GPU accelerated modelling and forecasting for large time series

Christos K. Filelis - Papadopoulos[1][0000-0002-6591-970X], John P. Morrison[1][0000-0002-0134-6025], and Philip O'Reilly[2][0000-0001-9286-2107]

[1] University College Cork, Department of Computer Science, Western Gateway Building, Cork, Ireland `christos.papadopoulos-filelis@cs.ucc.ie`, `j.morrison@cs.ucc.ie`

[2] University College Cork, Cork University Business School, O'Rahilly Building, Cork, Ireland
`philip.oreilly@ucc.ie`

Abstract. Modelling of large scale data series is of significant importance in fields such as astrophysics and finance. The continuous increase in available data requires new computational approaches such as the use of multicore processors and accelerators. Recently, a novel time series modelling and forecasting method was proposed, based on a recursively updated pseudoinverse matrix which enhances parsimony by enabling assessment of basis functions, before inclusion into the final model. Herewith, a novel GPU (Graphics Processing Unit) accelerated matrix based auto-regressive variant is presented, which utilizes lagged versions of a time series and interactions between them to form a model. The original approach is reviewed and a matrix multiplication based variant is proposed. The GPU accelerated and hybrid parallel versions are introduced, utilizing single and mixed precision arithmetic to increase GPU performance. Discussions around performance improvement and high order interactions are given. A block processing approach is also introduced to reduce memory requirements for the accelerator. Furthermore, the inclusion of constraints in the computation of weights, corresponding to the basis functions, with respect to the parallel implementation are discussed. The approach is assessed in a series of model problems and discussions are provided.

Keywords: Forecasting · Pseudoinverse matrix · Parallel Modelling · GPU acceleration

1 Introduction

Modelling and forecasting time series has several applications is a number of scientific fields including signal processing, computational finance and astrophysics. Modelling of time series can be performed with traditional approaches [21], such as Auto-Regressive Integrated Moving Average (ARIMA) models [3] and Exponential Smoothing (ES) [4, 16], or machine learning methods such as Long Short - Term Memory Networks (LSTM) [15] and Support Vector Regression (SVR)

[8]. Another important family of techniques are based on orthogonalization of a set of basis functions, such as Fast Orthogonal Search [17] or Matching Pursuit [22]. These techniques construct a model from a set of basis functions (linear or non-linear) using orthogonalization procedures such as Gram-Schmidt orthogonalization. Recently, a recursive Schur complement pseudoinverse approach for modelling time series was introduced [12]. This approach avoids orthogonalization, while enabling the use of preconditioned iterative methods for reducing memory requirements in the case of a large number of basis functions [10]. These methods enable the use of arbitrary basis functions including linear, trigonometric, auto-regressive or machine learning based [12, 13, 11].

The emergence of big data led to an increase in interest in the area of parallel computing in order to reduce processing times. Extensive research has been carried out in the parallelization of machine learning methods, especially neural networks in multicore systems, distributed systems and accelerators such as GPUs [2, 18, 1, 23]. Parallelization was used to reduce training times especially for deep neural network architectures and for very large input datasets. GPU acceleration has been also utilized to reduce training and optimization times for Support Vector Machines [25, 24, 19]. In the majority of these approaches, the training operations are reformed to take advantage of matrix by matrix (BLAS3) kernels that can be efficiently parallelized in GPUs and modern multicore hardware. Another important modification is the use of mixed precision arithmetic, combining half-precision, single precision and double precision arithmetic substantially improving performance and storage requirements [20, 1]. In the literature, efforts have been directed also in the parallelization of Matching Pursuit type methods [9, 6] for GPUs. Despite the extensive literature and software available for parallelizing machine learning methods, literature around parallelization of techniques such as Fast Orthogonal Search is limited.

Herewith, we propose a novel parallel implementation of the recently proposed recursive Schur complement pseudoinverse matrix modelling based on auto-regressive basis functions. Initially, the method is recast to take advantage of BLAS3 operations, during the basis search operation, while avoiding redundant computations which will increase computational work and memory requirements. Then, the parallel algorithm, that utilizes mixed precision arithmetic, is presented and discussed along with a pure GPU implementation and block mixed precision variants. Multiplicative interactions between auto-regressive basis functions are also discussed. Implications related to precision and memory transfers between CPU and GPU are analyzed. The proposed scheme is assessed by modelling and forecasting two time series with different characteristics and sets of candidate basis functions. Scalability results are also presented and discussed.

In Section 2, the recursive Schur complement based pseudoinverse matrix of basis functions is reviewed and insights on the basis functions selection, higher order basis interactions and termination criteria is given. In Section 3, the matrix based variant is proposed and the CPU/GPU and pure GPU implementations are presented. Moreover, a block variant is given along with discussions on memory requirements, data transfer overhead and the effects of mixed precision arithmetic. In Section 4, numerical results are presented depicting the applicability and accuracy of the proposed scheme along with discussions on scalability and implications of mixed precision arithmetic.

2 Recursive Schur complement based time - series modelling

The coefficients of a model, corresponding to the time series y, with linearly independent basis functions stored in the columns of a matrix X can be computed as follows:

$$Xa = y \iff a = X^{+}y \iff a = (X^TX)^{-1}X^Ty, \tag{1}$$

where a is a vector of length n retaining the coefficients corresponding to the n basis functions retained in X. The matrix X^TX and its inverse are Symmetric Positive Definite [10]. In most cases all basis functions are not known "a priori" or their contribution to error reduction is not significant. In order to avoid such issues a recursive pseudoinverse matrix approach has been proposed [12] based on a symmetric variant of the matrix preconditioning technique introduced in [10]. Following this approach and given an additional basis F, with $X_{i+1} = [X_i \;\; F]$, $1 \le i \le n$, we have:

$$a_{i+1} = [X_i \;\; F]^{+}y = G_{i+1}D_{i+1}^{-1}G_{i+1}^TX_{i+1}^Ty, \tag{2}$$

or equivalently:

$$\begin{bmatrix} a_i \\ b \end{bmatrix} = \begin{bmatrix} G_i & -G_iD_i^{-1}G_i^TX_i^TF \\ 0 & 1 \end{bmatrix} \begin{bmatrix} D_i^{-1} & 0 \\ 0 & s_{i+1}^{-1} \end{bmatrix} \begin{bmatrix} G_i^T & 0 \\ -F^TX_iG_iD_i^{-1}G_i^T & 1 \end{bmatrix} \begin{bmatrix} X_i^Ty \\ F^Ty \end{bmatrix}, \tag{3}$$

where $(X_i^TX_i)^{-1} = G_iD_i^{-1}G_i^T$ and $s_{i+1} = F^TF - F^TX_iG_iD_i^{-1}G_i^TX_i^TF$ denotes the Schur Complement corresponding to the addition of basis function F. The initial conditions for the recursive formulation are $G_{1,1} = 1$, $D_{1,1} = s_1^{-1} = (F^TF)^{-1}$ and $a_1 = s_1^{-1}F^Ty$. The updated set of coefficients a_{i+1} can be computed alternatively using the following equations:

$$a_{i+1} = \begin{bmatrix} a_i^* \\ b \end{bmatrix} = \begin{bmatrix} a_i + g_{i+1}b \\ s_{i+1}^{-1}(F^T + g_{i+1}^TX_i^T)y \end{bmatrix}, \tag{4}$$

with:

$$G_{i+1} = \begin{bmatrix} G_i & g_{i+1} \\ 0 & 1 \end{bmatrix} \quad \textit{and} \quad D_{i+1}^{-1} = \begin{bmatrix} D_i^{-1} & 0 \\ 0 & s_{i+1}^{-1} \end{bmatrix}, \tag{5}$$

where b denotes the coefficient corresponding to basis function F and a_i^* is the updated coefficients after addition of basis function F. The vector $g_{i+1} = -G_iD_i^{-1}G_i^TX_i^TF$ corresponds to the $(i+1)$ column and $s_{i+1} = F^T(F + X_ig_{i+1})$ is the Schur complement. The modelling error $\rho_{i+1} = \|r_{i+1}\|_2^2$ corresponding to the addition of a basis function F can be computed as:

$$\|r_{i+1}\|_2^2 = \|y - X_{i+1}a_{i+1}\|_2^2 = \|r_i\|_2^2 - \|(X_ig_{i+1} + F)b\|_2^2 = \|r_i\|_2^2 - b^Ts_{i+1}b, 0 \le i \le n-1 \tag{6}$$

with $\|r_0\|_2^2 = \|y\|_2^2$. The quantity $e_{i+1} = b^T s_{i+1} b$ denotes the error reduction corresponding to the addition of a basis function F [12]. In order to ensure positive definiteness of the dot product matrix $X_{i+1}^T X_{i+1}$ the quantity e_{i+1} should be bounded by $0 \leq e_{i+1} \leq \|r_i\|_2^2$. It should be noted that $\rho_{i+1} = \|r_{i+1}\|_2^2 = T \cdot MSE$ is the Squared Error, T is the number of samples in time series y, and MSE denotes the Mean Squared Error. Detailed description and additional discussions regarding the method are given in [12, 13, 11].

2.1 Assessment and selection of basis functions

The explicit expression of error reduction can be used to select a subset of basis functions to form a model from a candidate set U retaining N basis functions. Trigonometric, exponential and linear functions have been considered for modelling in [12], [13], while a Non-Negative Adaptive Auto - Regression approach was followed in [11]. The procedure of selecting an appropriate basis requires computation of the potential error reduction for each member of the set U. This procedure is algorithmically described in Alg. 1. The procedure, described by Alg. 1, proceeds through all candidate basis in U sequentially, storing the respective error reductions to vector u. Then, the algorithm proceeds by selecting the index of the basis function that lead to maximum error reduction under the constraints that ensure positive definiteness.

Algorithm 1 Basis Search
$(k = bs(y, G_i, D_i^{-1}, X_i, U, \rho_i))$

```
1: Let $N$ denote the number of candidate basis functions in $U$.
2: $e_i = 0, 1 \leq i \leq N$
3: for $i \in [1, N] \subset \mathbb{N}$ do
4:     $F = U_i$
5:     $g_{i+1} = -G_i D_i^{-1} G_i^T X_i^T F$
6:     $s_{i+1} = F^T (F + X_i g_{i+1})$
7:     $b = s_{i+1}^{-1} (F^T + g_{i+1}^T X_i^T) y$
8:     $e_i = b^T s_{i+1} b$
9: end for
10: $k = \arg\max_{i \in [1,N]} e_i$ under the constraint $0 \leq e_i \leq \rho_i$
```

The set U can host any type of basis functions even Machine Learning models such as Support Vector Machines [8]. In the current manuscript we focus on lagged basis function of the form:

$$U = [y_{-1} \;\; y_{-2} \;\; y_{-3} \;\; y_{-4} \;\; \ldots \;\; y_{-N}], \tag{7}$$

with $y_0 = y$. In practice, the number of samples in lagged time series y_{-i} is $n - N$, since the latest sample is removed (retained in the responses y), along with the first $N-1$ samples from each candidate lagged basis to ensure that there are no missing data. In case multiplicative interactions between basis functions are allowed [14], e.g. $y_i y_j \ldots$ the number of basis functions into the candidate set are:

$$\binom{N}{k} + N\,k, \; k > 1 \tag{8}$$

where k is the order of allowed interactions. In the case of $k = 1$, then the number of candidate basis functions is equal to N.

Additional constraints can be imposed during the selection of a basis function that leads to the maximum error reduction. These constraints can be imposed during step 10 of Alg. 1. Examples of constraints include non-negativity of the coefficients [11] or imposing a threshold on their magnitude.

After selection of an appropriate basis function or lag, addition of this basis function has to be performed and the corresponding matrices to be updated. Several basis functions can be fitted by executing the process described by Alg. 1 followed by the process of Alg. 2, iteratively. The fitting process is terminated based on criteria regarding the fitting error, e.g. [12]:

$$\sqrt{\rho_{i+1}} < \epsilon\sqrt{\rho_0}. \tag{9}$$

Another approach is to terminate the fitting process based on the magnitude of the coefficients:

$$|b| < \epsilon|a_1|. \tag{10}$$

where b is the coefficient corresponding to the $i+1$ added basis function, while a_1 is the coefficient corresponding to the basis function added first. In both termination criteria ϵ is the prescribed tolerance. It should be noted that the second criterion is more appropriate in the case of lagged basis.

Algorithm 2 Add Basis Function
$([G_{i+1}, D_{i+1}^{-1}, X_{i+1}, a_{i+1}, \rho_{i+1}] = addbasis(y, G_i, D_i^{-1}, X_i, F, a_i, \rho_i))$

```
1: $g_{i+1} = -G_i D_i^{-1} G_i^T X_i^T F$
2: $s_{i+1} = F^T(F + X_i g_{i+1})$
3: $b = s_{i+1}^{-1}(F^T + g_{i+1}^T X_i^T)y$
4: Check termination criterion and terminate if met
5: $a_{i+1} = \begin{bmatrix} a_i \\ b \end{bmatrix}$
6: $\rho_{i+1} = \rho_i - b^T s_{i+1} b$
7: $G_{i+1} = \begin{bmatrix} G_i & g_{i+1} \\ 0 & 1 \end{bmatrix}$; $D_{i+1} = \begin{bmatrix} D_i^{-1} & 0 \\ 0 & s_{i+1}^{-1} \end{bmatrix}$; $X_{i+1} = \begin{bmatrix} X_i & F \end{bmatrix}$
```

3 Performance optimization and parallelization

The procedure, described by Alg. 1, proceeds through all candidate basis in U sequentially. It can be performed in parallel by assigning a portion of basis functions to each thread of a multicore processor. However, the most computationally intensive operations are matrix by vector and vector by vector, which are BLAS2 (Basic Linear Algebra Subroutines - Type 2) and BLAS1 operations, respectively, [5]. These operations lead to decreased performance compared to operations between matrices which are BLAS3 operations, since they require increased memory transfers and cache tiling and data re-use is limited [7]. Thus, to increase performance the most computationally intensive part, which is the basis search described by Alg. 1, has to be redesigned in order to compute the corresponding error for all candidate basis functions.

Let us consider a set of N candidate basis functions, represented as vectors of length $N-n$, stored in the columns of matrix U $((n-N)\times N)$. The columns $g_{i+1}^{(j)}, 1 \leq j \leq N$ corresponding to each basis can be computed by the following matrix multiplication operations:

$$\mathfrak{g}_{i+1} = [g_{i+1}^{(1)} \;\; g_{i+1}^{(2)} \;\; \dots \;\; g_{i+1}^{(N)}] = -G_i D_i^{-1} G_i^T X_i^T U. \tag{11}$$

The matrix $\mathfrak{g}_{i+1}$ $(i \times N)$ is formed by four dense matrix multiplications, however matrix D_i^{-1} is diagonal matrix, thus it can be retained as a vector. Multiplying matrix D_i^{-1} by another matrix is equivalent to multiplying the elements of each row j with the corresponding element $d_{j,j}^{-1}$ of the vector retaining the elements of the diagonal matrix. For the remainder of the manuscript we will denote this operation as $(\star)$. This operation can be used in the process described in Alg. 2. This reduces operations required, as well as storage requirements, since a matrix multiplication is avoided and the computation can be performed in place in memory.

Following computation of the columns stored in matrix $\mathfrak{g}_{i+1}$, the Schur Complements $s_{i+1}^{(j)}, 1 \leq j \leq N$ corresponding to the candidate basis functions are computed as follows:

$$\mathfrak{s}_{i+1} = diag(\tilde{\mathfrak{s}}_{i+1}) = diag(U^T(U + X_i \mathfrak{g}_{i+1})). \tag{12}$$

The formula $U^T(U + X_i\mathfrak{g}_{i+1})$ leads to the computation of Schur complement of the block incorporation of basis and not the individual Schur complements corresponding to the candidate basis function, which are stored in the diagonal of the result. The Schur complement $\tilde{\mathfrak{s}}_{i+1}$ is a dense matrix of dimensions $N \times N$ and requires substantial computational effort. In order to avoid unnecessary operations each diagonal element can be computed as follows:

$$(\mathfrak{s}_{i+1})_j = (U^T)_{j,:}((U)_{:,j} + X_i(\mathfrak{g}_{i+1})_{:,j}), \tag{13}$$

where $(\,.\,)_{i,j}$ denotes an element at position (i,j) of a matrix and (:) denotes all elements of a row or column of a matrix. In order to compute all elements of the diagonal concurrently, eq. (13) can be reformed as follows:

$$\mathfrak{s}_{i+1} = (U^T \odot (U + X_i \mathfrak{g}_{i+1})^T)v, \tag{14}$$

where $\odot$ denotes the Hadamard product of two matrices and v is a vector of the form $[1 \;\; 1 \;\; \dots \;\; 1]^T$. The vector $\mathfrak{s}_{i+1}$ $(N \times 1)$ retains the Schur complements corresponding to the candidate basis functions in U. Dedicated (Optimized) Hadamard product is not included in the standard BLAS collection, however it is included in vendor versions or CUDA (Compute Unified Device Architecture). Following the same notation the coefficients corresponding to the basis functions in set U can be computed as:

$$\mathfrak{b} = \mathfrak{s}_{i+1}^{-1} \odot (U^T + \mathfrak{g}_{i+1}^T X_i^T)y \tag{15}$$

and the corresponding error reductions as:

$$\mathfrak{e} = \mathfrak{b} \odot \mathfrak{s}_{i+1} \odot \mathfrak{b}. \tag{16}$$

The most appropriate basis function is selected by finding the maximum error reduction in vector $\mathfrak{e}$. The matrix based basis selection procedure is algorithmically described in Alg. 3.

Algorithm 3 Matrix Based Basis Search
$(k = mbbs(y, G_i, D_i^{-1}, X_i, U, \rho_i))$

```
1: Let $N$ denote the number of candidate basis functions in $U$.
2: $v_i = 1, 1 \leq i \leq N$
3: $\mathfrak{g}_{i+1} = -G_i(D_i^{-1} \star (G_i^T X_i^T U))$
4: $\tilde{U} = U + X_i \mathfrak{g}_{i+1}$
5: $\mathfrak{s}_{i+1} = (U^T \odot \tilde{U})v$
6: $\mathfrak{b} = \mathfrak{s}_{i+1}^{-1} \odot \tilde{U}^T y$
7: $\mathfrak{e} = \mathfrak{b} \odot \mathfrak{s}_{i+1} \odot \mathfrak{b}$
8: $k = \arg\max_{i \in [1,N]} \mathfrak{e}_i$ under the constraint $0 \leq \mathfrak{e}_i \leq \rho_i$
```

A block variant of Alg. 3 can be utilized to process batches of candidate functions. This can be performed by splitting matrix U into groups, processing them and accumulating the corresponding error reductions in vector $\mathfrak{e}$ before computing the index of the most effective basis function. Despite the advantages in terms of performance, the matrix and block based matrix approaches have increased memory requirements. The memory requirements are analogous to the number of candidate basis functions, since they have to be evaluated before assessment, while in the original approach each candidate basis is evaluated only before its assessment. Thus, the matrix approach requires $\mathcal{O}(N(n-N))$, the block approach $\mathcal{O}(max(\nu(n-N)), b_s(n-N)))$ and the original approach $\mathcal{O}(\nu(n-N))$ 64-bit words, where ν denotes the number of basis functions included in the model and b_s the number of basis in each block.

3.1 Graphics Processing Unit Acceleration

The operations involved in Matrix Based Basis Search, given in Alg. 3, can be efficiently accelerated in a Graphics Processing Unit (GPU). However, most GPU units suffer from substantial reduction of the double precision performance, e.g. 32× in the case of double precision arithmetic (Geforce RTX 20 series). In order to mitigate this issue, 32-bit floating point operations and 16-bit half precision floating point operations are utilized. This gives rise to mixed precision computations, where GPU related operations are performed in reduced precision, while CPU related ones are performed in double precision arithmetic. This approach enables acceleration while minimizing round off errors from reduced precision computation.

The proposed scheme utilized a similar approach in order to accelerate the most computationally intensive part of the process, which is the basis search. Computations in the GPU require data movement from the main memory to the GPU memory, which is a time consuming operation, thus should be limited. For the case of of the Matrix Based Basis Search algorithm, data should be transferred in the GPU before processing. This includes the time series y, the matrix G_i and vector D_i, the matrix of included basis X_i and the previous modelling error ρ_i in order to mark basis that could hinder positive definiteness. Before copying these arrays to the GPU memory, they should be cast to single precision arithmetic to ensure increased performance during computations. The matrix U, retaining the candidate basis, and time series y can be transferred in the GPU before starting the fitting process, since they are "a priori" known, while all the other matrices should be updated after addition of new basis function to the model. However, the update process includes only a small number of values to be transferred at each iteration.

Algorithm 4 GPU accelerated modelling

```
1: Let $y$ denote the time series to be modelled, $N$ the maximum lag, $n$ the number
   of samples in $y$.
2: $v_i = 1, 1 \leq i \leq n - N$
3: $\mathbf{y} = cpu2gpu(single(y))$
4: $\mathbf{U} = cpu2gpu(single([y_{-1}\ \ y_{-2}\ \ \ldots\ \ y_{-N}])$
5: $\rho_0 = \|y\|_2^2;\ \ \boldsymbol{\rho} = cpu2gpu(single(\rho_0))$
6: $[G_1, D_1^{-1}, X_1, a_1, \rho_1] = addbasis(y, [\,], [\,], [\,], v, [\,], \rho_0)$
7: $\boldsymbol{G} = cpu2gpu(single(G_1));\ \ \boldsymbol{D}^{-1} = cpu2gpu(single(D_1))$
8: $\boldsymbol{X} = cpu2gpu(single(X_1));\ \ \boldsymbol{\rho} = cpu2gpu(single(\rho_1))$
9: for $i \in [1, N]$ do
10:     $\mathbf{k} = mbbs(\mathbf{y}, \boldsymbol{G}, \boldsymbol{D}^{-1}, \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\rho})$          ▷ GPU
11:     $k = gpu2cpu(\boldsymbol{k});\ \ F = y_{-k}$
12:     $[G_{i+1}, D_{i+1}^{-1}, X_{i+1}, a_{i+1}, \rho_{i+1}] = addbasis(y, G_i, D_i^{-1}, X_i, F, a_i, \rho_i)$
13:     $\boldsymbol{G} = update(single(G_{i+1}));\ \ \boldsymbol{D}^{-1} = update(single(D_{i+1}^{-1}))$
14:     $\boldsymbol{X} = update(single(X_{i+1}));\ \ \boldsymbol{\rho} = cpu2gpu(single(\rho_{i+1}))$
15: end for
```

The process is described in Alg. 4. The matrices and vectors stored in the GPU memory are given in bold. The process terminates if the termination criterion of eq. (10) is met during the basis addition process or line 12 of Alg. 4. It should be noted that the first basis included removes the mean value of the time series y. This is performed in line 6 where the *addbasis* function is invoked. In practice, addition of the first basis function is performed using the equations: $s_1 = F^T F, a_1 = s_1^{-1} F^T y, D_1^{-1} = s_1^{-1}, G_1 = 1$, where F is substituted with a vector $((n - N) \times 1)$ with all its components set to unity.

The functions *cpu2gpu* and *gpu2cpu* are used to transfer data from CPU memory to GPU memory and from GPU memory to CPU memory, respectively. Conversion of matrices, vectors and variables from double precision to single precision arithmetic is performed with function *single* and the function *update* is utilized to update matrices and vectors involved in the Matrix Based Basis Search performed in the GPU. At each iteration $2 + i + (n - N), 1 \leq i \leq \nu$ single precision floating point numbers need to be transferred to GPU memory, with ν denoting the number of basis functions included in the model.

A full GPU version of Alg. 4 can be also used, by forming and updating all matrices directly to the GPU memory. In this approach, the matrix of candidate matrices U and the time-series y need to be transferred to the GPU memory, before computation commences. The value of the first coefficient should also be transferred to the CPU since it is required by the termination criterion. Moreover, in every iteration the new coefficient has to be transferred to CPU in order to assess model formation through the termination criterion. This approach uses solely single precision arithmetic and is expected to yield slightly different results due to rounding errors.

4 Numerical results

In this section the applicability, performance and accuracy of the proposed scheme is examined by applying the proposed technique to two time series. The first time series is composed of large number of samples and lagged basis functions without multiplicative interactions are used as the candidate set. The scalability of different approaches is assessed with respect to single precision, mixed precision and double precision arithmetic executed either on CPU or GPU. The second time series has a reduced number of samples, however an extended set of lagged candidate basis functions, which include second order interactions, are included. The characteristics of the time series are given in Tab. 1. The two time series were extracted from R studio. The error measures used to assess the forecasting error was Mean Absolute Percentage Error (MAPE), Mean Absolute Deviation (MAE) and Root Mean Squared Error (RMSE):

$$MAPE = \frac{100}{T} \sum_{i=1}^{T} \frac{|y_i - \hat{y}_i|}{|y_i|}, \quad MAE = \frac{1}{T} \sum_{i=1}^{T} |y_i - \hat{y}_i|, \quad RMSE = \sqrt{\frac{1}{T} \sum_{i=1}^{T} (y_i - \hat{y}_i)^2} \tag{17}$$

where y_i are the actual values, $\hat{y}_i$ the forecasted values and T the length of the test set. All forecasts are performed out-of-sample using a multi-step approach without retraining. All experiments were executed on a system with an Intel Core i7 9700K 3.6-4.9 GHz CPU (8 cores) with 16 GBytes of RAM memory and an NVIDIA Geforce 2070 RTX (2304 CUDA Cores) with 8 GBytes of memory. All CPU computations were carried out in parallel using Intel MKL, while the GPU computations were carried out using NVIDIA CUDA libraries.

Table 1: Model time series with description and selected splitting.

#	Name	Frequency	Train	Test	Description
1	**Call volume for a large North American bank**	5-mins	22325	5391	Volume of calls, per five minute intervals, spanning 164 days starting from 3 March 2003
2	**Daily female births in California**	Daily	304	61	Daily observations in 1959

Different notation is used for the variants of the proposed scheme:
CPU-DP: Matrix based CPU implementation using double precision arithmetic. This is the baseline implementation.
CPU-SP: Matrix based CPU implementation using single precision arithmetic.
CPU-DP-GPU: Matrix based CPU/GPU implementation using mixed precision arithmetic. The basis search is performed in the GPU using single precision

arithmetic, while incorporation of the basis function is performed in double precision arithmetic.

CPU-DP-GPU-block(n_b): Block matrix based CPU/GPU implementation using mixed precision arithmetic. The basis search is performed in the GPU using single precision arithmetic, while incorporation of the basis function is performed in double precision arithmetic.

GPU: Pure matrix based GPU implementation using single precision arithmetic.

The parameter n_b denotes the number of blocks. The block approach requires less GPU memory.

4.1 Time series 1 - Scalability and accuracy

The average value of the dataset is 192.079, the minimum value is 11 and the maximum value is 465. For this model the lagged candidate basis has been utilized, while higher degree interactions are not allowed, resulting in an additive model. The performance in seconds for all variants is given in Fig. 1, while speedups are presented in Fig. 2. The pure GPU and CPU-DP-GPU implementations have the best performance overall leading to the best speedups. The pure GPU implementation has a speedup greater than 20× for more than 50 basis functions with a maximum of approximately 27×, with respect to the baseline implementation. With respect to the CPU single precision arithmetic implementation the pure GPU approach has a speedup of approximately 10× for more than 50 basis functions. The CPU-DP-GPU has a maximum speedup of approximately 22× attained for 134 basis functions. After that point the speedup decreases because the double precision operations in the CPU do not scale with same rate, reducing the overall speedup. It should be noted that even for low number of basis functions, e.g. 6, the speedup of the pure GPU implementation is approximately 6× with respect to CPU-DP and approximately 4× with respect to CPU-SP implementation.

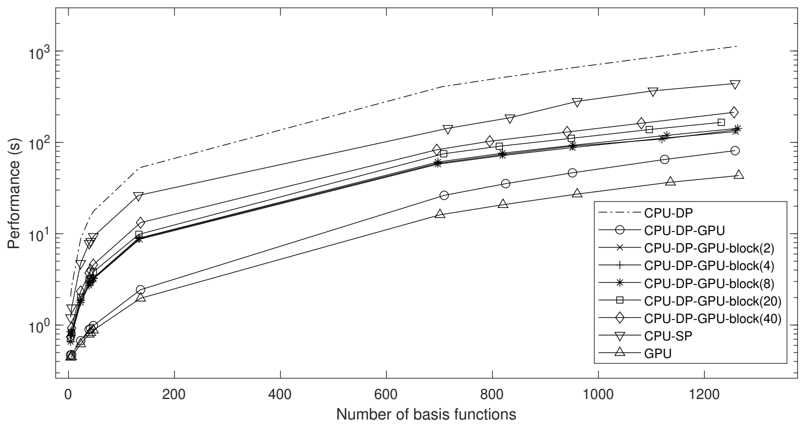

Fig. 1: Performance for all variants for different number of basis functions.

The performance of the block variants degrades when increasing the number of blocks retaining the candidate basis functions, since they require more data transfers between CPU and GPU. The speedups for the block variants range from 2.5 × − 8.6× over CPU-DP implementation and 1.5 × − 3.4× over the CPU-SP implementation.

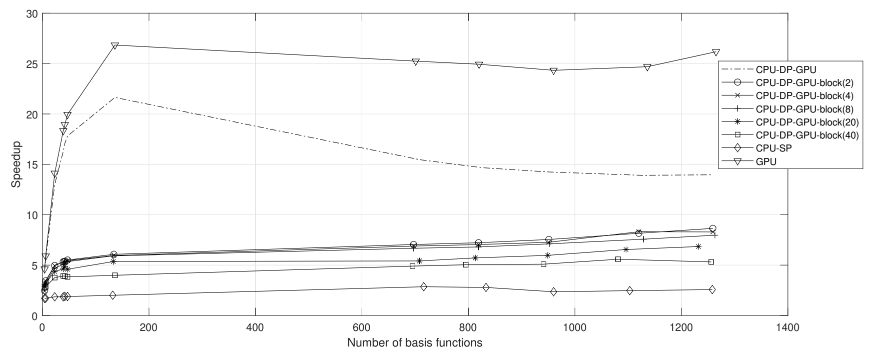

Fig. 2: Speedup for all variants for different number of basis functions.

Table 2: Minimum, maximum and average number of basis functions, RMSE, MAE and MAPE for all variants for the first time series.

	# Basis			**RMSE**			**MAE**			**MAPE**		
ϵ	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**
0.1	4	4	4.0	41.96	41.96	41.96	34.25	34.25	34.25	25.80	25.80	25.80
0.07	6	6	6.0	44.86	44.86	44.86	36.87	36.87	36.87	29.29	29.29	29.29
0.04	23	23	23.0	26.32	26.32	26.32	19.61	19.61	19.61	11.76	11.76	11.76
0.02	39	39	39.0	24.07	24.07	24.07	18.07	18.07	18.07	11.23	11.23	11.23
0.01	42	42	42.0	24.13	24.14	24.14	18.12	18.12	18.12	11.22	11.22	11.22
0.009	47	47	47.0	23.86	23.86	23.86	17.96	17.96	17.96	11.45	11.45	11.45
0.006	132	136	134.1	23.31	23.64	23.47	17.65	17.96	17.81	11.66	11.92	11.83
0.005	695	716	702.8	23.16	23.64	23.22	17.50	17.96	17.56	11.49	11.92	11.55
0.004	795	833	818.7	23.15	23.64	23.21	17.49	17.96	17.55	11.48	11.92	11.53
0.003	941	960	952.3	23.16	23.17	23.17	17.50	17.52	17.51	11.50	11.52	11.51
0.002	1081	1136	1116.0	23.16	23.17	23.16	17.49	17.50	17.50	11.46	11.47	11.47
0.001	1232	1266	1257.3	23.16	23.16	23.16	17.49	17.50	17.50	11.46	11.47	11.47

The number of basis functions included in the model along with forecasting errors are given in Tab. 2. The number of basis functions as well as the errors are not substantially affected by the use of mixed or single precision arithmetic. More specifically, up to approximately 50 basis functions all variants produce almost identical results. However, above 50 basis functions there is a minor difference in the number of basis functions included in the model which in turn slightly affects the error measures. The difference in the number of included basis functions is caused by rounding errors in the computation of error reduction ρ. This is caused by the rounding errors in the formation of the column vectors $\mathfrak{g}_{i+1}$, involved in the computation of respective Schur complements and potential basis coefficients.

An important observation is that the error measures regarding forecasts do not significantly reduce after the incorporation of approximately 130 basis functions. Thus, additional basis functions increase the complexity of the model. In order to ensure sparsity of the underlying model, a different termination criterion can be used, since the termination criterion of eq. (10) depends on the magnitude of the entries of the basis functions and is more susceptible to numerical rounding errors. The new termination criterion based on error reduction percentage is as follows:

$$\sqrt{\rho_i} - \sqrt{\rho_i - e_{i+1}} < \epsilon \sqrt{\rho_i}, \tag{18}$$

where e_{i+1} denotes the potential error reduction that will be caused by the incorporation of the $i+1$-th basis. $\epsilon \in [0,1] \subset \mathbb{R}$ denotes the acceptable percentage of error reduction to include a basis function. This criterion will be used to model the second time series along with higher level interactions.

4.2 Time series 2 - Flexibility and higher order interactions

The average value of the dataset is 41.9808, the minimum value is 23 and the maximum value is 73.

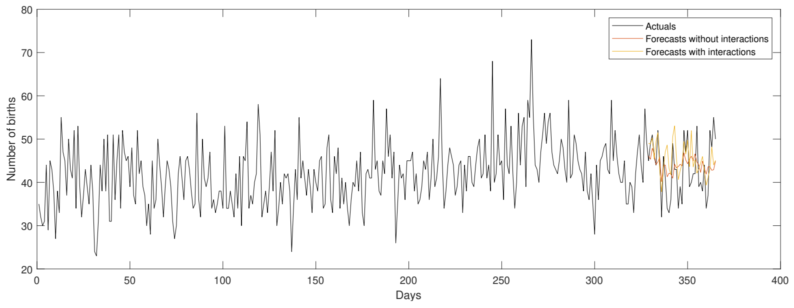

Fig. 3: Actual along with forecasted values with and without interactions.

The candidate set is composed of lagged basis functions and second order interactions of the form $y_j y_k$. The termination criterion of eq. (18) was used with $\epsilon = 0.002$, with maximum lags equal to 50. In Fig. 3 the actuals along with the forecasted values computed with and without interactions are given. The inclusion of second order interactions results in capturing the nonlinear behavior of the time series in the forecasted values. The error measures without interactions were: $RMSE = 5.96$, $MAE = 5.11$ and $MAPE = 12.33$, while with interactions the error measures were: $RMSE = 6.18$, $MAE = 4.93$ and $MAPE = 12.20$. The inclusion of higher order interactions led to reduction of the error measures and showcases the flexibility of the approach allowing for the inclusion of arbitrary order interactions in the candidate basis functions.

The execution time for CPU-SP, CPU-DP and GPU were 1.1621, 2.2076 and 0.3932, respectively. Thus, the speedup of the pure GPU variant was approximately 3× over the CPU-SP variant and 5.6× over the CPU-DP version. The pure GPU version is efficient even for time series with small number of samples, under a sufficiently sized space of candidate basis functions.

5 Conclusion

A matrix based parallel adaptive auto-regressive modelling technique has been proposed. The technique has been parallelized in multicore CPUs and GPUs and a block variant has been also proposed, based on a matrix (BLAS3) recast of the required operations. The pure GPU variant presented speedup up to 27× over the double precision arithmetic parallel CPU version and 10× over the parallel single precision CPU version for time series with large number of samples. The use of single and mixed precision did not affect substantially the forecasting error, rendering the technique suitable for modelling and forecasting large time series. Implementation details and discussions on higher order interactions between basis functions have been also given. The applicability and effectiveness of the method were also discussed and new termination criterion based on potential error reduction of basis functions, which is invariant to scaling, has been given.

Future work is directed towards the design of an improved basis search that will reduce the search space based on a tree approach. Moreover, backfitting procedures will be considered.

Acknowledgement

This research has emanated from research conducted with the financial support of Science Foundation Ireland under Grant 18/SPP/3459.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. p. 265–283. OSDI'16, USENIX Association, USA (2016)
2. Ben-Nun, T., Hoefler, T.: Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. ACM Comput. Surv. **52**(4) (aug 2019). https://doi.org/10.1145/3320060
3. Box, G.E.P., Jenkins, G.M.: Time Series Analysis Forecasting and Control. San Francisco: Holden Day (1976)
4. Brown, R.G.: Smoothing, Forecasting and prediction of discrete time series. Englewood Cliffs, NJ, Prentice Hall (1963)
5. Choi, J., Dongarra, J., Ostrouchov, S., Petitet, A., Walker, D., Whaley, R.C.: A proposal for a set of parallel basic linear algebra subprograms. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science. pp. 107–114. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
6. Dai, Y., He, D., Fang, Y., Yang, L.: Accelerating 2d orthogonal matching pursuit algorithm on gpu. J. Supercomput. **69**(3), 1363–1381 (sep 2014). https://doi.org/10.1007/s11227-014-1188-8
7. Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A.: The Sourcebook of Parallel Computing. Morgan Kaufmann (2002)

8. Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V.: Support vector regression machines. In: Mozer, M.C., Jordan, M., Petsche, T. (eds.) Advances in Neural Information Processing Systems. vol. 9. MIT Press (1997)
9. Fang, Y., Chen, L., Wu, J., Huang, B.: Gpu implementation of orthogonal matching pursuit for compressive sensing. In: 2011 IEEE 17th International Conference on Parallel and Distributed Systems. pp. 1044–1047 (2011). https://doi.org/10.1109/ICPADS.2011.158
10. Filelis-Papadopoulos, C.K.: Incomplete inverse matrices. Numerical Linear Algebra with Applications **28**(5), e2380 (2021). https://doi.org/10.1002/nla.2380
11. Filelis-Papadopoulos, C.K., Kirschner, S., O'Reilly, P.: Forecasting with limited data: Predicting aircraft co2 emissions following covid-19. Submitted (2021)
12. Filelis-Papadopoulos, C.K., Kyziropoulos, P.E., Morrison, J.P., O'Reilly, P.: Modelling and forecasting based on recurrent pseudoinverse matrices. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds.) Computational Science – ICCS 2021. pp. 229–242. Springer International Publishing, Cham (2021)
13. Filelis-Papadopoulos, C.K., Kyziropoulos, P.E., Morrison, J.P., O'Reilly, P.: Modelling and forecasting based on recursive incomplete pseudoinverse matrices. Mathematics and Computers in Simulation, accepted (2022)
14. Friedman, J.H.: Multivariate adaptive regression splines. The Annals of Statistics **19**(1), 1–67 (1991). https://doi.org/10.1214/aos/1176347963
15. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (11 1997). https://doi.org/10.1162/neco.1997.9.8.1735
16. Holt, C.C.: Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting **20**, 5–13 (2004)
17. Korenberg, M.J., Paarmann, L.D.: Orthogonal approaches to time-series analysis and system identification. IEEE Signal Processing Magazine **8**(3), 29–43 (1991)
18. Li, B., Zhou, E., Huang, B., Duan, J., Wang, Y., Xu, N., Zhang, J., Yang, H.: Large scale recurrent neural network on gpu. In: 2014 International Joint Conference on Neural Networks (IJCNN). pp. 4062–4069 (2014). https://doi.org/10.1109/IJCNN.2014.6889433
19. Li, Q., Salman, R., Test, E., Strack, R., Kecman, V.: Gpusvm: a comprehensive cuda based support vector machine package. Open Computer Science **1**(4), 387–405 (2011). https://doi.org/10.2478/s13537-011-0028-7
20. Lu, Y., Zhu, Y., Han, M., He, J.S., Zhang, Y.: A survey of gpu accelerated svm. In: Proceedings of the 2014 ACM Southeast Regional Conference. ACM SE '14, Association for Computing Machinery, New York, NY, USA (2014)
21. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The m4 competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting **36**(1), 54 – 74 (2020). https://doi.org/10.1016/j.ijforecast.2019.04.014, m4 Competition
22. Mallat, S., Zhang, Z.: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **41**(12), 3397–3415 (1993). https://doi.org/10.1109/78.258082
23. Oh, K.S., Jung, K.: Gpu implementation of neural networks. Pattern Recognition **37**(6), 1311–1314 (2004). https://doi.org/10.1016/j.patcog.2004.01.013
24. Paoletti, M.E., Haut, J.M., Tao, X., Miguel, J.P., Plaza, A.: A new gpu implementation of support vector machines for fast hyperspectral image classification. Remote Sensing **12**(8) (2020). https://doi.org/10.3390/rs12081257
25. Tan, K., Zhang, J., Du, Q., Wang, X.: Gpu parallel implementation of support vector machines for hyperspectral image classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **8**(10), 4647–4656 (2015)