calculation, i.e. the weighed p of the given sample, idx for the input being analyzed idy. As is the case for the activation values, the obtained w is then stored on shared memory, so that it is possible to access it from other threads inside the block, which is relevant for the following sub-step. This process is represented in figure 16 for a given thread with grid indexes of idx and idy.

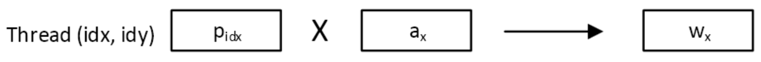

Figure 16: Thread's first weighing sub-step (Point forecast).

The correspondent kernel code in pseudo-c is shown in algorithm 4 below. Notice that in the presented algorithm, in order to calculate the sample start index we are multiplying idx, correspondent to the index of the sample, by $\mathrm{d}+2$, which is correspondent to the size of each sample, i.e. number of explicatory variables plus the variable p. This size is of $\mathrm{d}+1$ in the inputs array case, as no p value exists in that case.

Algorithm 4: (Parallelized) Deterministic activation and first weighing sub-step

```
int idx = blockIdx.x * blockDim.x + threadIdx.x;
int idy = blockIdx.y;
int x = threadIdx.x;

//if the thread is not idle
if idx < N
    float activation = 1.0f;
    int sIdx = idx * (d+2); //sample start index
    int iIdx = idy * (d+1); //input start index

    //for each variable
    for i=0 to d-1
        activation *= K(I[iIdx + i], H[sIdx + i], h[i]);

    cache_activation[x] = activation;
    cache_weigh[x] = activation * H[sIdx + (d+1)];
```

Second weighing sub-step (reduction)

The next weighing sub-step can only start after all the threads reach this point in the algorithm, as both shared arrays have to be filled with the obtained values of a and w in each block thread. In this step, the goal is to obtain a sum along each grid line, i.e. for each input, of all this calculated values.

Thus, this second sub-step consists in a reduction process realized within each block for the variables stored in shared-memory. This approach is frequently used to solve this kind of problem, where it is required to obtain a single element from a larger input stream [72].

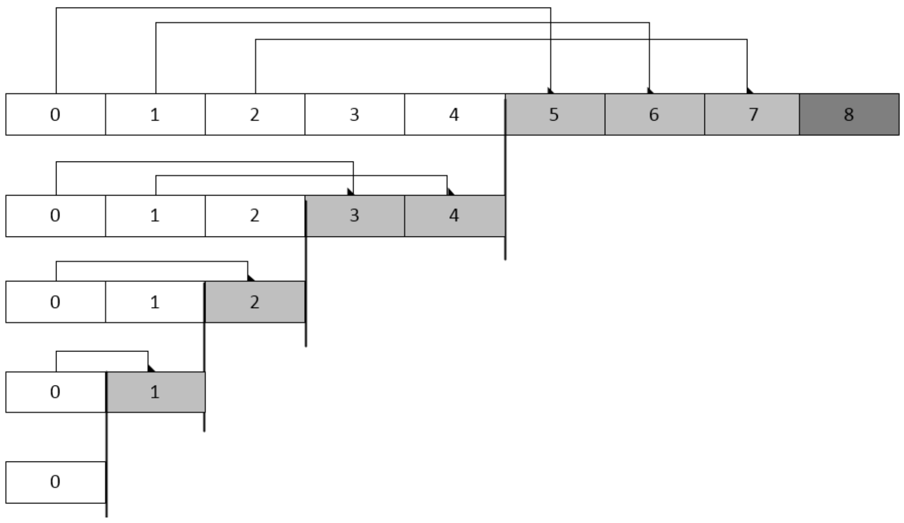

Figure 17: Reduction process (Point forecast).

A reduction process example is shown in figure 17. We can see that in each iteration the array size is cut by half, in this case plus one, as it shows an odd sized array. The threads corresponding to the first half indexes are thereby responsible for summing to its position on the shared array the element stored in the array index correspondent to its index plus the number of active threads, represented in white in the diagram.

However, as previously referred it is possible to have idle threads, which are not responsible for any historic sample and thereby do not have results stored in the array (represented as dark grey). Due to this fact, only threads that can reach positions that weren't idle during the past steps (represented as light grey) do the actual sum calculation.

Finally, the obtained results for each block are stored on global memory in an array of $I\ X\ G$ elements, being G the number of blocks per grid line. When all the blocks finish this step the results are copied back to the host device which is responsible for the final computations.

This process is presented in algorithm 5 below.

Algorithm 5: Deterministic reduction process

```
int r = blockDim.x

//while the array size is different from 1
while r != 1
    r = ceil(r / 2);

    if (idx + r < N && x < r)
        cache_activation [x] += cache_activation [x + r];
        cache_weigh [x] += cache_weigh [x + r];

    __syncthreads();

if threadIdx.x == 0
    int blockOffset = blockIdx.x + gridDim.x * blockIdx.y;

    block_weigh[blockOffset] = cache_weigh [0];
    block_activation[blockOffset] = cache_activation [0];
```

Third weighing sub-step (final block sum)

The final weighting sub-step is performed on the CPU, which is responsible for summing the final block results for each grid line, onto two variables C_i and A_i, which are respectively correspondent to the sum of all the activations and the sum of all the weighed p values, referent to the input i.

This sum is done sequentially as this operation would not benefit from parallelization, as besides involving accesses to global memory it has high thread divergence.

This process can be observed in figure 18.

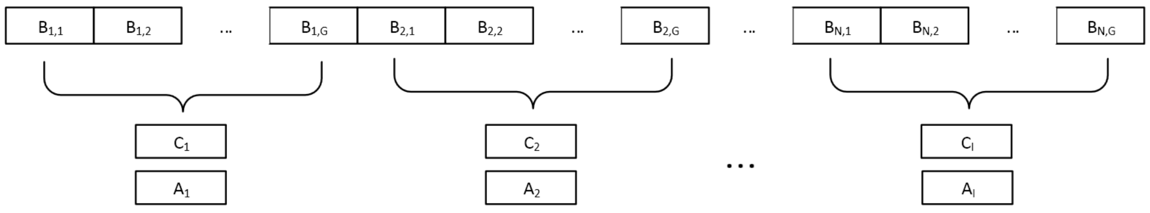

Figure 18: Final block sum (Point forecast).

Normalization

Finally, the normalization which is exactly the same as the one presented in the sequential point algorithm, where C_i is divided by A_i resulting in the point forecast for the given input, which is stored in the index i of the results array.

The last two steps are presented in pseudo-c in algorithm 6.

Algorithm 6: (Parallelized) Deterministic final block sum and normalization

```
//for each block for each input
for i=0 to (I * G)-1
    offset = i / G;
    final_activation[offset] += block_activation [i];
    weigh_colapse[offset] += block_weigh [i];

//for each input
for j=0 to I-1
    point[i] = weigh_colapse[i] / final_activation[i];
```

4.2.3 Probabilistic forecast

The implementation of the parallelized probabilistic forecast can be considered as a junction of the presented sequential version with the introduced parallelization ideas presented in the previous sub-sections.

Initialization

The variables to define are the same ones presented in the sequential version of the algorithm, although with the same modification introduced for the parallelized punctual forecast, i.e. all the arrays are one-dimensional.

Sample activation

The second step proceeds nearly in the same way as in the point parallelized implementation, being the only difference the fact that the activation values are not stored on shared memory.

First weighing sub-step

The modifications to this step are similar to the ones presented in sequential version, since, the first weighing sub-step in which a single weigh w is calculated, is replaced by a loop that computes a weigh for each bin value.

In this case, this calculation is performed in each individual thread, for a given sample and input data, as in the parallelized punctual version. This is shown in figure 19.

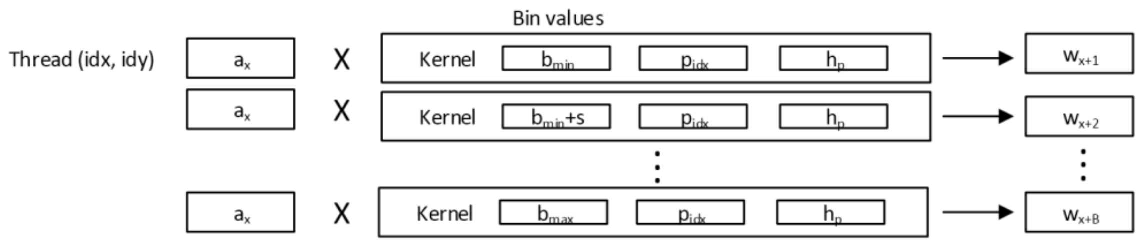

Figure 19: Thread's first weighing sub-step (Probabilistic forecast).

These obtained values of w are then stored in shared memory within each block, meaning that each thread is responsible for B elements of the shared array, which size is now $M \times B$, being M the number of threads per block. The modifications made to the kernel are presented in algorithm 7 in pseudo-c, where the modified instructions are highlighted.

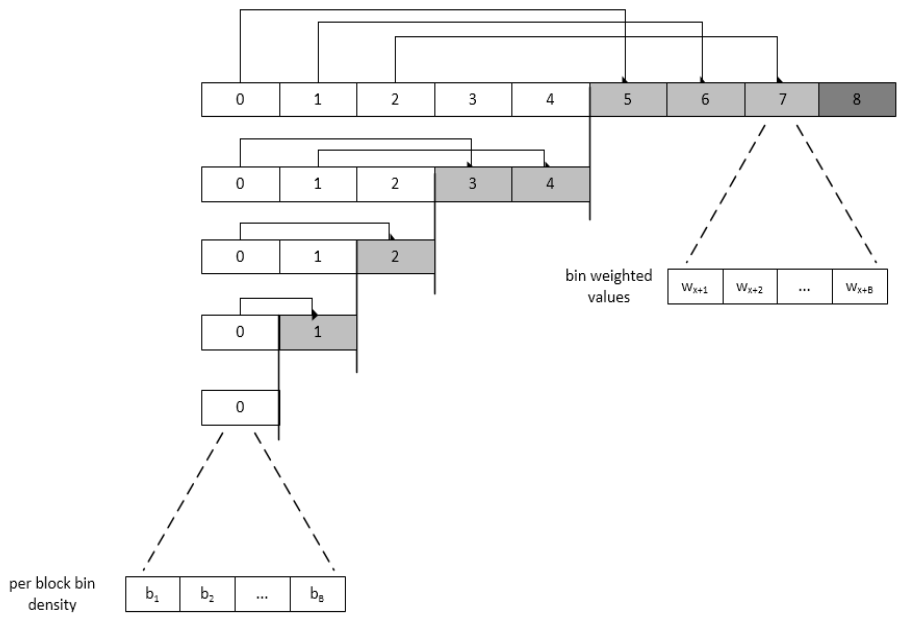

Figure 20: Reduction process (Probabilistic forecast).

Algorithm 7: (Parallelized) Probabilistic activation and first weighing sub-step

```
int idx = blockIdx.x * blockDim.x + threadIdx.x;
int idy = blockIdx.y;
int x = threadIdx.x;

//if the thread is not idle
if idx < N
    float activation = 1.0f;
    int sIdx = idx * (d+2); //sample start index
    int iIdx = idy * (d+1); //input start index

    //for each variable
    for i=0 to d-1
        activation *= K(I[iIdx + i], H[sIdx + i], h[i]);

   /**
   cache_activation[x] = activation;
   cache_weigh[x] = activation * H[sIdx + (d+1)];
    **/

    float bin = minBin;
    float sy = H[sIdx + (d+1)];
    float hp = std[d-1];

    for i=0 to B
        cache_weigh[x + i] = activation * K(bin, sy, hp);
        bin += step;
```

Second weighing sub-step (reduction)

After this first sub-step has been completed by all threads, the second weighing sub-step may start. This step is similar to the reduction step showed in figure 17, although each active thread is responsible for summing *B* elements, as displayed in figure 20 presented above. This results in the alterations presented in algorithm 8.

Algorithm 8: Probabilistic reduction process

```
int r = blockDim.x

//while the array size is different from 1
while r != 1
    r = ceil(r / 2);

    if (idx + r < N && x < r)
        /*
        cache_activation [x] += cache_activation [x + r];
        cache_weigh [x] += cache_weigh [x + r];
        */
        for i=0 to B-1
            cache_weigh[x+i] += cache_weigh [x+i+(r*B)];

    __syncthreads();

if threadIdx.x == 0
    int blockOffset = blockIdx.x + gridDim.x * blockIdx.y;
    /*
    block_weigh[blockOffset] = cache_weigh [0];
    block_activation[blockOffset] = cache_activation [0];
    */
    for i=0 to B-1
        block_weigh[blockOffset*B+i] = cache_weigh [i];
```

Third weighing sub-step (final block sum)

After summing, for each block, all the contributions of each thread, to the estimation of the occurrence probability of each bin value, the final block result is stored in an intermediate array of $B \times I \times G$ elements.

This array is thereby organized as shown in figure 21, being thereby responsible for storing the total contributions per block for a given input forecast, depending on the grid line idy where the block is located. This array is then copied back to the host device which is responsible for the sum of all the blocks results for each input.

These sum computations are similar to the ones performed in the punctual version, although replacing each element of the array by the set of values obtained for each bin.

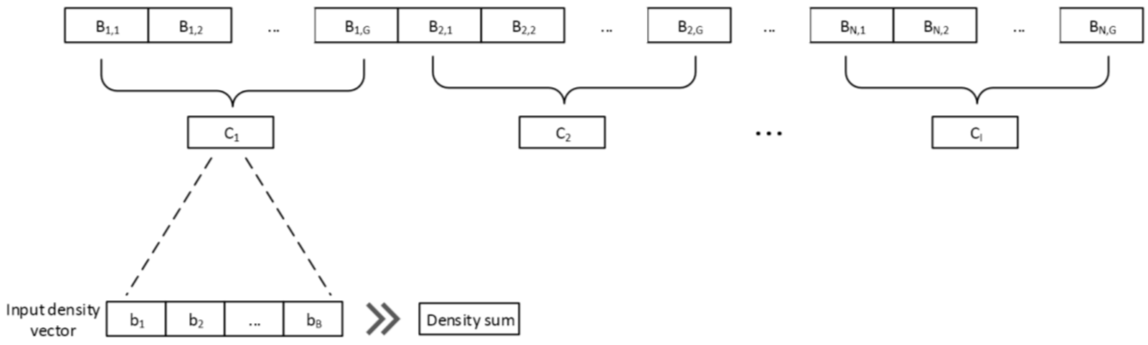

Figure 21: Final block sum (Probabilistic forecast).

Normalization and beta fitting

These two final steps are exactly the same ones performed in the sequential version of the probabilistic algorithm presented. In which each value of the discrete PDF, $b_{k,}$ of each input is divided by the density sum, which is considered as the sum of all the obtained b values for the given input. Finally each PDF has to be approximated to a beta distribution, obtaining a final matrix that stores, for each input, the values of the parameters that shape the beta distribution, alpha and beta.

4.3 Extensions implementation

In this section we present the additions that need to be made to all the presented algorithm versions in order to implement the improvements discussed in section 3.4, the use of a specific function for handling circular variables and a pre-calculated array of historic weights.

The first addition is in the initialization step, as two additional variables must be created:

- An array with the upper boundary of each variable v in the case it has to be treated as a circular variable and 0 otherwise.
- An array with a pre-calculated weight for each sample in the historic matrix.

Since both improvements only interfere in the activation step, they can be implemented exactly in the same way in the point forecast implementation as well as on the probabilistic version, since this step only as some alteration from sequential to parallel versions.

These are introduced as follows in the sequential activation algorithm 9:

Algorithm 9: Deterministic algorithm with extensions

```
//for each input
for i=0 to I-1

    //for each sample
    for j=0 to N-1
        float activation = 1.0;

        //for each variable
        for k=0 to d-1
            //activation *= K(H[j][k], I[i][k], h[k]);
            //if not circular
            if(C[k] = 0)
                activation*=K(I[i][k],H[j][k],h[k]) * W[j];
            else
                activation*=Kcircular(I[i][k],H[j][k],h[k]) * W[j];
```

contribution of variables which have their maximum limit, stored in the array C, different from 0.

Concerning the parallel version the modifications are very similar and presented below in algorithm 10.

Algorithm 10: Probabilistic algorithm with extensions

```
int idx = blockIdx.x * blockDim.x + threadIdx.x;
int idy = blockIdx.y;
int x = threadIdx.x;

//if the thread is not idle
if idx < N
    float activation = 1.0f;
    int sIdx = idx * (d+2); //sample start index
    int iIdx = idy * (d+1); //input start index

    //for each variable
    for i=0 to d-1
        //activation *= K(I[iIdx + i], H[sIdx + i], h[i]);
        if(C[i] = 0)
            activation*=K(I[iIdx+i],H[sIdx+i],h[i]) * W[iIdx];
        else
            activation*=Kcircular(I[iIdx+i],H[sIdx+i],h[i]) * W[iIdx];

    cache_activation[x] = activation;
    cache_weigh[x] = activation * H[sIdx + (d+1)];
```

Chapter 5

Experiments and results

In this chapter we present the experimental results of the implemented algorithms that were evaluated both at performance and accuracy levels.

First the problem and the available data are introduced, followed by a group of preliminary tests to the two method extensions previously discussed. After that, the results for the point forecast are presented and compared to the ones obtained by an ANN approach. Finally, the accuracy and performance results of the obtained probabilistic forecasts are presented and discussed.

5.1 Case study description

Before presenting the results, in this section we describe the available data and the conducted test sets.

The available historical data, related to several wind farms, comprises information about three meteorological variables obtained from NWPs made for one reference point inside each wind farm's site and correspondent wind farm's power output measured over time by the installed SCADA system.

These data sets are referent to the year of 2012 and consider values of air density, wind speed and wind direction as the explanatory variables, being its resolution of 15 minutes, which means that we have 4 samples per hour.

In order to quantitatively classify the methods used, two different assessments have to be considered: accuracy and time performance.

Regarding the accuracy evaluation, we used two data sets from two operating Portuguese wind farms (designated as WF-A and WF-B) located in distant geographical areas, being the former located in the northern and the latter in the southern zone of the country. Both data sets were split into a learning set, used only as historical observations, and a testing set, for which the predictions were made. Thus, the accuracy experiments were performed using a learning set of approximately one year of observations with 15 minutes time-step (33951 samples) and a set of different inputs referent to 7 horizons of 24 hours, also with 15 minutes resolution, which represent 672 forecasting points.

Although this can seem rather small quantity of data, these numbers are relative to the available data of a single point wind farm. On real operation, the algorithm is going to be used on hundreads of wind farms, some with several predictions points that are later combined. Moreover it is expected to have a continuosly growing learning set.

A different approach was taken with regard to the evaluation of the method time performance, since the data used to forecast do not have any impact on the results we want to observe. Thus, the tests were made just with data referent to the WF-B, for different sizes of historical and input sets. Moreover, in more extreme test cases, where the available data were not enough, the values were replicated in order to increase the number of data that have to be computed.

These performance evaluations were conducted using three different test sets:

The first one measures the execution time to make a forecast, using an increasing number of historical data and maintaining a fixed number of points to be forecasted and histogram bins (only in the probabilistic case) equal to 672 and 10 respectively.

The second performance test involves measuring the time for a different number of inputs, i.e. various horizons, with 15 minute resolution, varying from one day ahead (96 points) to two weeks (1344 points). In this test we keep a fixed number of historic samples equal to 35040, which corresponds to a year of samples with 15 minutes time-step and an unchanged number of 10 bins in the probabilistic case.

The last test is only applicable to the probabilistic forecast case, and consists of varying the number of bins for a rigid prevision horizon of one week and learning set of half a year of historic samples.

To get reliable test results, the execution times are presented as an average of ten runs.

Furthermore it is worth mentioning that all the experiments were conducted after making a selection of the explanatory variables' bandwidth parameters using the aforementioned method, limiting the iterations number to 2000, and using an initial set of parameters selected by pure observation of previous cases.

Moreover, all the tests were performed on a desktop with a 32 bits operating system, with 4096 MB of installed RAM and comprising a dual core processor and a GPU. The processor is an Intel Pentium dual core with 1.8 GHz clock speed. The GPU is an nVidia GeForce GT 640, with a CUDA compute capability 3.0 and 2 multiprocessors, each one with 192 cores, which results in a total of 384 processor cores, running at 900 MHz with a theoretical single precision computing power of 691 GFLOPS.

5.2 Preliminary tests regarding the method extensions

Before employing an algorithm implementation some preliminary tests were performed in order to assess the benefits and drawbacks of the utilization of the proposed model extensions. Thus, in this section we present the obtained results and explain the decisions adopted for the next tests.

In these experiments we evaluated both accuracy and time performance using the data sets of the two referred wind farms as test cases.

Regarding the utilization of the circular variables extension, results for both data sets showed slightly better results over the basic implementation. However, the improvement was rather small, presenting in average of less than 0.1% for the NMAE and NRMSE, though, when analyzing the wind direction values, it was possible to observe that there are few occasions when the values approach the boundary values, representing 1.4% and 1.6% of the cases for respectively the WF-A and WF-B, which is the most probable cause for the little impact on the prevision quality.

On these tests the algorithm processing time on GPU increased from an average of 65.1 milliseconds, in the cases where this modification has not been used, to 104.5 milliseconds when the change was introduced in the algorithm.

Although the results have not shown significant improvements for both test cases, we believe that in other instances this extension can significantly improve the quality of the forecasting results, thereby compensating the longer execution time.

Concerning the use of the weighted historic data, results showed that this optimization brings no significant improvement to the results. Despite the increase on the actual forecasting execution being a mere 3 milliseconds, its use implies a first pre-processing step that involves the execution of the forecast processing for the historic data. This is a major disadvantage since the total time needed to make the forecast itself will be considerably larger.

In conclusion, we decided that the circular variables extension should be used, since, although the algorithm performance undergoes a relatively significant increase, its use can have considerable impact on the forecasting quality. In the case of the weighted historic samples, its utilization was discarded, because the pre-processing was very time-consuming and its use did not show any impact on the quality of the results.

5.3 Point Forecast results and comparison with ANN approach

In this section we present the obtained results for the tests made for the point forecasting version of the algorithm. First the forecast's accuracy is assessed and compared with the results obtained by ANN model. This is followed by an analysis of the algorithm time performance, in which a comparison is made between the execution times of the sequential implementation and the GPU parallelized version.

With respect to the method used as comparison, it consists of a simple ANN model with 3 layers. The method receives the same three explicatory variables as inputs and returns the same type of output, correspondent to the wind farm's power production, as can be seen in figure 22 that shows the used ANN architecture.

Two different networks were created, one for each wind farm and trained with a back-propagation algorithm, using the same historic set as the one used by our model referred as KDE-NW.

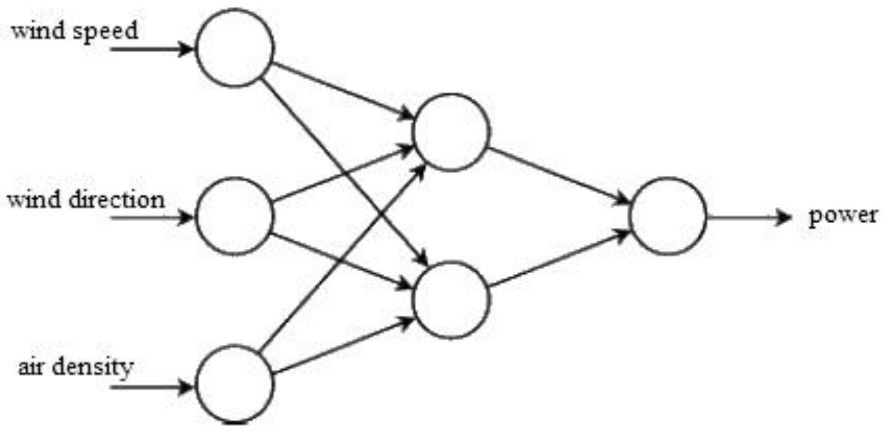

Figure 22: ANN architecture.

5.3.1 Accuracy

In terms of accuracy the obtained results were rather similar for both methods, showing no substantial differences. The results are presented in a normalized form for both wind farms, in figure 23 for the WF-A and in figure 24 for the WF-B. On both graphics it is possible to observe that the prediction values are practically equivalent for both methods, being the only noticeable difference that values obtained by the ANN for the WF-B prediction are always a little higher than the ones resulting from our model.

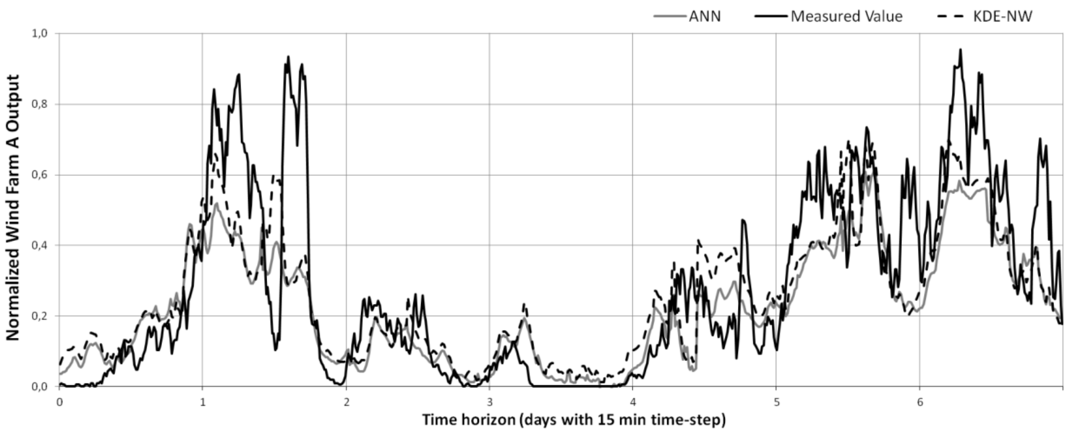

Figure 23: Point forecast for WF-A.

Two different networks were created, one for each wind farm and trained with a back-propagation algorithm, using the same historic set as the one used by our model referred as KDE-NW.

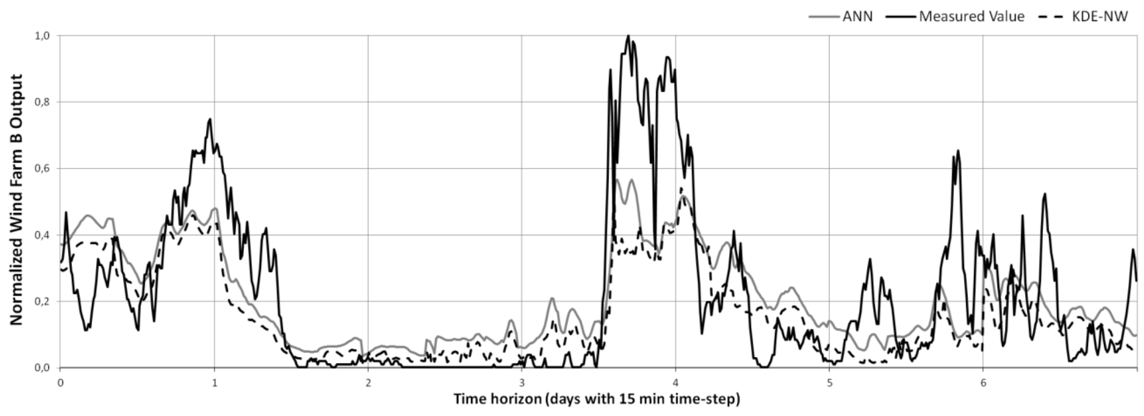

Figure 24: Point forecast for WF-B.

In order to quantitatively evaluate the forecast results two different forecasting errors were calculated: the NMAE and the NRMSE. The results for WF-A presented a NMAE of 11.3% and 11.8% for the ANN and the KDE-NW respectively, while the NRMSE was approximately 16.2% for both methods.

Similarly the results for WF-B also showed no significant divergence between the methods, while the KDE-NW showed a very small improvement regarding the NMAE, obtaining 12% against the 12.7% of the ANN, it showed worst NRMSE results, 17.9% over the 16.7% achieved by the ANN.

Indeed, no significant difference exists in the obtained errors of both methods, thus, we can conclude that in terms of forecasting accuracy there is no considerable difference between the utilization of either method.

5.3.2 Time

As previously explained two different test cases were conducted to test the time performance of the two implementations of the point forecasting algorithm, the sequential and parallelized version. This section is focused on presenting the obtained results on those tests.

On the first test case in which different sizes of historical data are analyzed by the method, we can observe in figure 25-a that the GPU implementation could achieve a significant acceleration of the process, being more than 295 times faster than the sequential version. It must be noticed that the axis representing the time is in a logarithmic scale.

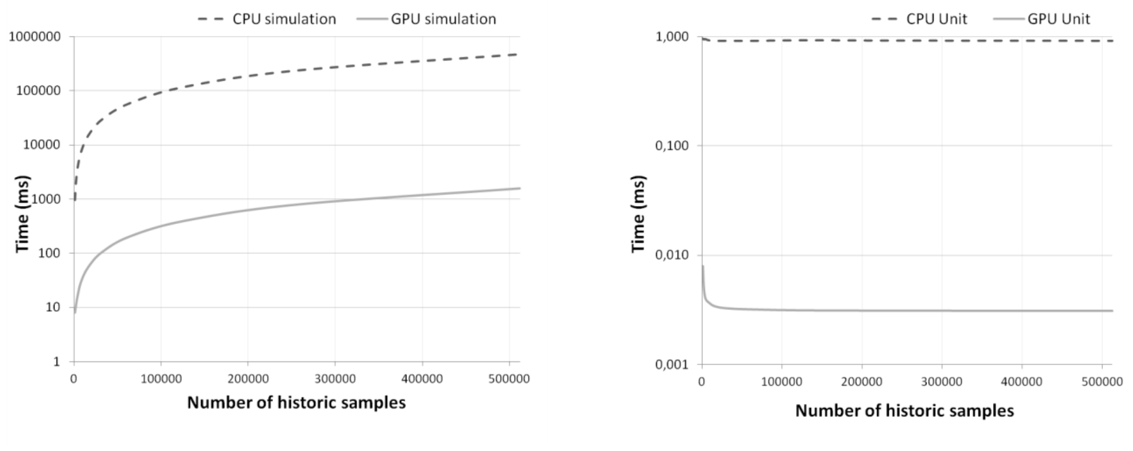

Figure 25: Execution total time (a) and average time per sample (b) (Notice that the axis representing the time is in a logarithmic scale).

As an example, for 520000 historic samples, while the CPU takes 7 minutes and 48 seconds to calculate the forecast, the GPU can achieve the same results in approximately one and half seconds.

The graph in figure 25-b displays the impact of each historic sample in the total computation time on both implementations. We can observe that while for the CPU simulation, the value remains almost unchanged, except for small variations, in the GPU case in addition to being significantly lower, it suffers a slightly decrease, since, initially the relatively low number of samples is not using the GPU capacity to its fullness. Thus, as we increase the number of historic samples, we are diluting the costs that can be considered fixed as the time needed to transfer data between host and device, which suffer a relatively small increase when compared to the impact on the time of the actual algorithm computations.

On the second test case, we vary the number of points to forecast, fixing the number of historic samples. The results are shown in figure 26-a and 26-b, where we can observe a similar scenario than the one presented for the first test case. On this test the GPU achieved an acceleration of more than 296 times in relation to the sequential simulation.

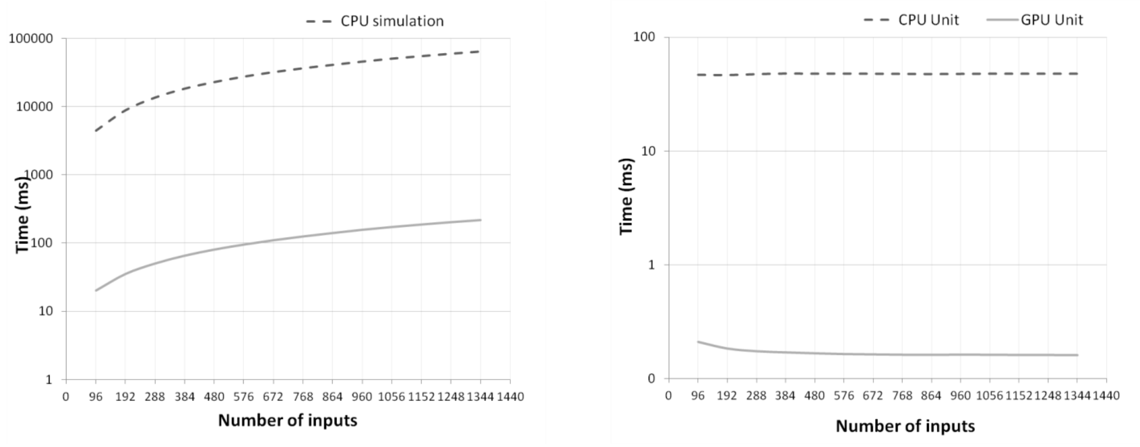

Figure 26: Execution total time (a) and average time per input (b) (Notice that the axis representing the time is in a logarithmic scale).

On the simulation for two weeks horizon, i.e. forecasting for 1344 points, while the CPU took approximately one minute to calculate, the parallelized implementation was able to perform the same calculation in less than 217 milliseconds.

We can also notice that the time spent calculating each point forecasted varies approximately as in the previous test case, while in the CPU the time remains the same, about 47 milliseconds per point, the GPU shows a slight decrease, due to the fact explained above, although in this case the decrease is considerably lower. This happens mainly because the quantity of data that has to be copied from host to device memory and backwards is very much dependent on the number of inputs to which we need to forecast and less in the number of sample observations, making this fixed cost more difficult to attenuate, which results in a more flat line.

When comparing this method to the ANN in terms of time performance, it is hard to obtain a fair comparison as the two processes are essentially different. While a KDE-based method involves the use of all the historic samples for each forecast, the ANN needs to be trained, which can be a time-consuming process, neverthless, after that training, the time required to forecast a large number of points is almost insignificant. It must be noticed that this train has to be performed for each wind farm individually, which is a major disadvantage when we need to modify the prediction model. Despite the ANN training process could be compared to the parameter optimization used in these tests, its importance to the actual forecasting process is much greater, since, when reasonable bandwidth values are known for the problem's explanatory variables, the improvement that can be achieved by the optimization for each historic set is not that significant. On the contrary, the neural network has to be trained for each set of historic data mandatorily.

Thus we can conclude that the use of an ANN method would be advantageous in cases where the model is fixed, i.e. there is not the need of changing some parameters on the model or introducing for example a new explanatory variable. Moreover, the differences in terms of time for the KDE-NW model as implemented in the GPU are not that relevant, when it comes to forecasting usual cases. For instance a prediction for a horizon of one week with a year of available historical data is performed in little more than 110 milliseconds, which is a relatively low overhead over the ANN that although forecasting in less than 3 milliseconds, needs approximately 9 minutes to complete the training process.

5.4 Probabilistic forecast results

In this section we present the obtained results of the probabilistic forecasting model. The model forecasts are presented in the form of point predictions with prediction intervals computed from two quantiles obtained from the beta fitted distribution. In order to assess the predictions accuracy, graphs of reliability and sharpness are presented.

Finally the graphs of time performance for both sequential and parallelized implementations are analyzed and the algorithms behaviors are discussed.

5.4.1 Accuracy

A probabilistic prediction example is presented for each wind farm in figures 27 and 28, where we can see the probabilistic prediction for the interval between the quantiles 10 and 90. This interval suggests that with a probability of 80% the power output will be inside the blue area of the graph. However, results show that for this interval the model overestimated on both test cases, presenting respectively for WF-A and WF_B, 90% and 88% of the measured values inside the interval.

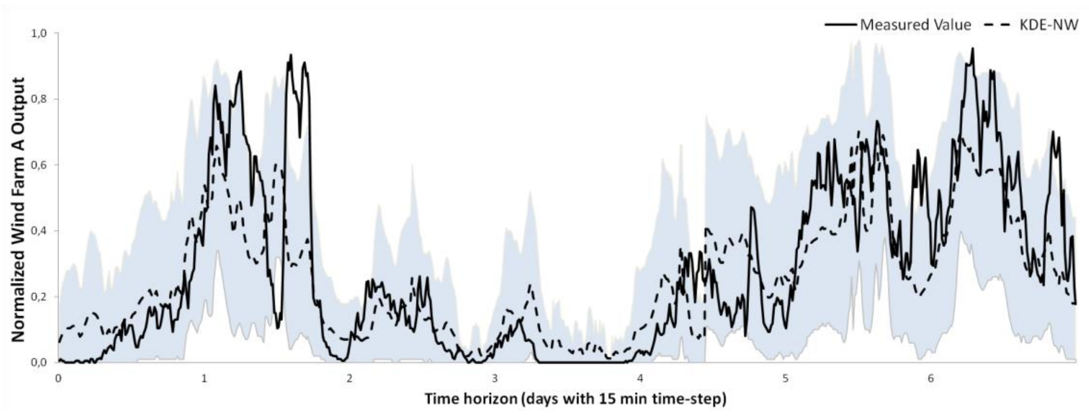

Figure 27: Probabilistic forecast for WF-A.

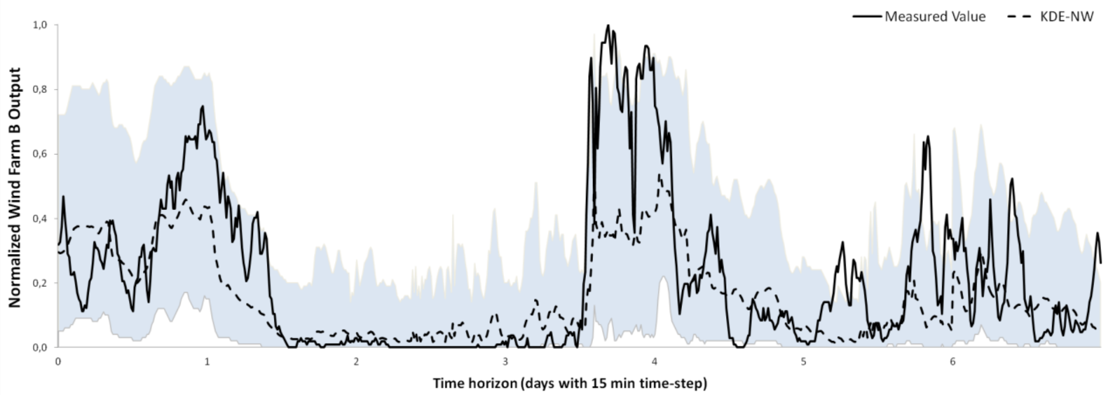

Figure 28: Probabilistic forecast for WF-B.

The reliability and sharpness graphs presented in figure 29 show that for WF-A the model presents a small underestimation on both left and right tails while overestimating the middle quantiles. In terms of sharpness, the interval size increases with the increase of nominal coverage as expected, ranging from 4% to 53%.

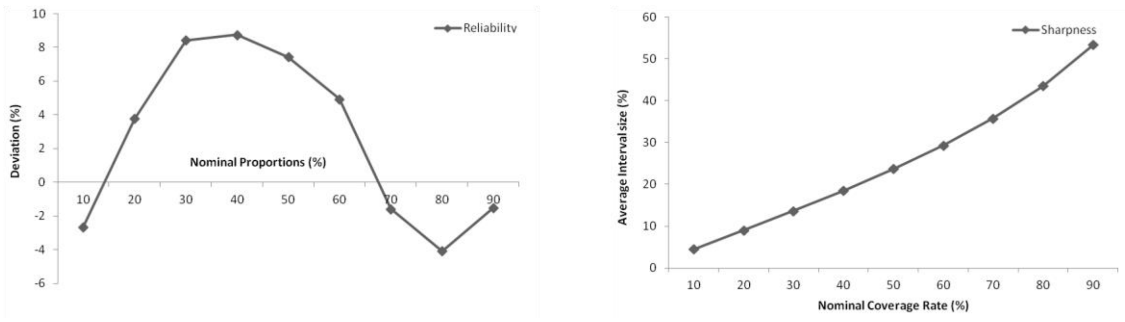

Figure 29: Reliability and sharpness for WF-A.

The graphs displayed in figure 30 are related to the WF-B forecast, which presents a slightly better reliability, obtaining the worst result for the 10% quantile. In terms of sharpness the results are virtually identical to the ones presented in the WF-A test, although presenting slightly better results.

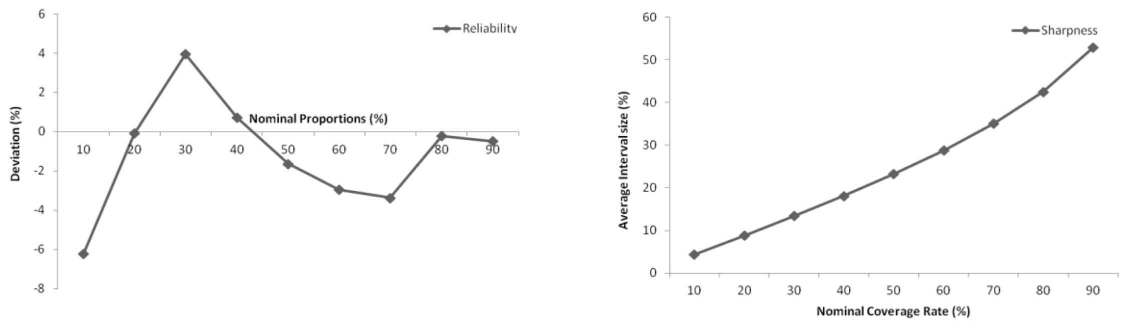

Figure 30: Reliability and sharpness for WF-B.

On both test cases reliability and sharpness presented normal results that are in accordance to what is described in literature, namely in [36] and [41].

5.4.2 Time

As previously mentioned, three different tests were performed in order to evaluate the behavior, with respect to time performance, of the proposed probabilistic forecasting algorithm implementation.

In both the first and second test case, for a varying a number of historic samples and inputs, respectively, the simulations performed very similarly to the ones presented for the punctual version of the algorithm.

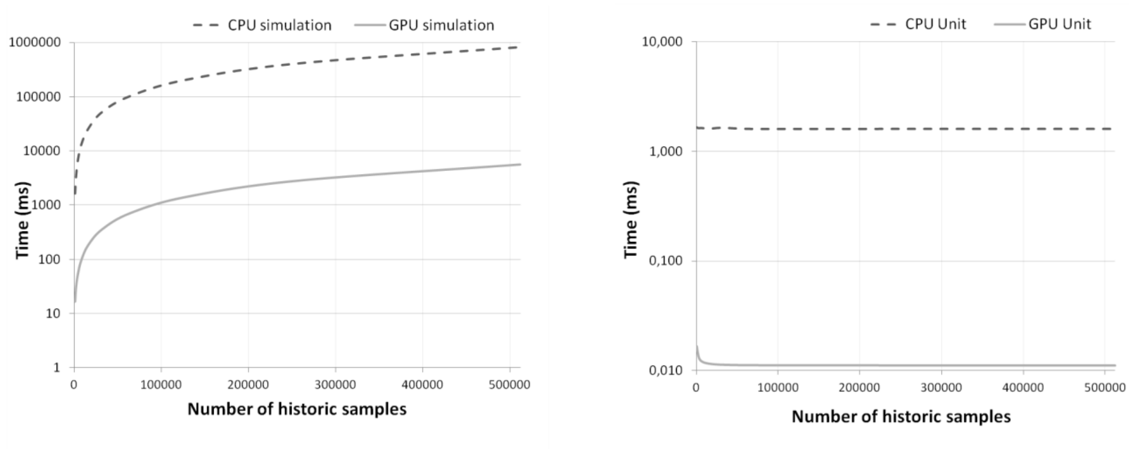

Figure 31: Execution total time (a) and average time per sample (b) (Notice that the axis representing the time is in a logarithmic scale).

However, it is possible to observe on the graphs displayed on figures 31-a and 32-a that the execution times suffered an overhead in relation to the ones presented for the point forecast. This overhead is related to the bins that need to be computed, which imply a lot more computations, including a beta fitting function that is executed on CPU on both algorithm versions. Due to this fact, it is understandable that the GPU improvement over the sequential algorithm is lower than the one obtained for the point forecast, however, still performed approximately 145 times faster on both tests.

As an example, for a prediction of one week horizon and 520000 historic samples, CPU takes 13 minutes and 42 seconds, while the GPU achieved approximately 5 and half seconds.

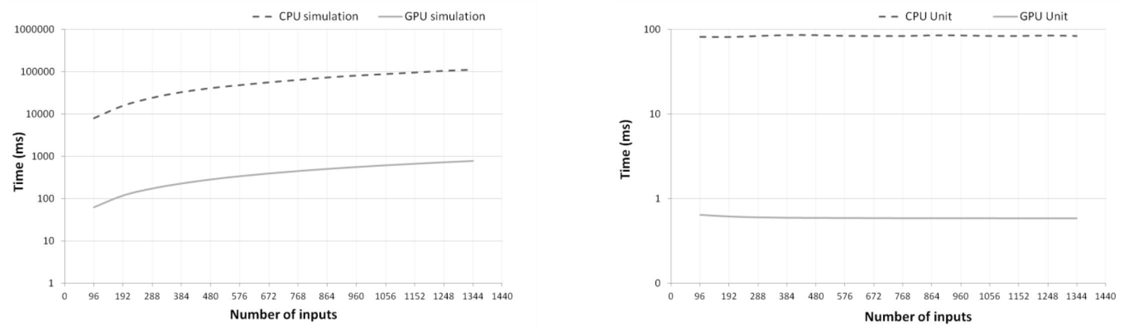

Figure 32: Execution total time (a) and average time per input (b) (Notice that the axis representing the time is in a logarithmic scale).

Figures 31-b and 32-b present, respectively, the results for the impact of each historic sample and input in the processing time. We can detect that, as in the point forecast version, the value of the CPU implementation keeps almost unchanged, while the GPU diminishes slightly. However, in this case the decrease is less accentuated, especially in the second test case, with an increasing number of inputs, where the difference is almost imperceptible. Indeed, these alterations on the results are also consequence of the additional bin calculations, which are greatly dependent on the number of inputs, as the beta fitting function is performed for each input, although the number of samples has also some impact.

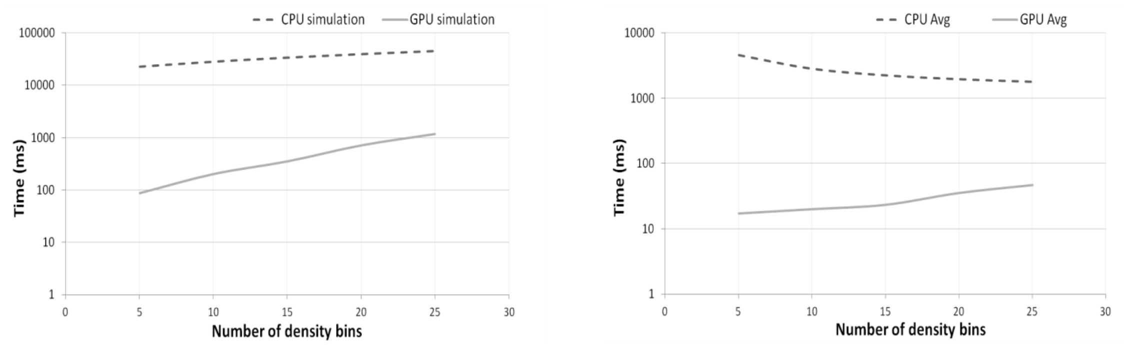

Figure 33: Execution total time (a) and average time per bin (b) (Notice that the axis representing the time is in a logarithmic scale).

On the third and last test case the algorithm is tested for an increasing number of bins and the execution time results are presented in figure 33-a. In this graph it is possible to observe a clear performance reduction from the GPU as the number of bins increase. However, it must be noticed (again) that the graph time axis is in a logarithmic scale and that the GPU implementation could achieve a performance improvement between 262 times on the 5 bins execution and 38 times on the 25. This improvement decrease happens due to the fact of bins having a great computation component performed on CPU, which is relative to the beta fitting function.

Consequences of this fact are also observable on figure 33-b, where the impact of each bin in the execution time is shown. In this graph we can see an opposing behavior to what occurs in the two previous tests, since, while in the GPU the impact increases, on the sequential algorithm it suffers a reduction. This reduction is due to the activation step which does not depend on the number of bins and has a relatively great impact on the execution time of the sequential algorithm, thus, initially, the importance of each bin is diluted. This does not happen in the GPU case, where the activation step is parallelized, representing a lower percentage of the total execution time, being the beta function that takes longer to process, which means that each bin is going to have a higher relevance in the algorithm performance.

When comparing the probabilistic to the deterministic implementations, we can observe that in terms of time performance the latter executes approximatelly 3.5 times faster, however, we lose the information about the forecast uncertainty. Regarding the accuracy, we cannot make a comparison between methods as they are evaluated differently.

Chapter 6

Conclusion

In this thesis we addressed two problems related to wind power forecasting. The first was increasing the amount of useful information about the forecast and therefore increase the value of the predictions as well as providing greater confidence in Smartwatt's solutions. The second was reducing the execution time, coping with the huge amount of data that needed to be processed, in order to allow that forecasts were delivered in real time.

To approach these problems first we proposed the implementation of a forecasting method based on KDE, which was used to compute single expected values and then modified to obtain probabilistic forecasts of wind power production. Moreover, the addition of two extensions to the base methodology was also discussed and evaluated. Then, in order to improve the performance of the proposed method, in terms of execution time, a parallelized version for GPU was developed using the CUDA programming model.

Based on experimental results we can draw the following general conclusions.

First, regarding the proposed extensions, we can conclude that the use of a specific approach for handling circular variables can have a considerable impact on the forecasting quality, although in the performed tests it did not show any significant impact. On the contrary, the use of weighted historical data was discarded, since, in addition to not showing any improvement on the quality of the results, it involves a great overhead as it is dependent on pre-processing computations.

Second, in terms of deterministic forecasting, the proposed method presented good prediction errors and when compared to a ANN it showed almost identical predictions results, not showing any great advantage over this method for this type of forecast. In terms of time performance, both the sequential and parallelized versions of the algorithm were evaluated. The GPU implementation could reach a 295 times improvement over the sequential version.

Finally, as regards to the probabilistic forecasts, both the reliability and sharpness of the forecasts were evaluated, presenting values in accordance to the literature. Moreover, the time performance tests for this case also presented high speedups, although with an increasing number of histogram bins the performance of the parallelized version decreased considerably.

Additional information on the uncertainty of the forecasts is increasingly important especially when performing risk assessments, thus this probabilistic forecasts can be of great importance for instance in electrical dispatch optimization, allowing for more reliable results. Furthermore, the decrease on the model execution time, will allow offering forecasts in real time.

References

1. R.Gelman, S.Gosset, “2011 Renewable Energy Data Book”, Energy efficiency and Renewable Energy, U.S. Department of Energy, 2011.

2. Council, Global Wind Energy, "Global wind report 2012," 2013.

3. REN21, “Renewables 2012, Global status report,” 2013.

4. P. Pinson, J. Juban and G. N. Kariniotakis, “On the quality and value of probabilistic forecasts of wind generation,” in Probabilistic Methods Applied to Power Systems, 2006, PMAPS 2006. International Conference on IEEE, June 2006.

5. H. K. Alfares and M. Nazeeruddin, “Electric load forecasting: literature survey and classification of methods,” International Journal of Systems Science, vol. 33, no. 1, pp. 23-34, 2002.

6. J. W. Taylor and P. E. McSharry, “Short-Term Load Forecasting Methods: An Evaluation Based on European Data,” IEEE Transactions on Power Systems, vol. 22, no. 4, pp. 2213-2219, 2007.

7. S. Makridakis and S.C. Wheelwright, “Forecasting Methods for Management,” John Wiley & Sons, 1989. Available: http://www.poms.ucl.ac.be/etudes/notes/prod2100/cours/Part%206-Forecast.pdf

8. J. S. Armstrong, F. Collopy, K. C. Green, “Answers to Frequently Asked Questions (FAQ) in Forecasting,” published on the Internet at forecastingprinciples.com and updated periodically. 24 November 2004.

9. K. C. Green, J. S. Armstrong and A. Graefe, “Methods to Elicit Forecasts from Groups: Delphi and Prediction Markets Compared,” 2007. Available: http://qbox.wharton.upenn.edu/documents/mktg/research/Delphi-WPv33.pdf

10. J. E. Beasley, “OR-notes”. Available: http://people.brunel.ac.uk/~mastjjb/jeb/or/forecast.html

11. R. J. Hyndman and G. Athanasopoulos, “Forecasting: principles and practice,” 2012. Available: http://otexts.com/fpp/

12. P. Pinson, “Estimation of the uncertainty in wind power forecasting,” Ph.D. dissertation, Ecole des Mines de Paris, 2006.

13. P. Pinson, H. A. Nielsen, J. K.Moller, H. Madsen, and G. Kariniotakis, “Nonparametric probabilistic forecasts of wind power: Required properties and evaluation,” Wind Energy, vol. 10, no. 6, pp. 497–516, November 2007.

14. C. Monteiro, R. J. Bessa, V. Miranda, A. Botterud, J. Wang, and G. Conzelmann, “Wind power forecasting: state-of-the-art 2009,” Report ANL/DIS-10-1, Argonne National Laboratory, Nov. 2009

15. E. A. Bossanyi, “Short-Term Wind Prediction Using Kalman Filters,” Wind Engineering, vol. 9, no. 1, pp. 1-8, 1985.

16. Hamed Babazadeh, Wenzhong Gao, Lin Cheng,Member and Jin Lin, “An Hour Ahead Wind Speed Prediction by Kalman Filter”, 2012.

17. G. W. Morrison and D. H. Pike, “Kalman Filtering Applied to Statistical Forecasting” Management Science, vol. 23, no. 7, March 2007.

18. M. Milligan, M.N. Schwartz, and Y. Wan, “Statistical Wind Power Forecasting for U.S. Wind Farms,” in Proceedings of the 17th Conference on Probability and Statistics in the Atmospheric Sciences/2004 American Meteorological Society Annual Meeting, Seattle, Washington, January 11–15, 2004.

19. J.L. Torres, A. García, M. de Blas, and A. de Francisco, “Forecast of hourly averages wind speed with ARMA models in Navarre,” Solar Energy, vol. 79, no. 1, pp. 65–77, 2005.

20. R.G. Kavasseri and K. Seetharaman, “Day-ahead wind speed forecasting using f-ARIMA models,” Renewable Energy, vol. 34, no. 5, pp. 1388–1393, May 2009.

21. P. Chen, T. Pedersen, B. Bak-Jensen, Z. Chen, “ARIMA-Based Time Series Model of Stochastic Wind Power Generation”, IEEE Transactions on Power Systems, vol. 25, no. 2, pp. 667-676, May 2010.

22. E. Cadenas, and W. Rivera, “Wind speed forecasting in the South Coast of Oaxaca, Mexico,” Renewable Energy, vol. 32, no. 12, pp. 2116-2128, October 2007.

23. P. Castro and R. Gomes, “Wind Speed and Wind Power Forecasting using Statistical Models: AutoRegressive Moving Average (ARMA) and Artificial Neural Networks (ANN),” International Journal of Sustainable Energy Development (IJSED), Volume 1, Issues 1/2, March/June 2012.

24. E. Cadenas and W. Rivera, “Short term wind speed forecasting in La Venta, Oaxaca, México, using artificial neural networks,” Renewable Energy, vol. 34, no. 1, pp. 274–278, January 2009.

25. E. Cadenas, O. A. Jaramillo and W. Rivera, “Analysis and forecasting of wind velocity in chetumal, quintana roo, using the single exponential smoothing method,” Renewable Energy, vol. 35, no. 5, pp. 925–930, May 2010.

26. M. Monfared, H. Rastegar and H. M. Kojabadi, “A new strategy for wind speed forecasting using artificial intelligent methods,” Renewable Energy, vol. 34, no. 3, pp. 845–848, 2009.

27. J. P. S. Catalao, H. M. I. Pousinho, and V. M. F. Mendes, “An artificial neural network approach for short-term wind power forecasting in Portugal,” in 15th International Conference on Intelligent System Applications to Power Systems, 2009, ISAP '09, pp.1-5, 8-12 November 2009.

28. G. Li, and J. Shi, "On comparing three artificial neural networks for wind speed forecasting," Applied Energy, vol. 87, no. 7, pp. 2313-2320, July 2010.

29. C.W. Potter and M. Negnevistky, "Very short-term wind forecasting for Tasmanian power generation," IEEE Transactions on Power Systems, vol. 21, no. 2, pp. 965–972, 2006.

30. L. Fugon, J. Juban and G. Kariniotakis, "Data mining for Wind Power Forecasting," in Proceedings of the European Wind Energy Conference EWEC'08, Brussels, Belgium, April 2008.

31. Z. Zheng, Y. Chen, X. Zhoua, M. Huoa, B. Zhaoc and M. Guod, "Short-Term Wind Power Forecasting Using Empirical Mode Decomposition and RBFNN," International Journal of Smart Grid and Clean Energy, vol. 2, no. 2, pp. 192–199, May 2013.

32. Ignacio J. Ramirez-Rosado, L.A. Fernandez-Jimenez, C. Monteiro, J.N. Sousa, and R. Bessa, "Comparison of two new short-term wind-power forecasting systems," Renewable Energy, accepted November 16, 2008.

33. Y. A. Katsigiannis, A. G. Tsikalakis, P. S. Georgilakis and N. D. Hatziargyriou, "Improved Wind Power Forecasting Using a Combined Neuro-fuzzy and Artificial Neural Network Model," Advances in Artificial Intelligence-Lecture Notes in Computer Science, 3955, pp. 105-115, 2006.

34. J. B. Bremnes, "Probabilistic wind power forecasts using local quantile Regression," Wind Energy, vol. 7, no. 1, pp. 47–54, March 2004.

35. J. B. Bremnes, "A comparison of a few statistical models for making quantile wind power forecasts," Wind Energy, vol. 9, no. 1-2, pp. 3-11, 2006.

36. J. Juban, L. Fugon, and G. Kariniotakis, "Probabilistic short-term wind power forecasting based on kernel density estimators," in Proceedings of the European Wind Energy Conference EWEC'07, Milan, Italy, 2007.

37. N. Meinshausen, "Quantile Regression Forests," Journal of Machine Learning Research, vol. 7, pp. 983–999, June 2006.

38. J. Juban, L. Fugon, and G. Kariniotakis, "Uncertainty Estimation of Wind Power Forecasts." in Proceedings of the European Wind Energy Conference EWEC'08, Brussels, Belgium, March 31–April 3, 2008.

39. M. Yang, S. Fang and W. J. Lee, "Probabilistic short-term wind power forecast using componential Sparse Bayesian Learning." in Industrial & Commercial Power Systems Technical Conference (I&CPS), May, 2012.

40. R. J. Bessa, J. Sumaili, V. Miranda, A. Botterud, J.Wang, and E. Constantinescu, "Time-adaptive kernel density forecast: A new method for wind power uncertainty modeling," in Proc. 17th PSCC Conf., Stockholm, Sweden, Aug. 2011.

41. R. J. Bessa, V. Miranda, A. Botterud, J.Wang, and E. Constantinescu, "Time adaptive conditional kernel density estimation for wind power forecasting." 2012.

42. NVIDIA Inc, “What is GPU computing?” Available: http://www.nvidia.com/object/what-is-gpu-computing.html

43. C. McClanahan, “History and Evolution of GPU Architecture,” 2010. Available: http://mcclanahoochie.com/blog/wp-content/uploads/2011/03/gpu-hist-paper.pdf

44. NVIDIA Inc,” CUDATM Programming Guide 3.1.” Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, Jun 2013.

45. J. Ghorpade, J. Parande, M. Kulkarni and A. Bawaskar, “Gpgpu processing in CUDA architecture,” Advanced Computing: An International Journal (ACIJ), vol. 3, no. 1, January 2012

46. S. Tariq and NVIDIA Inc., “An Introduction to GPU Computing and CUDA Architecture,” 2011. Available: http://developer.download.nvidia.com/CUDA/training/GTC_Express_Sarah_Tariq_June2011.pdf

47. V.K. Pallipuram, M. Bhuiyan, M. C. Smith, “A comparative study of GPU programming models and architectures using neural networks” Journal of Supercomputing, Vol. 61, no. 3, pp. 673-718, September 2012.

48. J. Fang, A. L. Varbanescu and H. Sips, “A Comprehensive Performance Comparison of CUDA and OpenCL,” Proceedings of the International Conference on Parallel Processing, art. no. 6047190, pp. 216-225, 2011.

49. K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, “Evaluating Performance and Portability of openCL Programs,” in The Fifth International Workshop on Automatic Performance Tuning, 2010.

50. K. Karimi, N. G. Dickson and F. Hamze, “A Performance Comparison of CUDA and OpenCL,” arXiv:1005.2581 [cs.PF], 14 May 2010.

51. NVIDIA Inc, “Kepler GK110,” Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

52. M.-Y. Huang, S.-C. Wei, B. Huang, Y.-L. Chang, "Accelerating the Kalman filter on a GPU," in Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, 6121397, pp. 1016-1020, 2011.

53. P. Trebatický, J. Pospíchal, “Neural network training with extended Kalman filter using graphics processing unit,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5164 (2), pp. 198-207, 2008.

54. C. E. Davis, “Graphics processing unit computation of neural networks,” Doctoral dissertation, The University of New Mexico, 2005.

55. X. Sierra-Canto, F. Madera-Ramírez and V. Uc-Cetina, ”Parallel training of a back-propagatio neural network using CUDA,” Proceedings - 9th International Conference on Machine Learning and Applications, 5708849, pp. 307-312, 2010.

56. S. Scanzio, S. Cumani, R. Gemell, F. Mana and P. Laface, "Parallel implementation of artificial neural network training," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, 5495108, pp. 4902-4905, 2010.

57. Z. Luo, H. Liu,, and X. Wu, "Artificial neural network computation on graphic process unit," Proceedings of International Joint Conference on Neural Networks, Montreal, Canada, vol. 1, pp. 622-626, 2005.

58. H. Takizawa, T. Chida and H. Kobayashi, "Evaluating Computational Performance of Backpropagation Learning on Graphics Hardware," Electronic Notes in Theoretical Computer Science, Volume 225, 2 January 2009, Pages 379-389, January 2009.

59. K.-S. Oh and K. Jung, "GPU implementation of neural networks," Pattern Recognition, vol. 37, no. 6, pp. 1311-1314, 2004.

60. T. Sharp, "Implementing decision trees and forests on a GPU," Computer Vision–ECCV 2008, 595-608, 2008.

61. D. Slat and M. H. Lapajne, "Random Forests for CUDA GPUs," 2010.

62. T.-N. Do and V.-H. Nguyen, "A novel speed-up SVM algorithm for massive classification tasks," IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies, 4586358, pp. 215-220, 2008.

63. T.-N. Do, V.-H. Nguyen and F. Poulet, "Speed up SVM algorithm for massive classification tasks," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5139 LNAI, pp. 147-157, 2008.

64. B. Catanzaro, N. Sundaram and K. Keutzer, "Fast support vector machine training and classification on graphics processors," Proceedings of the 25th international conference on Machine learning, pp. 104-111, July 2008.´

65. S. Herrero-Lopez, J. R. Williams and A. Sanchez, "Parallel multiclass classification using SVMs on GPUs," Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp. 2-11, March 2010.

66. F. Giannesini and B. Le Saux, "Gpu-accelerated one-class svm for exploration of remote sensing data".

67. P. D. Michailidis and K. G. Margaritis, "Accelerating Kernel Density Estimation on the GPU Using the CUDA Framework," Applied Mathematical Sciences, vol. 7, no. 30, pp. 1447-1476, 2013.

68. J. Juban, N. Siebert, and G. Kariniotakis, "Probabilistic short-term wind power forecasting for the optimal management of wind generation," in Proc. IEEE PowerTech Conf., Lausanne, Switzerland, July 2007.

69. S. Kuehn, "Kernel Regression by Mode Calculation of the Conditional Probability Distribution," arXiv preprint arXiv:0811.3499, 2008.

70. H. Bludszuweit, J.A. Dominguez-Navarro and A. Llombart, “Statistical Analysis of Wind Power Forecast Error,” IEEE Transactions on Power Systems, vol. 23, no. 3, pp. 983–991, August 2008.

71. O. Kramer, B. Satzger and J. Lässig, “Power prediction in smart grids with evolutionary local kernel regression,” in Hybrid Artificial Intelligence Systems, pp. 262-269, 2010.

72. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell, “A Survey of general-purpose computation on graphics hardware,” in Computer graphics forum, vol. 26, no. 1, pp. 80-113, March 2007.

Annex A

Reference tables

This appendix presents two tables for quick reference. Table 2 shows the parameters used to describe the model implementation in chapter 3, while table 3 presents the variables used in the model implementation in chapter 4.

Table 2: Prediction model's parameters reference table.

Parameter	*Description*
t	Time
k	Time-step
x	Explanatory variables given at time t
y	Variable to predict
D_t	Historic set at time t
X	Historic sample
N	Number of historic samples
h	Variable bandwidth
d	Number of explanatory variables
B	Number of bins
K	Kernel function

Table 3: Algorithms' variables reference table.

Parameter	*Description*
N	Number of historic samples
I	Number of inputs
d	Number of explanatory variables
B	Number of bins

Parameter	*Description*
K	Kernel function
G	Number of blocks per grid line
M	Number of threads per block