Applied Energy 201 (2017) 412–418

Contents lists available at ScienceDirect

Applied Energy

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

A GPU deep learning metaheuristic based model for time series forecasting

Igor M. Coelho [a,b,*], Vitor N. Coelho [a,c,*], Eduardo J. da S. Luz [d], Luiz S. Ochi [c], Frederico G. Guimarães [e], Eyder Rios [f]

[a] Grupo da Causa Humana, Ouro Preto, Brazil
[b] Department of Computing, State University of Rio de Janeiro, Rio de Janeiro, Brazil
[c] Institute of Computing, Universidade Federal Fluminense, Niterói, Brazil
[d] Department of Computing, Universidade Federal de Ouro Preto, Ouro Preto, Brazil
[e] Department of Electrical Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[f] Institute of Computing, UESPI, Paranaíba, Brazil

HIGHLIGHTS

- A CPU-GPU mechanism is proposed in order to accelerate time series learning.
- Disaggregated household energy demand forecasting is used as case of study.
- Suggestions to embed the proposed low energy GPU based system into smart sensors.
- Parallel forecasting model accuracy evaluation with a metaheuristic training phase.

ARTICLE INFO

Article history:
Received 30 September 2016
Received in revised form 2 January 2017
Accepted 3 January 2017
Available online 9 January 2017

Keywords:
Deep learning
Graphics processing unit
Hybrid forecasting model
Smart sensors
Household electricity demand
Big data time-series

ABSTRACT

As the new generation of smart sensors is evolving towards high sampling acquisitions systems, the amount of information to be handled by learning algorithms has been increasing. The Graphics Processing Unit (GPU) architecture provides a greener alternative with low energy consumption for mining big data, bringing the power of thousands of processing cores into a single chip, thus opening a wide range of possible applications. In this paper (a substantial extension of the short version presented at REM2016 on April 19–21, Maldives [1]), we design a novel parallel strategy for time series learning, in which different parts of the time series are evaluated by different threads. The proposed strategy is inserted inside the core a hybrid metaheuristic model, applied for learning patterns from an important mini/microgrid forecasting problem, the household electricity demand forecasting. The future smart cities will surely rely on distributed energy generation, in which citizens should be aware about how to manage and control their own resources. In this sense, energy disaggregation research will be part of several typical and useful microgrid applications. Computational results show that the proposed GPU learning strategy is scalable as the number of training rounds increases, emerging as a promising deep learning tool to be embedded into smart sensors.

1. Introduction

Sometimes called as the hugest machine ever built, the power grid has been undergoing several improvements. Researchers and the industry have been focusing on efficiently integrating Renewable Energy Resources (RER) into the grid. The massive insertion of RER is usually assisted by Artificial Intelligence (AI) based algorithms and models [2], which are being embedded into Smart Meters (SM) [3]. The proposal described in this current study is a potential tool to be embedded into SM, being able to forecast useful information from big data disaggregated load time series. These load time series have the potential of assisting RER integration in mini/microgrid systems, in which users might employ smart devices to self-manage their resources and demands.

SM are "smart" in the sense that the modest use of sensors is being replaced by devices with plenty of computational abilities.

* Corresponding authors at: Grupo da Causa Humana, Ouro Preto, Brazil.
E-mail addresses: igor.machado@ime.uerj.br (I.M. Coelho), vncoelho@gmail.com (V.N. Coelho).

http://dx.doi.org/10.1016/j.apenergy.2017.01.003
0306-2619/

Usually, these computational abilities are developed based on AI techniques or specific strategies envisioned by its creator/programmer. This class of meters are starting to communicate to each other [4] and to introduce important information to be dealt with by decision makers. These software based sensors are crucial for the decision making process over these scenarios filled with uncertainties.

Among AI techniques found in the literature, deep learning based ones are in evidence. Deep learning has been applied to several classification and regression problems. Part of its success is due to automatic feature extraction at different levels of abstraction. Automatic feature extraction promotes the easy re-utilization of models on different domains without a field-specialist human intervention. Moreover, deep learning allows the representation of the nonlinearities, often associated with complex real-world data. Deep learning models have been used to achieve state-of-the art results in the field of computer vision [5,6] and have also been applied to the problem of time series forecasting [7–10].

Popular deep learning approaches are based on convolutional networks [5], restricted boltzman machines (deep belief networks) [11] and deep autoencoders [12]. However, these methods are often difficult to interpret and reproduce. According to Hu et al. [13] several authors treat deep network architectures as a black box. Another limitation of popular deep learning methods is the high memory consumption [14]. In contrast, the method proposed in this work is of easy interpretation and has low memory consumption, which means a competitive advantage over popular methods of deep learning.

Coelho et al. [15] recently introduced a Hybrid Forecasting Model (HFM), which calibrates its fuzzy rules using metaheuristic based procedures. Without applying any filter or pre-processing energy consumption time series, the HFM model showed to be competitive with other specialized approaches from the literature and easily generalized for performing n-steps-ahead forecast.

The extension proposed here (a substantial extension of the short version presented at REM2016 on April 19–21, Maldives [1]) explores the learning capabilities of the HFM tool, in which feature extraction is done by Neighborhood Structures (NS). Fig. 1 details a generalized version of how the proposed model works. In this current work, only layer 2 is considered, in which the special operator returns the average values of all active functions, namely "activations". NS are used for calibrating each parameter of each activation function: lag input for the backshift operator, rule position and application weight. Furthermore, the metaheuristic calibration algorithm is able to regulate the size of the layer, adding or removing functions.

Motivated by the new class of big data time series, which are reality in several areas (such as in the energy industry, biology, neuroscience, image processing, among others), we decide to enhance the HFM model with a new parallel forecasting evaluation strategy. In particular, in this current work, Graphics Processing Unit (GPU) were designed to be used for forecasting different parts of a microgrid load time series. The use of GPU based architectures can provide a greener alternative with low energy consumption for mining information from such huge datasets [16]. Each GPU provides thousands of processing cores with much faster arithmetic operations than a classic Central Processing Unit (CPU). In a nutshell, we aim at generating ensemble GPU threads learning process, which provide independent forecasts, optimized in order to reduce a given statistical quality measure. GPU seems to fit the scope of the HFM, since the model can be implemented and adapted to GPU computing, particularly because the method uses metaheuristics algorithms and was implemented in the core of the OptFrame [17]. The automatic parameters calibration process of the HFM also matches big data time-series requirements, mainly due to its metaheuristic based learning phase. For this purpose, NS plays a vital role in calibrating the model and finding more efficient solutions. Associated with the power and flexibility of the metaheuristics, the absence of parameters tuning simplify the application of the proposed framework to different times series, in particular, when n-steps-ahead forecasting is required.

This paper considers a mini/microgrid forecasting problem as case of study, the Disaggregated Household Electricity Demand Forecasting. Researchers had begun to publicly release their data sets, such as the Reference Energy Disaggregation Dataset (REDD) [18], which provides low-frequency power measurements (3–4 s intervals) available for 10–25 individually monitored circuits. The household electricity demand forecasting has great potential for microgrid applications, such as the design of green buildings and houses [19]. Forecasting different disaggregated time series from a house opens a wide range of possibilities for efficient RER integration. Considering that billions of dollars are being spent to install SM [20], researchers are advocating that appliance level data can promote numerous benefits.

In the remaining of this paper we introduce the GPU architecture in detail (Section 2) and the GPU disaggregated forecasting process (Section 3). The computational results and the analyzed parameters are presented in Section 4 and, finally, Section 5 draws some final considerations.

2. GPU architecture

The GPU was originally designed for graphic applications (thus receiving the name of a Graphics Processing Unit) such that any non-graphic algorithm designed for GPU had to be written in terms of graphics APIs such as OpenGL. This allowed the development of scalable applications for computationally expensive problems, such as collision simulations in physics [21]. The GPU programming model evolved towards the modern General Purpose GPU (GPGPU), with more user-friendly and mature tools for application development such as CUDA, a proprietary C++ language extension from NVIDIA, one of the main GPU manufacturers [16].

The GPU architecture is organized as an array of highly threaded streaming multiprocessors, each one containing a number of processing units (cores), besides a multi-level memory structure. The configuration of GPU hardware depends on the *compute capability*, a parameter related to GPU micro-architecture that defines which hardware features will be available for CUDA development. A CUDA program consists of one or more phases that are executed in CPU or GPU. The GPU code is implemented as C++ functions known as *kernels*, that are launched by CPU code in a *compute grid*, usually formed by a large number of threads that execute the same kernel aiming at exploiting data parallelism. The threads in a grid are organized in a two-level hierarchy, where first level is arranged as a three-dimensional array of blocks, each one containing up to 1,024 threads. In second level, each block is also arranged in a three-dimensional way. The dimensions of a grid are designed by the programmer and should observe the limits determined by the compute capability of the hardware.

After a kernel is launched, each block is assigned to a single streaming multiprocessors, which executes the threads of a block in groups called *warps*. Each warp is processed according to SIMD (Single Instruction, Multiple Data) model, meaning all threads in a warp execute same instruction at any time. Global memory accesses or arithmetic operations performed by some thread also affects warp execution, forcing all threads in the same warp to wait until the operation is completed. To hide the latency related to those operations, GPU schedules another warp to keep SM busy. This is possible because GPU is able to handle more threads per SM than cores available.

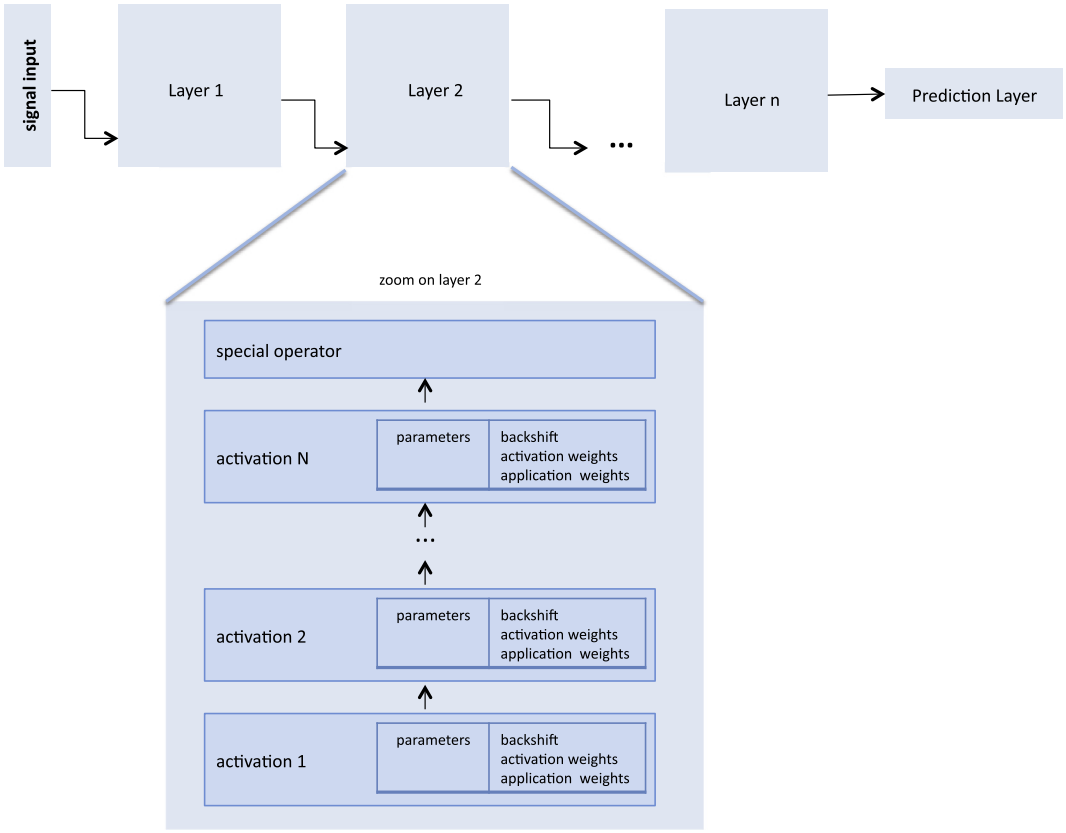


Fig. 1. HFM deep learning model.

The communication between the CPU and the GPU is done through the GPU global memory. However, the data transfer from/to this resource is usually a bottleneck to application performance. Therefore, the amount of data copies between CPU and GPU have to be minimized for efficient implementation, and the other levels of the GPU memory hierarchy must be explored. The memory hierarchy is one of the most distinguished features of the current GPUs. It includes the global memory, a local register memory for each thread and a shared memory for each block of threads. This last type of memory is usually of main importance in order to achieve high performance in GPU programming, by assigning similar tasks to threads in the same GPU block. Due to the low energy consumption of the GPUs and high processing power, the integration between CPUs and GPUs can be explored in order to improve forecasting algorithms.

3. HFM with GPU disaggregated forecasting process

Let us consider a target time series $ts = y_1, \ldots, y_t$, comprising a set of t observations. The goal is to estimate/predict a finite sequence $\{\hat{y}_{t+1}, \ldots, \hat{y}_{t+k}\}$, with k indicating the k-steps-ahead to be predicted, namely the *forecasting horizon*.

The idea explored in our approach relies on the independence of the training rounds, which are then performed by different GPU threads. Different training rounds can be seen as independent windows, which are known to be an interesting application for designing high-performance hardware with parallel computation [22]. In fact, this application is specially fit to the GPU computing SIMD paradigm [23]. The threads in a GPU warp execute the same instruction simultaneously, but over different parts of the training set. With this disaggregated forecasting process, it is possible to drastically reduce the computing effort by sharing it with thousands of independent processing units in a single GPU.

Fig. 2 depicts an example of a time series divided during the model performance evaluation process, performed by n different threads. Due to the nature of the load time series dealt with in this current work, it only depends on past data (in green[1]) for predicting the next values. In this sense, each thread can handle a forecasting round over a part of the complete time series (an analogy to the bag of tasks problems), generating, as output (in blue), the predicted values for that specific part, stored in the GPU global memory. Formally, in a single step sliding window strategy, each thread runs an independent forecasting process, returning its predicted values for a given k-steps-ahead forecasting horizon (a finite sequence of $\{\hat{y}_{t_{thread_i}+1}, \ldots, \hat{y}_{t_{thread_i}+k}\}$ predicted values, with $t_{thread_i} \in [1, \ldots, t-k]$). In the sliding window strategy [24], the two boxes together should move along the whole time series. Finally, the CPU concatenates the set of predicted values returned by each thread. Following this strategy, the CPU should only dedicate its efforts in performing the accuracy evaluation of the returned forecasts. For the family of time series tackled here, the disaggregated household energy consumption, the HFM will now have a parallel strategy for evaluating its accuracy, which will check the quality of the neighbor solutions faster.

In particular, the HFM requires a minimum number of ts_{maxLag} samples for feeding its model. This parameter guides the range in which the NS can calibrate the backshift operators. Parameter ts_{maxLag} limits the oldest lag to be used by the forecasting model (represented by the length of the green boxes).

[1] For interpretation of color in Fig. 2, the reader is referred to the web version of this article.

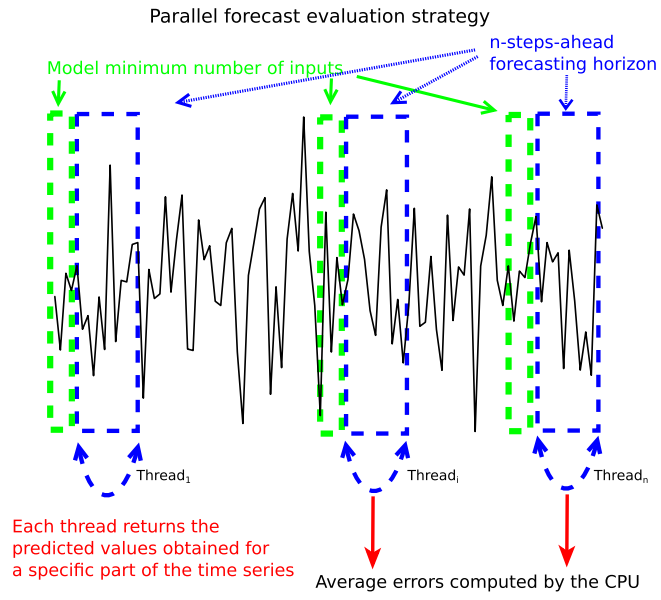


Fig. 2. HFM with a sliding window parallel GPU forecasting process.

4. Computational experiments

The REDD considered in this study provides low-frequency data for household electricity demand forecasting. Statistical analysis of well-known forecasting models over this data set was done by Veit et al. [25]. Following their description, we extracted the data of house 1 from Apr 18th 2011 22:00:00 GMT to May 2nd 2011 21:59:00 GMT.

Four different time series are analyzed here: three different individually monitored circuits; and one of the two-phase main input. Since the time series have several gaps, holes/breaks due to meters or sensors not providing measurements, they were interpolated. Each individual house component passed through linear interpolations in order to have a granularity of 1 s for each analyzed time series. The tests were carried out on a computer Intel i7-4790 K 4.00 GHz, 16 GB RAM, equipped with a NVIDIA GPU GTX TITAN X, 12 GB of global memory, 3072 CUDA cores, compliant with Maxwell microarchitecture.

4.1. Speedup analysis

Different parts of the analyzed time series were analyzed, being composed of a total number of samples hereafter called *nSamples*: 5,000; 10,000; 50,000; 100,000; 200,000; 500,000 and 1,000,000. The ability of the proposal in handling different forecasting horizons was also analyzed for the following values of steps-ahead k (s): 1, 2, 5, 10, 60, 360, 720, 1800, 3600 and 7200. Thus, the maximum forecasting horizon analyzed was 2 h, the equivalent of 7200 s.

Following this design, the HFM model was fed with $nSamples + k$ samples, allowing it to perform at least one training round for each configuration. For each configuration, 1000 function evaluations were performed during the metaheuristic based training phase of the HFM. Fig. 3 depicts a superimposed interaction plots, generated using the *ggplot2* R package. Time was measured with precision of millisecond (ms).

In order to compare the pure CPU and the parallel CPU-GPU strategy, we rely on the concept of speedup, where the time spent by the CPU sequential algorithm is divided by the time spend by the parallel (to perform the same task). Analyzing Fig. 3, one can conclude that as the number of samples increases, the GPU outperforms CPU performance with an average speedup of nearly 15 times. On the other hand, we could also conclude that the disaggregated forecasting process was not suitable for small size time series, when the GPU gain does not compensate the initial overhead and the lack of work for the GPU threads.

We also study the speedups in a different way, by relating the speedup growth with the Number of Training Rounds, namely *NTR*, related to each configuration. Basically, the NTR, also known as Number of Training Cycles, following a k step-ahead sliding window strategy, is equal to the number of samples divided by the forecasting horizon, as described in Eq. (1). The *NTR* is equal to the number of tasks launched by the GPU.

$$ts_{NTR} = \frac{ts_{nSamples} - ts_{maxLag}}{ts_k} \quad (1)$$

Thus, we calculated the *NTR* for each configuration, rounded it and cut into the following intervals: 0; 200; 500; 1,000; 10,000; 50,000; 100,000; 500,000 and 1,000,000. As can be noticed, the last two higher intervals were only executed for forecasting horizons with steps-ahead equal to 1, 2, 5, 10 or 60.

Fig. 4 shows the speedup in relation to the *NTR*, considering different forecasting horizons. By considering the gains on different training rounds it is possible to verify an increasing speedup. Even for long term forecasting, the amount of global memory accessed in GPU did not considerably reduce the speedup. In fact, the GPU seems to be able to produce greater speedups from 10 to 60 steps ahead. When the method performs longer term predictions the high amount of work done by each GPU thread implied in lower speedups.

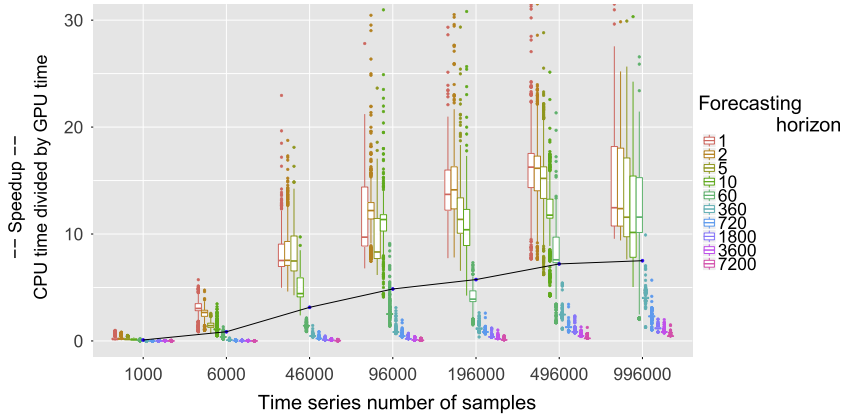


Fig. 3. Parallel forecasting model speedup according to the number of samples.

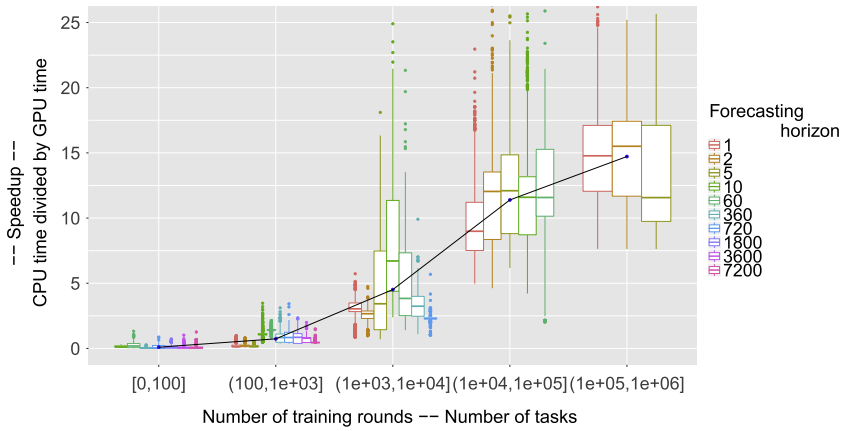


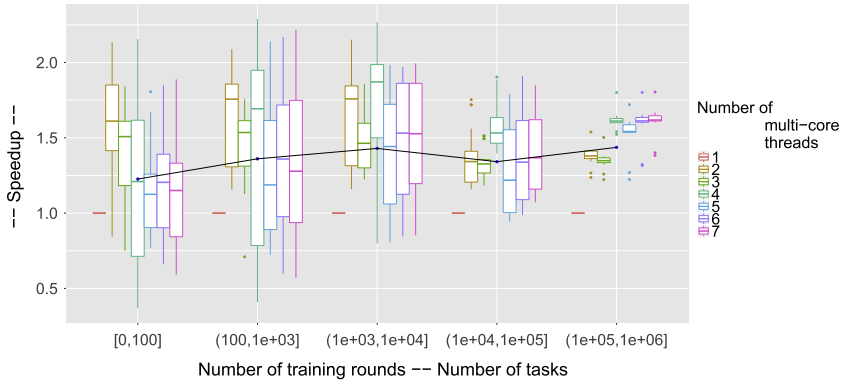
Fig. 4. Speedup according to the number of tasks.

In addition, we verify the performance of the proposed parallel evaluation strategy in a multi-CPU architecture. Experiments were run using an Intel Core i7-7500U, 2.70 GHz, with the OpenMP of g++ 5.4. The batch of experiments was designed in order to verify the speedup with 1 thread to 7 CPU threads. Analogously to the previous experiments, we analyzed the same set of values for k and *nSamples*. The best model performance was verified when 4 threads were running in parallel. In this case, the average speedup was around 180%, quite low compared to the speedup of the proposed GPU strategy. However, we believe that investing in this kind multi-CPU architecture is also promising, in particular, due to its more uniform speedup, as can be verified in both plots depicted in Fig. 5.

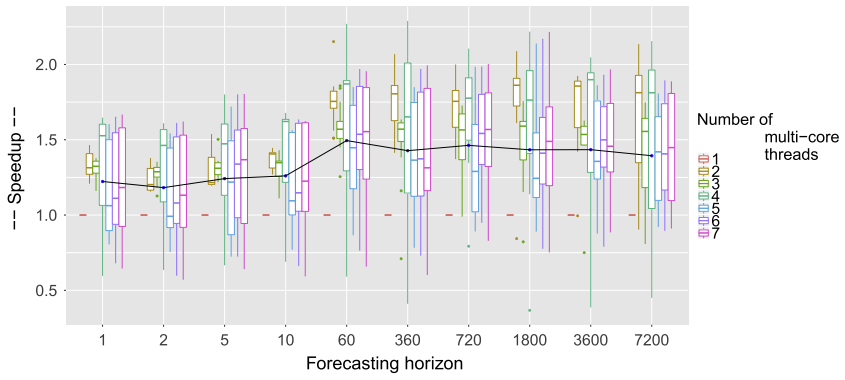
4.2. REDD dataset results with high granularities

In this section, we report some initial results with 1 s training phases of the proposed parallel HFM model, following a similar batch of experiments reported by Veit et al. [25]. As done by them, we verified the performance of our model regarding 18 different configurations, with granularities of 15, 30 and 60 min, covering seven different forecasting horizons of 1440, 720, 360, 180, 60, 30 and 15 min.

The sliding window strategy was used for iteratively training and testing the model. Thus, the data set was split into windows with defined lengths (3 days + 3 days as possible inputs to be used by the model, parameter ts_{maxLag}). Instead of setting the sliding



(a) Speedup according to the number of tasks



(b) Speedup for different forecasting horizons

Fig. 5. Speedups according to the number of CPU threads.

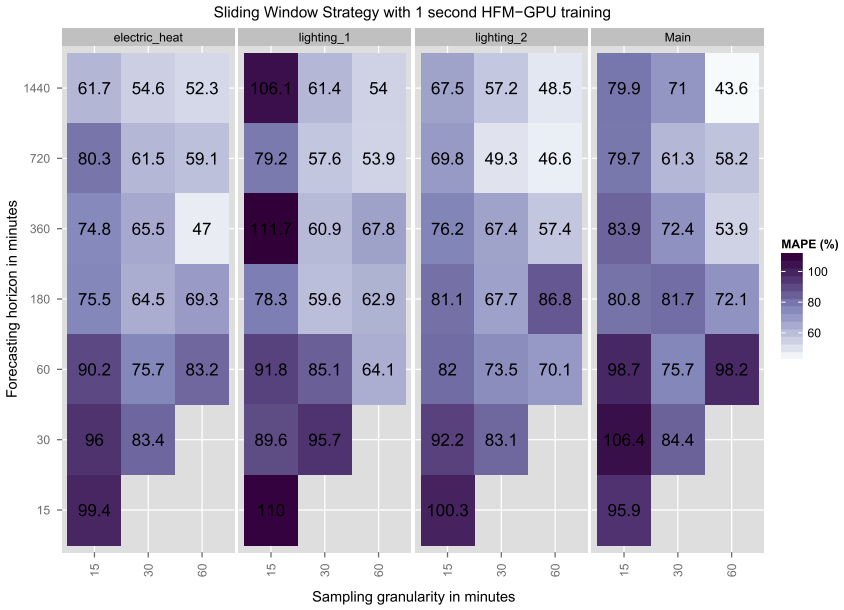


Fig. 6. MAPE for n-steps-ahead forecasting and different granularities with a sliding window strategy.

length to 24 h, we explore the whole time series, moving the sliding windows with a single unit (15, 30 or 60 min).

Fig. 6 shows heatmaps of the Mean Absolute Percentage Error (MAPE). These initial results suggest the competitiveness of our proposal with the results reported in the literature. In particular, we highlight our solid performance in forecasting horizons higher than 60, mainly due to the metaheuristic self-calibration strategy used by our proposed framework. Since the proposed parallel forecasting evaluation strategy does not interfere with the quality of the prediction model, the convergence of the proposed parallel HFM model follows the previously validated pure HFM model.

Embedding the proposed framework inside SM and daily devices [26], using personal supercomputers with accelerators such as FPGA [27], GPUs, ARM, Intel MIC architecture, would allow more precise real-time decision making. Furthermore, for energy applications, such as embedding the proposed tool into SM, GPUs have been advocated as the green alternative, with low cost per computation. More specifically, future studies should design novel strategies for limiting, or self-calibrating, the maximum oldest lag to be used by the HFM model. When the model requests inputs of recent lags, values that had just been predicted should be feedback into the model, what generates a highly dependent computation (over previous values) that may not be easily explored by parallel paradigms. Exploring the advantages and disadvantages of those inputs on a parallel n-step-ahead time series forecasting sounds a worth topic to be studied.

Finally, we suggest the application of the proposed framework for tackling other important big data time series of the energy sector, such as solar radiation, rainfall and wind speed forecasting, or even other types of time series, such as for Electroencephalography and Magnetoencephalography learning and classification.

5. Conclusions

In this paper, a multithread based strategy was designed for checking the performance of a metaheuristic based n-steps-ahead time series forecasting model. The main core of our strategy was to split the time series into independent training parts. As case of study, data from an important microgrid energy forecasting problem was used.

The proposed GPU deep learning strategy appears to be scalable as the number of time series training rounds increases, achieving up to 45 times speedup over a single threaded CPU implementation and, on average, around 15 times. Being extensible for n-steps-ahead forecasting, we verified that increasing the forecasting horizon, at a certain level, may decrease the total speedup performance, since more work is effectively assigned to each of the thousands of processing cores in a GPU.

Exploring the fact that machine learning techniques can be used to break down household energy consumption data into individual appliances, we explored our proposal performing forecasting in a public energy disaggregation dataset. In this sense, we used this kind of time series forecasting problem as example expecting that using efficient AI algorithms, in conjunction with smart meters, is a cost-effective and scalable solution for the rise of smart homes. The obtained results suggested that the proposal could be applied in the new generation of mini/microgrid software based sensors, since it showed a reasonable performance for predicting energy consumption of different household components.

Acknowledgment

Vitor N. Coelho would like to thank the Brazilian funding agencies CAPES and the PPGEE/UFMG, for sponsoring the trip to REM 2016, and FAPERJ (inside the scope of the project “Autonomia em redes inteligentes assistida por ferramentas da inteligência computacional”), for supporting and encouraging the development of this study. Frederico G. Guimarães was supported by the Brazilian agencies CNPq (312276/2013-3 and 306850/2016-8) and FAPEMIG. Luiz S. Ochi was supported by FAPERJ and CNPq (301593/2013-2) and Igor M. Coelho by FAPERJ.

The authors would like to thank the anonymous reviewers for their valuable comments and feedback, as well as the REM 2016 session chairs, for motivating us to extended the presented work.

References

[1] A hybrid deep learning forecasting model using gpu disaggregated function evaluations applied for household electricity demand forecasting, Energy Procedia 2016;103C:280–285. http://dx.doi.org/10.1016/j.egypro.2016.11.286.

[2] Kow KW, Wong YW, Rajkumar RK, Rajkumar RK. A review on performance of artificial intelligence and conventional method in mitigating PV grid-tied

related power quality events. Renew Sust Energy Rev 2016;56:334–46. http://dx.doi.org/10.1016/j.rser.2015.11.064.
[3] McHenry MP. Technical and governance considerations for advanced metering infrastructure/smart meters: technology, security, uncertainty, costs, benefits, and risks. Energy Policy 2013;59:834–42. http://dx.doi.org/10.1016/j.enpol.2013.04.048.
[4] Bertoldo R, Poumadère M, Rodrigues Jr LC. When meters start to talk: the public's encounter with smart meters in France. Energy Res Soc Sci 2015;9:146–56. http://dx.doi.org/10.1016/j.erss.2015.08.014 [special Issue on Smart Grids and the Social Sciences].
[5] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521(7553):436–44.
[6] Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R, LeCun Y. Overfeat: integrated recognition, localization and detection using convolutional networks. In: International Conference on Learning Representations (ICLR 2014); 2014. arXiv preprint arXiv:1312.6229.
[7] Mirowski PW, LeCun Y, Madhavan D, Kuzniecky R. Comparing svm and convolutional networks for epileptic seizure prediction from intracranial eeg. In: 2008 IEEE workshop on machine learning for signal processing. IEEE; 2008. p. 244–9.
[8] Mohamed A-r, Dahl GE, Hinton G. Acoustic modeling using deep belief networks. IEEE Trans Audio Speech Language Proc 2012;20(1):14–22.
[9] Ngiam J, Khosla A, Kim M, Nam J, Lee H, Ng AY. Multimodal deep learning. In: Proceedings of the 28th international conference on machine learning (ICML-11). p. 689–96.
[10] Lv Y, Duan Y, Kang W, Li Z, Wang F-Y. Traffic flow prediction with big data: a deep learning approach. IEEE Trans Intell Transp Syst 2015;16(2):865–73.
[11] Hinton GE, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. Neural Comput 2006;18(7):1527–54.
[12] Bengio Y, Yao L, Alain G, Vincent P. Generalized denoising auto-encoders as generative models. In: Advances in neural information processing systems. p. 899–907.
[13] Hu G, Yang Y, Yi D, Kittler J, Christmas W, Li SZ, et al. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In: Proceedings of the IEEE international conference on computer vision workshops. p. 142–50.
[14] Rhu M, Gimelshein N, Clemons J, Zulfiqar A, Keckler SW. Virtualizing deep neural networks for memory-efficient neural network design. arXiv preprint arXiv:1602.08124.
[15] Coelho VN, Coelho IM, Coelho BN, Reis AJ, Enayatifar R, Souza MJ, et al. A self-adaptive evolutionary fuzzy model for load forecasting problems on smart grid environment. Appl Energy 2016;169:567–84. http://dx.doi.org/10.1016/j.apenergy.2016.02.045.
[16] Kirk DB, Wen-mei WH. Programming massively parallel processors: a hands-on approach. 2nd ed. Morgan Kaufman; 2012.
[17] Coelho IM, Munhoz PLA, Haddad MN, Coelho VN, Silva MM, Souza MJF, et al. A computational framework for combinatorial optimization problems. In: VII ALIO/EURO workshop on applied combinatorial optimization, Porto. p. 51–4.
[18] Kolter JZ, Johnson MJ. Redd: a public data set for energy disaggregation research. In: Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA.
[19] Pipattanasomporn M, Kuzlu M, Rahman S. An algorithm for intelligent home energy management and demand response analysis. IEEE Trans Smart Grid 2012;3(4):2166–73. http://dx.doi.org/10.1109/TSG.2012.2201182.
[20] Armel KC, Gupta A, Shrimali G, Albert A. Is disaggregation the holy grail of energy efficiency? The case of electricity. Energy Policy 2013;52:213–34. http://dx.doi.org/10.1016/j.enpol.2012.08.062 [special Section: Transition Pathways to a Low Carbon Economy].
[21] Zou Y, Zhou X, Ding G, He Y, Jia M. A GPGPU-based collision detection algorithm. In: ICIG '09. Fifth international conference on image and graphics, 2009. p. 938–42. http://dx.doi.org/10.1109/ICIG.2009.127.
[22] Nedjah N, Macedo Mourelle L. High-performance hardware of the sliding-window method for parallel computation of modular exponentiations. Int J Parallel Programming 2009;37(6):537–55. http://dx.doi.org/10.1007/s10766-009-0108-7.
[23] Flynn MJ. Some computer organizations and their effectiveness. IEEE Trans Comput 1972;C-21(9):948–60.
[24] Fowers J, Brown G, Cooke P, Stitt G. A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In: Proceedings of the ACM/SIGDA international symposium on field programmable gate arrays. FPGA '12. New York, NY, USA: ACM; 2012. p. 47–56. http://dx.doi.org/10.1145/2145694.2145704.
[25] Veit A, Goebel C, Tidke R, Doblander C, Jacobsen H-A. Household electricity demand forecasting: benchmarking state-of-the-art methods. In: Proceedings of the 5th international conference on future energy systems. e-Energy '14. New York, NY, USA: ACM; 2014. p. 233–4. http://dx.doi.org/10.1145/2602044.2602082.
[26] Gradl S, Kugler P, Lohmüller C, Eskofier B. Real-time ecg monitoring and arrhythmia detection using android-based mobile devices. In: 2012 annual international conference of the IEEE engineering in medicine and biology society. p. 2452–5.
[27] Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, et al. Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. ACM; 2016. p. 26–35.