Parallel Approach for Time Series Analysis with General Regression Neural Networks

J.C. Cuevas-Tello*[1], R.A. González-Grimaldo[1], O. Rodríguez-González[1], H.G. Pérez-González[1], O. Vital-Ochoa[1]

[1]Facultad de Ingeniería, Universidad Autónoma de San Luis Potosí
Av. Dr. Manuel Nava No.8, Zona Universitaria, 78290
San Luis Potosí, SLP, México
*cuevas@uaslp.mx

ABSTRACT
The accuracy on time delay estimation given pairs of irregularly sampled time series is of great relevance in astrophysics. However the computational time is also important because the study of large data sets is needed. Besides introducing a new approach for time delay estimation, this paper presents a parallel approach to obtain a fast algorithm for time delay estimation. The neural network architecture that we use is general Regression Neural Network (GRNN). For the parallel approach, we use Message Passing Interface (MPI) on a beowulf-type cluster and on a Cray supercomputer and we also use the Compute Unified Device Architecture (CUDA™) language on Graphics Processing Units (GPUs). We demonstrate that, with our approach, fast algorithms can be obtained for time delay estimation on large data sets with the same accuracy as state-of-the-art methods.

Keywords: neural networks, time series, parallel algorithms, machine learning

RESUMEN
La precisión para estimar retrasos en tiempo en series de tiempo muestreadas irregularmente es de gran importancia en astrofísica. Sin embargo, el tiempo computacional también es importante para el estudio de conjuntos de datos de gran tamaño. Este artículo primero presenta un nuevo método para estimar retrasos en tiempo, posteriormente se presenta una metodología basada en cómputo paralelo para estimar de manera rápida retrasos en tiempo. En ambos casos se utiliza una arquitectura de redes neuronales denominada regresión generalizada (General Regression Neural Networks — GRNN). Para el cómputo paralelo se utiliza MPI (Message Passing Interface) en un cluster tipo beowulf y en una supercomputadora Cray, también se utiliza el lenguaje CUDA™) (Compute Unified Device Architecture) para GPUs (Graphics Processing Units). Finalmente se demuestra empiricamente que con nuestra metodología se obtienen algoritmos rápidos para estimar retrasos en tiempo en conjuntos de datos de gran tamaño con la misma precisión que métodos que se usan en la actualidad.

1. Introduction

The time series analysis has great relevance in astrophysics [1]. Although diverse sciences study time series, in astrophysics time series analysis has special characteristics that makes it a challenging area open to research. The time series are irregularly sampled with several levels of noise and with missing data, also known as gaps [2, 3]. The problem consists in estimating time delays between pairs of time series [1, 4]. In Figures 1-4, we show graphically the time delay problem, see §2.3.

Upon predicting that the Hubble's parameter can be estimated through time delays on gravitational lenses [5], many observation campaigns have been launched since then [4], and new projects for ambitious surveys like Large Synoptic Survey Telescope (LSST) and the Super-Nova Acceleration Probe (SNAP) devoted to study dark matter are in development. Moreover, current surveys like The Sloan Digital Sky Survey (SDSS) and Sloan Lens ACS (SLACS) are generating a tremendous amount of large monitoring data sets. The above surveys are not only useful to estimate the Hubble's parameter, because they are also important to study lensed supernovae (SNe) [6]. Therefore, time delay estimations become a big issue to study dark matter and microlensing [7].

So far, methods to estimate time delays have concentrated on the accuracy of time delay

estimations [8], because the accuracy to measure dark matter depends precisely on the accuracy of time delay estimations [4].

Our approach is based on artificial neural networks, in particular, General Regression Neural Networks (GRNN) [9], which are based on Radial Basis Function neural networks (RBF) [10]. It has been shown that GRNN are more suitable for the time delay problem than backpropagation-based neural networks [11].

Several time delay methods have been proposed across the literature, including a survey of methods [8]. In the astrophysics literature, the most popular method based on correlation analysis is Dispersion Spectra [12]. Another popular method is the PRH method [12, 13]. In the machine learning literature, only two methods appear: one which is based on kernel methods and evolutionary algorithms [14, 15], and another which is based on Bayesian analysis [16].

In this paper, we compare the results obtained from several methods: Linear Interpolation (LI) [14], two versions of Dispersion spectra method D_1^2 and $D_{4,2}^2$ [12], PRH method [17, 13], Kernel-based method (K-V) and a kernel method with evolutionary computation (EA-M-CV) [15].

We compare the results with both artificial and real data. In particular, our research focus on large data sets so the time computation can be evaluated. The contribution of this paper extend in several directions:

1. The introduction of a fast method for time delay estimation based on GRNN, which is sequential.
2. The parallelization of the GRNN method.
3. The performance of the parallel GRNN with MPI running on a cluster and on a supercomputer.
4. A parallel version of GRNN running on GPUs.
5. The comparison of performance of GRNN with state-of-the-art-methods.

The remainder of the paper is organized as follows: the next section contains the description of the data sets used in this research. In Section 3, we describe our sequential algorithm for time delay estimation. Section 4 presents our parallel algorithms. It follows the experiments and results section, and finally it comes the conclusions and future work.

2. Time Series and Time Delay

This section describes the type of time series we studied. We also describe the time delay problem.

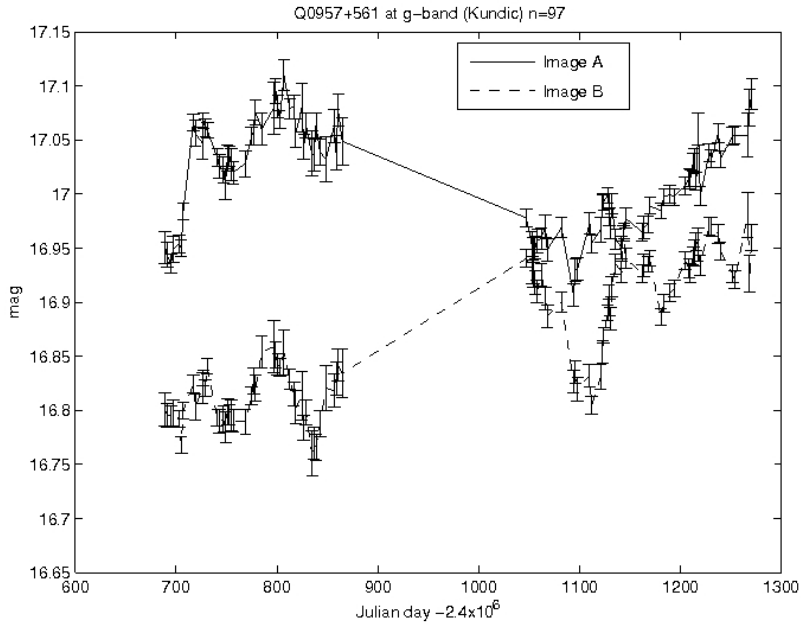


Figure 1. Kundic data. Real Data: Q0957+561. Optical data at g-band with 97 observations.

2.1 Real Data

First, we start describing real data for the quasar Q0957+561, which is the most studied quasar so far [4]. We present two data sets for real data: Kundic data [4] and Schild data[1]. We studied Kundic data because, apparently, it stopped a controversy regarding the definite time delay for the quasar Q0957+561 (see Fig. 1) making this data set and its time delay the most accepted across the literature [29]. Schild data is important because it is a large data set with 1,232 observations (see Fig. 2). Typical data sets for Q0957+561 are about one hundred observations, which are far from the new generations of observations such as the LSST, SNAP, SDSS and SLACS projects. These projects will generate data sets with thousands of observations automatically. Currently, the data sets for quasars are obtained manually [4]. In Figures 1–2, the *x*-axis represents the time when the source of light is observed and the *y*-axis represents the flux *f* of light from a source, which is expressed in logarithmic units known as magnitudes (mag).

2.2 Artificial Data

Due to the importance of the study of time delay, many efforts have been made to estimate the time delay with real and artificial data [12, 13, 17, 20, 21]. The problem with real data is that the definite time delay estimation for most known gravitational lenses remains uncertain [20]. Thus, in this paper, we also study the GRNN on artificial data.

The public artificial data DS-5 has been generated to test algorithms for time delay estimation on gravitational lenses [14, 20]. These data simulate one observation every 1.3 years with 50 samples taken irregularly. The time delay is 5 days with a shift of M = 0.1, between image-A and image-B. These data are grouped in five different forms (see Fig. 3), tree noise levels 0.03%, 0.106% and 0.466% (see Fig. 4), fifty realizations per noise level and ten realizations per gap size.

We simulated gap size in observations by imposing five blocks of missing data. The blocks were

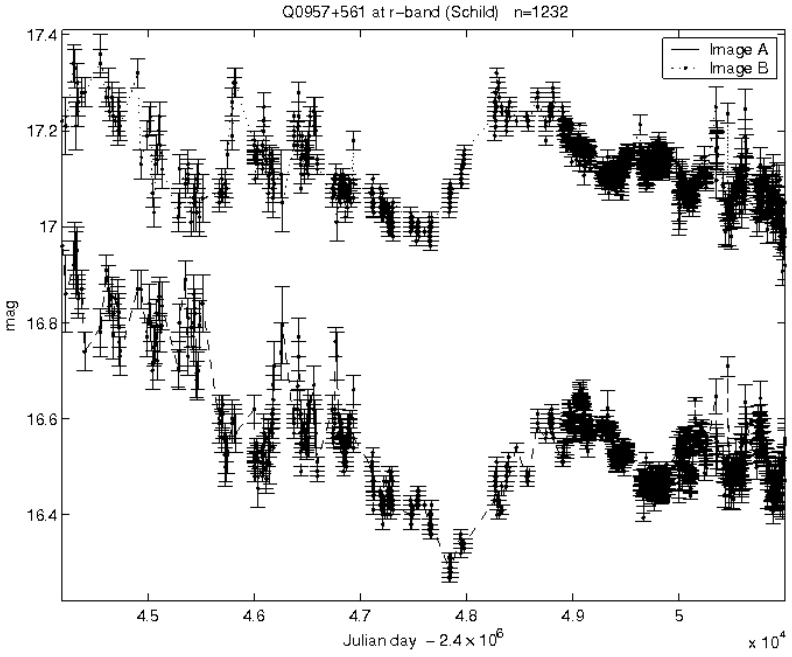


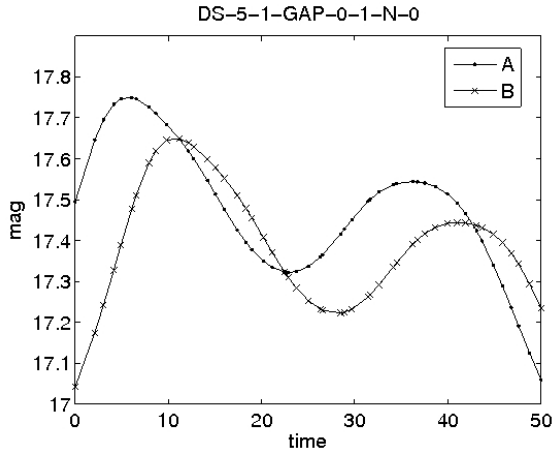
Figure 2. Schild Data. Real Data: Q0957+561. Image-A is shifted up 0.6 mag for clarity. Optical data at r-band with 1.232 observations.

[1] These data are unpublished data and collected by Schild et al. [18], and provided thanks to Somak Raychaudhury. Available on request.

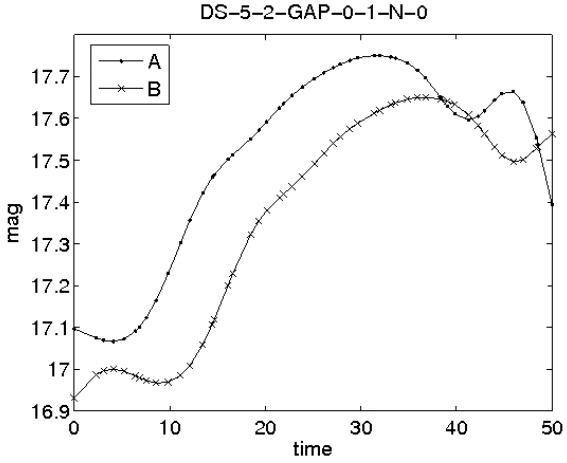
located randomly with at least one sample between them. We used gap sizes of 1, 2, 3, 4 and 5. Since this process is repeated ten times, we obtained ten pairs of time series with randomly located gaps.

For each noise level, there are fifty different realizations in which the percentage of noise is represented by the size of the error bars which, in turn, are proportional to the mag (*y*-axis).

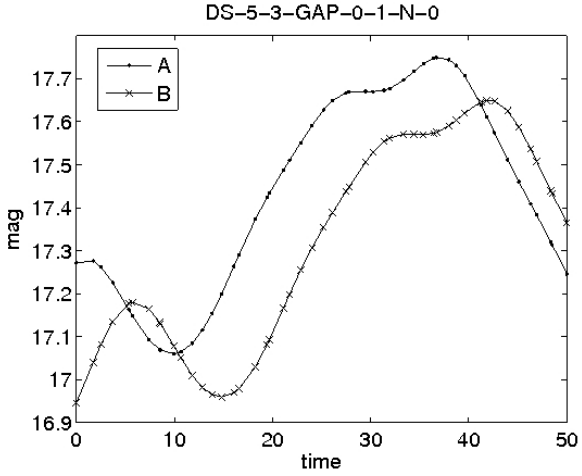
Considering all forms, the different noise levels and gap sizes, the DS-5 contains 38,505 pairs of time series (see Table 1). The true time delay is known to be five units. These data are shown in Fig. 5. These data simulate optical data with short time delay and high precision. Figure 5 shows realizations for different noise levels and gap sizes: (a) Form 3, no gaps and no noise, (b) Form 3, no gaps and noise level of 0.106%, (c) Form 3, no gaps and noise level of 0.466%, (d) Form 3, gap size 1, realization 1, noise level of 0.466%. (e) Form 3, gap size 1, realization 6, noise level of 0.466%, and (f) Form 3, gap size 5, realization 2, noise level of 0.106%.



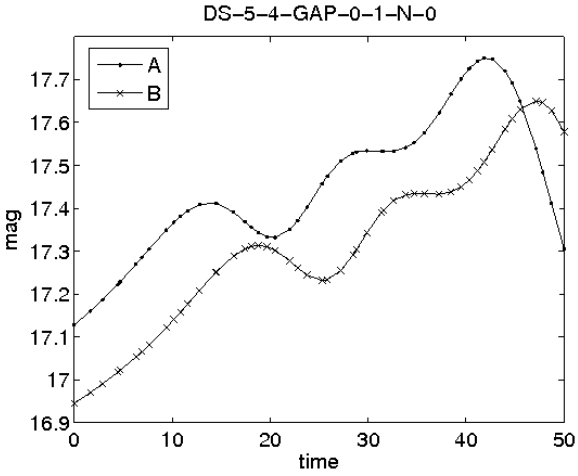
(a) Form 1.



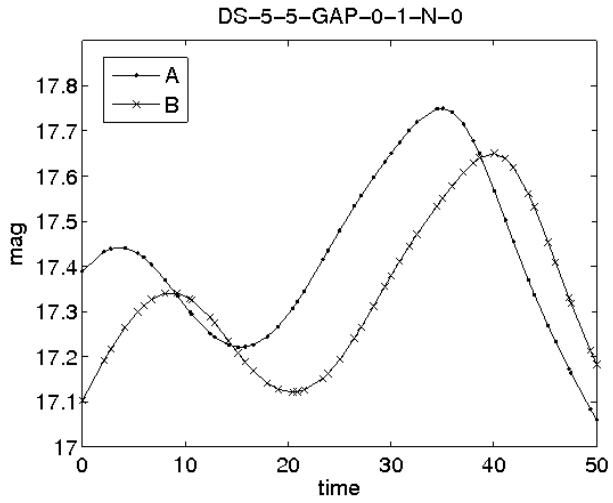
(b) Form 2.



(c) Form 3.



(d) Form 4.



(e) Form 5.

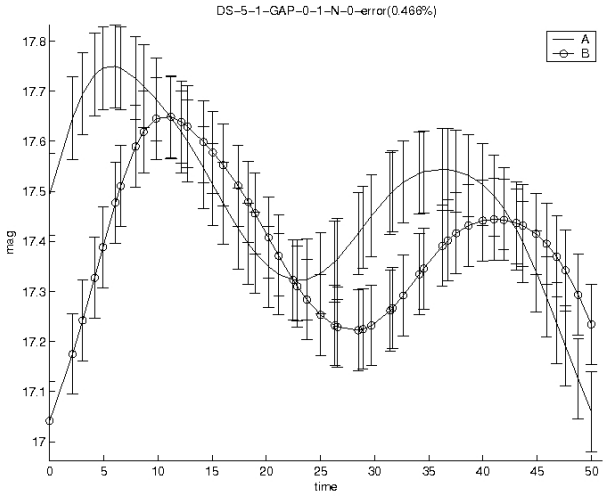
Figure 3. The artificial data DS-5. The data is grouped in five different forms or shapes. These examples do not contain noise.

2.3 Time Delay Problem

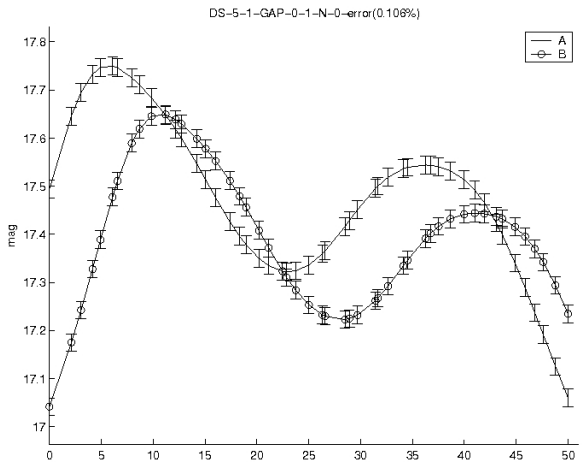
The main problem is the estimation of the time shift between pairs of time series. In Figure 3, it is clear that there is a time shift between A and B. The same occurs with Figures 1 and 2, which show real data. In fact, the presence of the time shift is clearer on artificial data, since real data have gaps and noise. The study of the time delay is important because it is a method to measure dark matter. The time delay is proportional to the mass that causes the time delay, which acts as a gravitational lens. This fact has many implications and applications in astrophysics.

	Gap Size					
Noise	**0**	**1**	**2**	**3**	**4**	**5**
0%	1	10	10	10	10	10
0.03%	50	500	500	500	500	500
0.106%	50	500	500	500	500	500
0.466%	50	500	500	500	500	500
SUB-TOTAL	151	1510	1510	1510	1510	1510

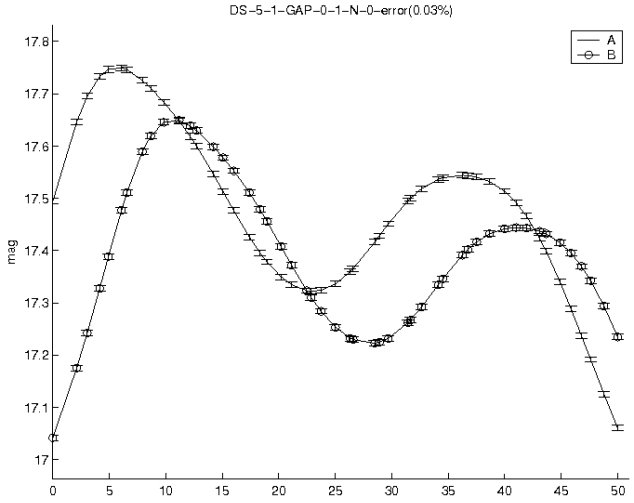
Table 1. Artificial Data DS-5. 7,701 pairs per form, generating 38,505 pairs of time series.



(a) Level of noise 1.



(b) Level of noise 2.



(c) Level of noise 3.

Figure 4. Form 1 with error bars, and different noise levels.

3. General Regression Neural Network (GRNN)

The GRNN model (see Fig. 6) is supported by the theory of non-linear regression theory [22]:

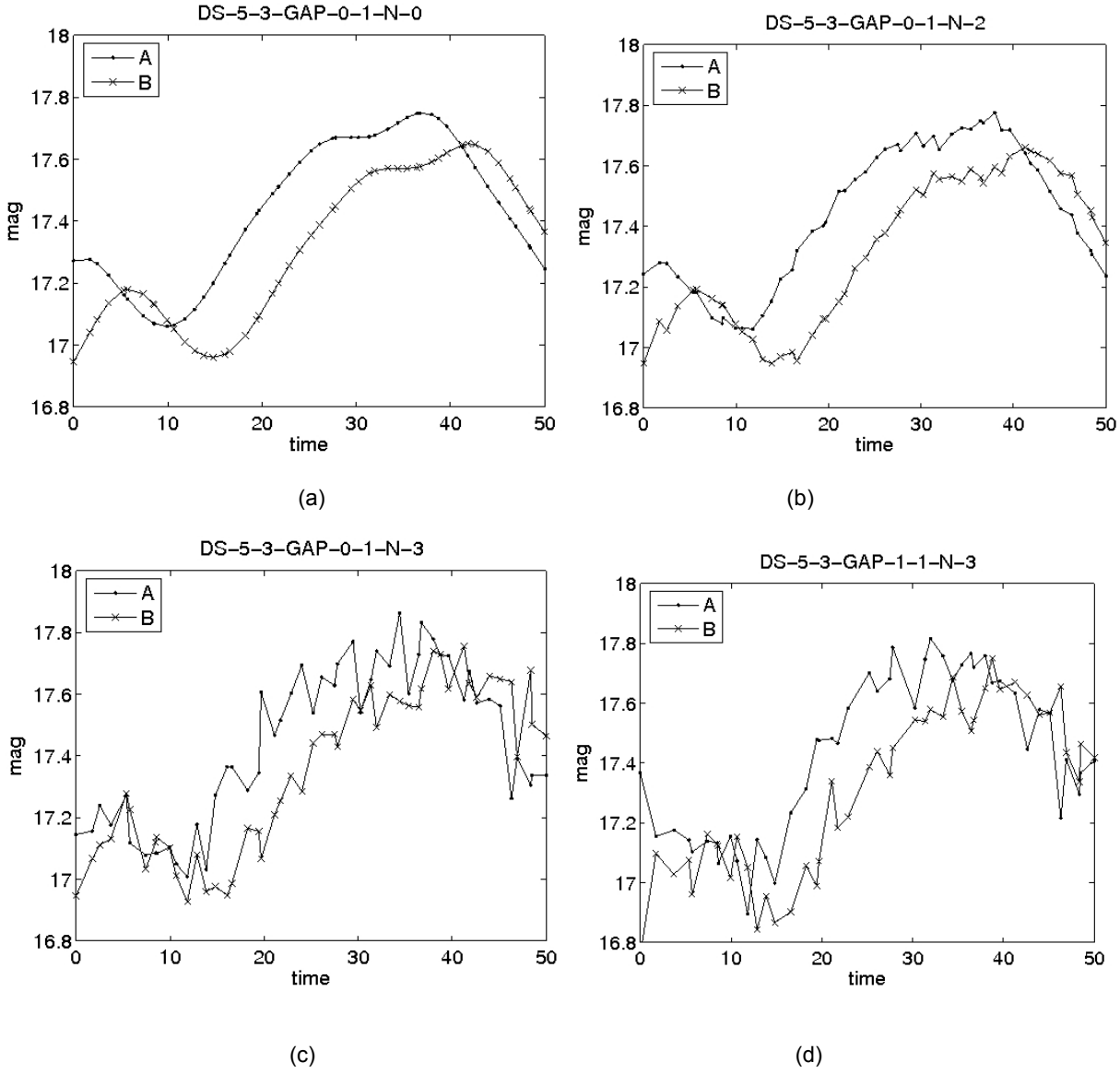
$$E(y|X) = \frac{\int_{-\infty}^{\infty} yf(X,y)dy}{\int_{-\infty}^{\infty} f(X,y)dy} \quad (1)$$

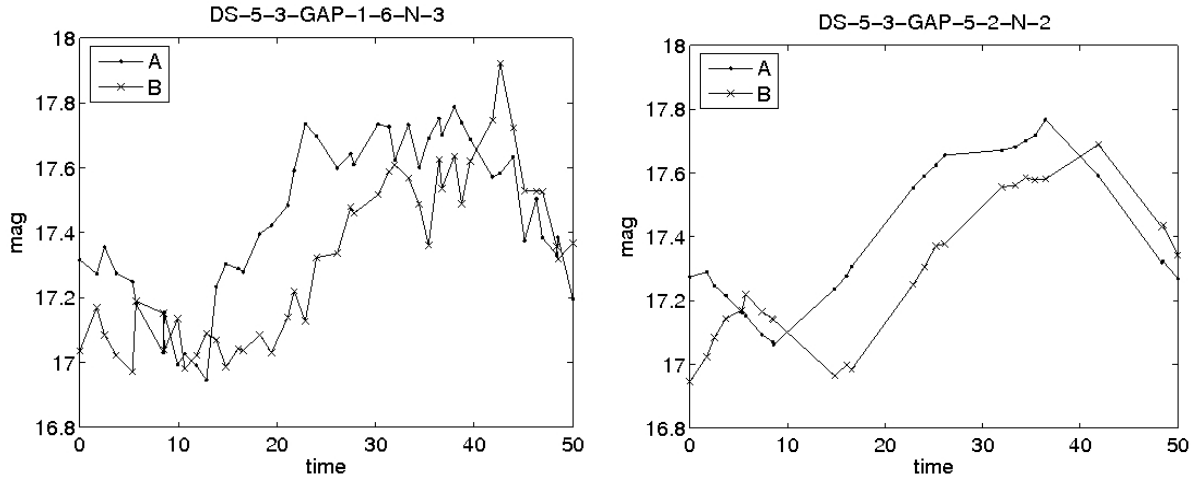
where $\boldsymbol{X}$ is the input vector (x_1 , x_2 , ..., x_d) with d inputs and y represents the output. $E(y|\boldsymbol{X})$ is the expected value of the output given the input $\boldsymbol{X}$, and $f(\boldsymbol{X},y)$ is the probability of a density function. By using a Parzen estimator to obtain $f(\boldsymbol{X},y)$ from the training data of size m, the output is as follows:

$$y(X) = E[y|X] = \frac{\sum_{i=1}^{m} w_i \phi_i}{\sum_{i=1}^{m} \phi_i} \quad (2)$$

The ϕ_i functions represent the radial functions, which give to RBF networks their name. In the kernel methods literature, these ϕ_i functions are known as kernel functions $k(\cdot,\cdot)$,where the ϕ function embeds the data into a feature space where the nonlinear pattern now appears linear [9,39]. Functions ϕ_i may have different forms including Gaussian, multicuadratic and inverse multicuadratic. However, the Gaussian functions are the most used in the literature, known as Parzen-Rosenblatt density estimator or Parzen window [10, 25]. Consequently, $\phi_i\,(n)$ is defined as follows:

$$\phi i\,(n) = \exp\left(\frac{-\|X(n)-c_i\|^2}{2\sigma_i^2}\right) \quad (3)$$





(e) (f)

Figure 5. These plots show the form 3 from DS-5 with different noise levels and gap sizes.

where $\|X(n) - c_i\|$ is the Euclidean distance between the input $X(n)$ and the center c_i (located at each observation of the training data). The standard deviation, spread, of the Gaussian function is σ_i (Parzen window size).

Employing the Gaussian functions as basis in Eq. 2, it is expressed as [22]:

$$y(X) = \frac{\sum_{i=1}^{m} w_i exp\left(\frac{-D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{m} exp\left(\frac{-D_i^2}{2\sigma^2}\right)} = \frac{S_w}{S_s} \quad (4)$$

where D_i is the Euclidean distance, similar to Eq. 3. Note that Eq. 4 is similar to Eqs. 5.134 and 5.138 [10], which correspond to the Nadaraya-Watson regression estimator and the normalized RBF network, respectively.

In practice, the advantage of the GRNN model over the RBF one, is that σ (known as spread) is the only parameter to estimate.

3.1 Learning in GRNN

Now, the issue is how to estimate w_i in Eq. 4, which refers to the learning process of the weights. In backpropagation, we use the steepest descend methods based on gradients to learn the weights while, in RBF, linear algebra is used through the pseudo-inverse and singular value decomposition (SVD) for the same purpose [10, 26, 27, 28]. In GRNN, it is well known in the literature that the weights are obtained straightforward from the outputs of the training data, see Eq. 1 against Eq. 2 [22].

Here, we show from Gaussian mixture models (GMM) how the weights w_i in Eq. 4 are learned from data [29, 30]. Let us see the output y as a probabilistic model:

$$p(y) = \sum_{i=1}^{m} p(y|i)\phi_i \quad (5)$$

which is a linear combination of m models where each ϕ_i is a Gaussian model, and $p(y|i)$ are the mixing coefficients (weights). Therefore, $p(y)$ is a combination of Gaussian models ϕ_i. Assuming the data $\boldsymbol{X}$ is Gaussian $N(\mu_i, \sigma_i^2)$, then $E[p(y|i)] = \mu_i$. Now, Eq. 5 reads:

$$p(y) = \sum_{i=1}^{m} \mu_i \phi_i \quad (6)$$

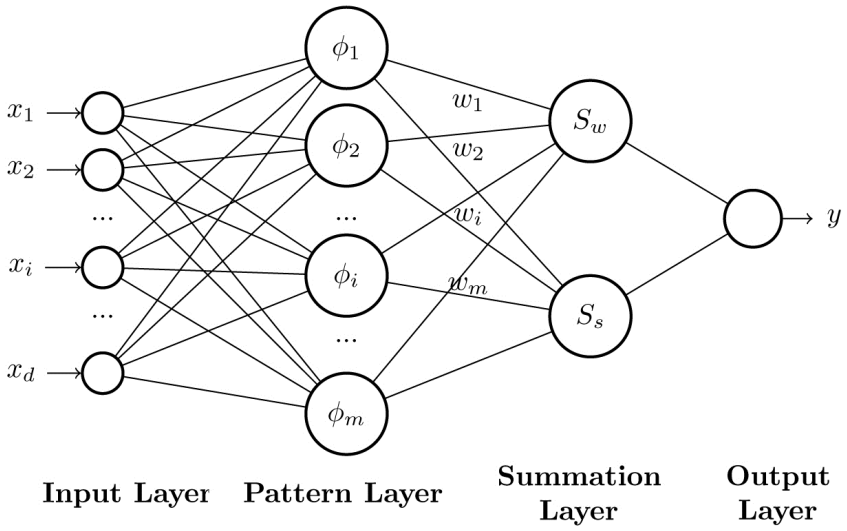


Figure 6. GRNN Architecture.

where μ_i comes from the observed data $(x_i, \hat{y}_i)$, so the weights w_i are equal to μ_i. It is important to recall that, in supervised learning, the observed data comes in pairs$(x_i, \hat{y}_i)$, the input and the observed output.

3.2 Sequential Algorithm for Time Delay Estimation with GRNN

We have two irregularly sampled time series with noise as input (series ***A*** and series ***B***) and the time delays for those series as the output. In this method, series ***A*** and ***B*** are combined to generate a new series called ***C***. This series must be learned by a GRNN. To achieve this, a parameter σ (spread) must be found. We have two different series with potentially different spreads. Using five-fold cross-validation, we find σ for every time series. The average of these two values σ_p is used for the GRNN. We define a range of values $\Delta_p = [\Delta_{min}, \Delta_{max}]$ where the real time delay is found and we test for each Δ_p. In other words, Δ_{min} and Δ_{max} are defined with prior knowledge about the quasar under analysis.

Data: Time T , series A, series B, Δt and M
Result: Combination of series A and B

$\mathbf{T_B} = \mathbf{T} - \Delta t$
$\mathbf{C} = \mathbf{A}$
$\mathbf{T_C} = \mathbf{T}$
For each $T_{B_i} \in \mathbf{T_B}$
  If T_{B_i} $\exists$ $\mathbf{T_C}$
    $C_i = \left(C_i + \left(B_i + M\right)\right) /2$
  else
    Add T_{B_i} **to** $\mathbf{T_C}$ **in order with its respective value B in C**
  end If
end For

Figure 7. Algorithm to combine the series A and B.

The combination of the series is made as follows: we have two vectors, ***A*** and ***B***, representing both time series and the vector ***T*** representing the sampling time for both series. A new vector $\boldsymbol{T_B}$ is generated representing the time of the series ***B***. A delay Δ_t is applied to this vector. Now we generate two new vectors, ***C*** and $\boldsymbol{T_C}$, representing the combined series and its sampling time, respectively. Vectors ***T*** and $\boldsymbol{T_B}$ are orderly included in $\boldsymbol{T_C}$ and the corresponding values for ***A*** and ***B*** into ***C***. If the time value T_i of vector ***T*** is equal to the time value $\boldsymbol{T_{Bi}}$ of vector $\boldsymbol{T_B}$, then we use $C_i = (C_i + (B_i + M))/2$, where ***M*** is the vertical shift between ***A*** and ***B***. This procedure is summarized in Fig. 7.

Once the combined time series has been generated, the neural network is trained with the time series (T_{ci}, C_i) and σ. Then, we obtain the Mean Squared Error (MSE), at training points between ***C*** and the curve generated by GRNN. Afterwards, we register the pair (Δ_p, MSE) in a vector called ***RMSE***. Finally, the best time delay is obtained when the MSE reaches its minimum in ***RMSE***. The pseudo-code of the sequential algorithm is showed in Fig. 8.

We performed a time complexity analysis of the sequential GRNN algorithm. This analysis is based on asymptotic notation [31], especially on the O-notation which is an upper bound. In other words, we are interested on the order of growth of the running time of an algorithm. Because we are looking at the input size (of training data) to n_d the upper bound of the running time (time complexity), we are studying the algorithm efficiency [31].

Therefore, the time complexity of the sequential GRNN algorithm is $O(n^2)$. Because the cross-validation procedure is always quadratic.

4. Parallel Algorithm for Time Delay Estimation with GRNN

In the next two sections, we introduce our parallel algorithms for time delay estimation. The parallelization is carried out using two technologies: MPI and GPU. Therefore, we come up with two versions of a parallel algorithm. These parallel algorithms are based on our sequential algorithm described in §3.2.

Data:T, A, B and M.
Result: The best time delay

Use five-fold cross-validation on A and B to obtain the average spread σ_p
Define a set of trial time delays $\Delta_p = [\Delta_{min}, \Delta_{max}]$
For each $\Delta_{p_i} \in \Delta_p$
  Combine the series A and B with the delay Δ_{p_i} (Figure 7)
  Train the GRNN on the combined curve C by using the spread σ_p estimated as above
  Obtain the mean squared error, MSE, at training points between C and the curve generated by GRNN
  Register the pair (Δ_p,MSE) in $RMSE$
end For
The best time delay is obtained when the MSE reaches its minimum in $RMSE$

Figure 8. Sequential GRNN to estimate the time delay between pairs of time series.

4.1 Performance Analysis of GRNN

Before parallelizing the GRNN sequential algorithm, we performed a study that allows us to calculate the amount of time spent in each routine. So, we used the GNU profiler, i.e. gprof. The results are in Table 2, where % time is the percentage of the total running time; cumulative seconds is a running sum of the number of seconds accounted for by this function and those listed above it; self seconds means the number of seconds accounted for by this function alone; calls is the number of times this function was invoked; and name is the name of the function.

From Table 2, we can see that the time-consuming functions are OutputHidden and CreateGRNN. These functions correspond to the pattern-layer of GRNN (see Fig. 6), and also the parallel part of the GPU-CUDA™ approach below.

4.2 MPI

MPI is a widely used interface to parallelize algorithms, due to its portability [32]. This feature makes it an important option to develop parallel algorithms using clusters and supercomputers as targets.

The first aspect to consider when parallelizing algorithms is using data parallelizing, which means dividing the input data in several parts in order to be concurrently processed [33, 34]. This technique dramatically reduces the processing time. There is a limit in the number of processes to use. In other words, as we increase the number of processes, we expect to reduce the computing time. However, the more processes there are, the more the exchange of information among processes will be. This is known as the latency time [35]. For example, in Fig. 13, when the number of processes reaches the amount of nine, the computational time starts increasing rather than decreasing, because of the latency time. Upon comparing a cluster with a supercomputer, it tourns out that the latency time of a cluster is greater than that of a supercomputer, due to the technology of the communication among nodes – including the bandwidth. Nevertheless, it is a lot cheaper to build a cluster than a supercomputer [36].

Given the portability of MPI, we use this technology to test our parallel algorithms using different architectures.

In Figure 9, we present the MPI algorithm for the time delay estimation. From the sequential algorithm (Fig. 8), we parallelize the loop *For each* $\Delta p_i \in \Delta p$. As we can see in Fig. 9, the master process divides the tasks according to $\Delta p = [\Delta_{min}, \Delta_{max}]$ and the number of computing processes desired. The master process sends each part to its corresponding computing process. Then, the best time delay for each part is estimated with the computing process and is returned to the master process. Finally, the best time delay result, out of all of the parts, is computed by the master process.

% time	cumulative seconds	self seconds	calls	name
74.78	5.62	5.62	367,330	OutputHidden
13.57	6.64	1.02	49,742	CreateGRNN
4.79	7.00	0.36	61’220,544	FindIndex
4.79	7.36	0.36	99,384	CreateCVSets
2.13	7.52	0.16	50	CombineSeries

Table 2. Results of GNU profiler for GRNN.

4.3 GPU and CUDA™

Although the GPU was originally designed for graphics processing, nowadays it is been used for parallelizing algorithms which are not related to graphics processing [37, 38, 39]. This is due to the fact that the GPU has several cores help to speedup floating point operations.

GPU programming can use different technologies including NVIDIA® CUDA™ [40], GLSL[2] and OpenCL[3]. For these technologies, a kernel definition is required for GPU programming. When a kernel is called, it is executed *M* times in parallel by *M* different threads, as opposed to only once in sequential programming. This is illustrated in Fig. 10.

Contrary to our parallel GRNN algorithm with MPI, in our parallel GRNN algorithm with CUDA™, we use our sequential GRNN algorithm as reference. Then, we only parallelize the pattern-layer of GRNN (see Fig. 6). If we observe Fig. 8, the parallelized part corresponds to the computing of the curve estimated by the GRNN, which was also used to obtain the mean squared error.

Data: Time T, series A, series B, Δt and M.
Result: The best time delay

CP = current process
N = Number of parallel process
If CP = Master process
  Use five-fold cross-validation on A and B to obtain the average spread σ_p
  Define a set of trial time delays $\Delta_p = [\Delta_{min}, \Delta_{max}]$
  Divide Δ_p in $N-1$ parts
  For $i = 1$ to N
    Send N_i part of Δ_p to process CP_i
  end For
  For $i = 1$ to N
    Receive and save the best time delay of process CP_i in $RMSE$
  end For
  The best time delay is when the MSE reaches its minimum in $RMSE$
  The function return the best time delay
else
  Receive Δ_p
  For each Δ_{p_i} in Δ_p
    Combine the series A and B with the delay Δ_{p_i} (Fig. 7)
    Train the GRNN on the combined curve C by using the spread σ_p,
      estimated as above
    Obtain the mean squared error, MSE, at training points between C
      and the curve generated by GRNN
    Register the pair (Δ_p,MSE) in $RMSE$
  end For
  The best time delay is when the MSE reaches its minimum in $RMSE$
  The function send the best time delay to master process
end If

Figure 9. Parallel GRNN with MPI. This algorithm estimates the time delay between pairs of time series.

[2] http://www.opengl.org/documentation/glsl/
[3] http://www.khronos.org/opencl/

Data: $\vec{A}$, $\vec{B}$
Result: $\vec{C}$

$i = thread$
$C_i = A_i + B_i$

Figure 10. Kernel example. Two vectors A and B of size N are added and the result is stored as vector C, where i is the identifier of a thread in M threads per core.

Data: input x **, centers C, spread S, weights W**
Result: S_s , S_w

$i = thread$

$$S_{w_i} = W_i exp\left(\frac{-\|x - C_i\|^2}{2S_i^2}\right)$$

$$S_{s_i} = exp\left(\frac{-\|x - C_i\|^2}{2S_i^2}\right)$$

Figure 11. Parallel GRNN. GPU kernel to compute the output of a pattern-layer.

In Fig. 11, we present the kernel for our CUDA-based algorithm for time delay estimation. This kernel computes a single output of a pattern layer, i.e. S_{wi} and S_{si} in Eq. 4. Our parallel GRNN algorithm with CUDA™ is the combination of our sequential algorithm and the algorithm shown in Fig. 12.

Data: $sizeInputs$, **simInputs,** $grnn$
Result: GRNN output

size=grnn.numberHiddenNeurons
Reserve memory for $centerd$, $spredd$, $weightd$, ssd **and** swd **in the graphics device**
Copy $grnn.centers$ **to** $centerd$, $grnn.spreads$ **to** $spreadd$ **and** $grnn.weights$ **to** $weightd$ **in the graphics device**
For $i = 0$ **to** $sizeInputs$
  [ssd,swd] = Call Kernel with parameters $simInputs_i$, $centersd$, $spreadd$ **and** $weightd$ **(see Fig. 11)**
  Copy ssd **to** ss **and** swd **to** sw **from the graphics device**
  For $j = 0$ **to** $size$
    $vss+ = ss_i$
    $vsw+ = sw_i$
  end For
  $output_j = vsw/vss$
end For

Figure 12. Parallel GRNN with GPU. Function to simulate a GRNN with input x.

5. Results and Discussion

First, we tested the sequential algorithm to estimate time delays on the artificial data, dataset DS-5, described in §2.2. In order to test the accuracy of our algorithm, we compared these results using several methods: Linear Interpolation (LI) [14], two versions of the Dispersion spectra method D_1^2 and $D_{4,2}^2$ [12], PRH method [13, 17], K-V a Kernel-based method [20] and EA-M-CV method (evolutionary algorithm) [15]. These are state-of-the-art methods used across the literature, and for each of these methods, trial time delays in the range of $\Delta_{min}= 0.1$ to $\Delta_{max} = 10$ were used with increments of 0.1, since the true delay is known a priori, i.e., to be 5 days [20].

Table 3 contains the results obtained from all of the methods, including our sequential GRNN algorithm. The statistic values of MSE (Mean Squared Error), AE (Absolute Error, average), $\hat{\mu}$ (mean) and $\hat{\sigma}$ (standard deviation) were obtained over the 38,505 datasets described in §2.2. The best results are highlighted in bold text. The sequential GRNN method shows the best results when taking into account the MSE, AE and $\hat{\sigma}$ statistic values, while the D_1^2 method shows the best results when considering the $\hat{\mu}$ statistic value.

MSE and AE measure the accuracy of estimates, that is the error between the estimated delay $\{\Delta_i;\ i = 1, 2, \ldots, 38\,505\}$ and the true delay $\Delta_0= 5$. The $\hat{\mu}$ measures the bias and $\hat{\sigma}$ the dispersion or variance.

Secondly, we tested our algorithms on real data (see §2.1). We use $\Delta_{min}= 400$ and $\Delta_{max}= 449$. Two datasets were used: Kundic and Schild data. Kundic data is the most accepted data across the literature [19]. Our algorithms on these data suggest a time delay of 420 days, which are consistent with previous estimations of time delays on Q0957+561 [14]. It is worthwhile mentioning that the time delay for a given quasar must be the same regardless of the dataset used and the time delay estimation method [5].

We also test our parallel GRNN algorithm on Schild data. We use this large dataset because it has more than one thousand samples. New projects such as LSST, SNAP, SDSS and SLACS (see §1) will generate large datasets in the future. Our motivation for this paper is based precisely on the fact that fast and accurate algorithms will be required to deal with such datasets. Furthermore, it is important to emphasize that some current time delay estimation methods cannot manage large datasets [15].

With Schild data, our aim was to test the speedup of our algorithms. Our sequential GRNN algorithm lasts one second on Kundic data (97 samples) and 108 seconds on Schild data (1232 samples). The best time delay from Schild data is 428 days. In Table 4, we show the results of our parallel algorithm, MPI and CUDA™, on Schild data only. Let us describe Table 4, the first row shows the computational time from a sequential algorithm and the specifications of the computer where the algorithm was running. The second row shows the

Statistic	L1	D_1^2	$D_{4,2}^2$	PRH	K-V	EA-M-CV	GRNN
MSE	0.49	0.74	0.99	13.46	0.47	0.63	**0.30**
AE	0.39	0.52	0.59	3.01	0.39	0.41	**0.31**
$\hat{\mu}$	5.068	**5.013**	5.589	2.704	4.946	5.015	5.034
$\hat{\sigma}$	0.70	0.86	0.80	2.86	0.68	0.79	**0.54**

Table 3. Comparison among different methods, including GRNN

Parallel GRNN MPI Cluster (1p) and the computational time from a cluster with a single process. This cluster is a beouwolf-type cluster, and it has six nodes including the master node, each one with a single processor. The third row presents the results from the cluster with six processes. The forth and fifth rows show the results from the Parallel GRNN with MPI running on a Cray XD1 with one or eight processes. The sixth row refers, once again, to the sequential GRNN results obtained from a different computer, which has the GPU graphics card. The last two row shows the results from our parallel algorithm with CUDA™ using two different graphics cards.

From Table 4, we can observe that the best results were obtained from our parallel GRNN method with MPI running on the Cray XD1, since it was the fastest one. Therefore, in Figure 13, we compare the results from the number of processes against the computing time of the Cray on Schild data. Our best results occur when we use eight processes.

Although we compared two versions of our parallel GRNN method, the parallelization between MPI and CUDA™ turned out to be different. Nevertheless, we achieved to optimize both technologies in order for them to run as good as possible. It was determined that the GPU memory management restricts the parallelization procedure, if the CUDA™ version runs as MPI works.

In fact, we performed a profiling analysis for the CUDA™ version similar to the sequential algorithm in §4.1. We use the Compute Visual Profiler for NVIDIA® CUDA™ technology, the results are in Table 5. We found that the time-consuming methods in CUDA™ are *memcpyHtoD* and *memcpyDtoH*, which correspond in Figure 12 to *Copy ... in the graphics device and Copy ... from the graphics device*, respectively.

Technology	Time	Specifications
Sequential GRNN	108.00 seg	Intel Core2 2.40GHz, 2GB
Parallel GRNN, MPI Cluster (1p)	256.80 seg	Intel PIV 2.80GHz, 512MB
Parallel GRNN, MPI Cluster (6p)	81.13 seg	Intel PIV 2.80GHz, 512MB
Parallel GRNN, MPI Cray XD1 (1p)	45.54 seg	AMD Dual Core 2.2GHz
Parallel GRNN, MPI Cray XD1 (8p)	12.32 seg	AMD Dual Core 2.2GHz
Sequential GRNN	158.00 seg	Intel Core2 1.86GHz, 2GB
Parallel GRNN, CUDA™ CPU+GPU	87.00 seg	Intel Core2 1.86GHz, 2GB GeForce 8800 GTX (Graphics card) Number of multiprocessors: 16 Number of cores: 128
Parallel GRNN, CUDA™ CPU+GPU	20.00 seg	AMD Phenom II 3 Ghz , 2GB GeForce GTX 470 (Graphics card) Number of multiprocessors:14 Number of cores: 448

Table 4. Sequential GRNN versus Parallel GRNN (MPI and CUDA™).

Method	#Calls	GPU time (μs)	% GPU time
memcpyHtoD	147,750	175,277	0.65
memcpyDtoH	729,904	1.9e+06	7.17

Table 5. Results of Compute Visual Profiler for NVIDIA® CUDA™.

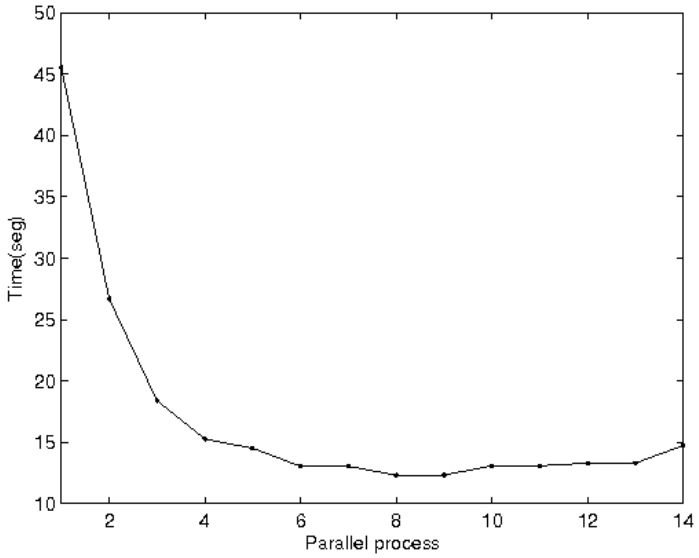


Figure 13. Number of processes versus computing time. This plot was obtained with Schild Data and Parallel GRNN with MPI running on a supercomputer Cray XD1.

We tested a MPI version similar to the CUDA™ version, i.e. parallelizing the pattern-layer of GRNN (see Fig. 6), but the MPI algorithm in Figure 9 give us the best performance.

6. Conclusions

We introduced a new approach for time delay estimation based on GRNN. According to the best of our knowledge, this is the first methodology for time delay by using artificial neural networks. In Table 3, we show that our sequential algorithm based on GRNN (Fig. 8) is competitive with state-of-the-art methods in terms of accuracy. Since the speedup on large datasets is important for new monitoring projects, i.e. LSST, SNAP, SDSS and SLACS, besides the new sequential algorithm based on GRNN, we developed two parallel versions of our sequential algorithm. The first version of our parallel algorithms is based on MPI (Fig. 9), running in a cluster and a supercomputer. The second parallel version is based on CUDA™ for GPU (Fig. 12). Therefore, we also presented the computing time from our sequential and parallel GRNN algorithms in Table 4. Finally, we conclude that the parallel GRNN with MPI running on a Cray supercomputer with eight processes provides a superior performance, because the Cray architecture, i.e. communication among nodes, allows fast data transfer. As we see in Table 5, the bootleneck of GPU are the functions to manage the GPU memory for data transfer

7. Future work

Part of the future work is to explore new CUDA™ capabilities, so we can take advantage of the GPU technology. The new survey projects are starting to generate new large datasets and some of these projects are still in development. As part of the future work, we would like to test the performance of our parallel approaches on real large datasets, because these datasets may differ to the Schild data used in our experiments. We only parallelize the GRNN method, so the parallelization of other methods has not been done. More work on this direction is also part of our future work.

Acknowledgements

We thank the reviewers who have helped to improve the paper. We would like to thank the CNS-IPICyT for allowing us to use the Cray XD1 to run our algorithms. We also thank Somak Raychaudhury for providing us the Schild real data, private communication, and introducing us to this challenging research area. Last but not least, we thank M. Sofía Muñoz-Mejía for proofreading this paper.

References

[1] Saha, P.: Gravitational Lensing. Encyclopedia of Astronomy and Astrophysics (2000)

[2] Sanguinetti, G., Lawrence, N.: Missing data in kernel pca. In: Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence (LNAI 4212), pp. 751–758. Springer-Verlag (2006)

[3] Bridewell, W., Langley, P., Racunas, S., Borrett, S.: Learning process models with missing data. In: Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence (LNAI 4212), pp. 557–565. Springer-Verlag (2006)

[4] Kundic, T., Turner, E., Colley, W., Gott-III, J., Rhoads, J., Wang, Y., Bergeron, L., Gloria, K., Long, D., Malhorta, S., Wambsganss, J.: A robust determination of the time delay in 0957+561A,B and a measurement of the global value of Hubble's constant. Astrophysical Journal 482(1), 75–82 (1997)

[5] Refsdal, S.: On the possibility of determining Hubble's parameter and the masses of galaxies from the gravitational lens effect. Monthly Notices of The Royal Astronomical Society 128, 307–+ (1964)

[6] Dobler, G., Keeton, C.R.: Microlensing of Lensed Supernovae. Astrophysical Journal 653, 1391–1399 (2006). DOI 10.1086/508769

[7] Paraficz, D., Hjorth, J., Burud, I., Jakobsson, P., Elíasdóttir, Á.: Microlensing variability in time-delay quasars. Astronomy and Astrophysics 455, L1–L4 (2006). DOI 10.1051/0004-6361:20065502

[8] Cuevas-Tello, J., Tiňo, P., Raychaudhury, S.: How accurate are the time delay estimates in gravitational lensing? Astronomy and Astrophysics 454, 695–706 (2006)

[9] Howlett, R., Jain, L. (eds.): Radial Basis Function Networks 2: new advances in design. Physica-Verlag (2001)

[10] Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall (1999)

[11] Gonzalez-Grimaldo, R., Cuevas-Tello, J.: Analysis of time series with neural networks. In: Proceedings of 7th Mexican International Conference on Artificial Intelligence (MICAI), pp. 131–137. IEEE Computer Society (2008)

[12] Pelt,J.,Hjorth,J.,Refsdal,S.,Schild,R.,Stabell,R.:Estimatio nofmultipletimedelays in complex gravitational lens systems. Astronomy and Astrophysics 337(3), 681–684 (1998)

[13] Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes in C++: The Art of Scientific Computing, second edn. Cambridge University Press (2002)

[14] Cuevas-Tello, J.: Estimating time delays between irregularly sampled time series. Ph.D. thesis, School of Computer Science, University of Birmingham (2007). http://etheses.bham.ac.uk/88/

[15] Cuevas-Tello, J., Tiňo, P., Raychaudhury, S., Yao, X., Harva, M.: Uncovering delayed patterns in noisy and irregularly sampled time series: an astronomy application. Pattern Recognition 3(43), 1165–1179 (2010)

[16] Harva, M., Raychaudhury, S.: Bayesian estimation of time delay between unevenly sam- plead signals. In: IEEE International Workshop on Machine Learning for Signal Pro- cessing, pp. 111–122. IEEE (2006)

[17] Press, W., Rybicki, G., Hewitt, J.: The time delay of gravitational lens 0957+561, I. Methodology and analysis of optical photometric data. Astrophysical Journal 385(1), 404–415 (1992)

[18] Schild, R., Thomson, D.: The Q0957+561 time delay from optical data. Astronomical Journal 113(1), 130–135 (1997)

[19] Oguri, M.: Gravitational Lens Time Delays: A Statistical Assessment of Lens Model Dependences and Implications for the Global Hubble Constant. The Astrophysical Journal 660, 1–15 (2007). DOI 10.1086/513093

[20] Cuevas-Tello, J., Tiňo, P., Raychaudhury, S.: A kernel-based approach to estimating phase shifts between irregularly sampled time series: An application to gravitational lenses. In: Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence (LNAI 4212), pp. 614–621. Springer-Verlag (2006)

[21] Harva, M., Raychaudhury, S.: Bayesian estimation of time delays between unevenly sampled signals. Neurocomputing 72(1-3), 32–38 (2008)

[22] Specht, D.: A general regression neural network. IEEE Transactions on Neural Networks, 2(6), 568–576 (1991)

[23] Cristianini, N.,Shawe-Taylor, J.:Support Vector Machines and other kernel-based learning methods. Cambridge University Press (2000)

[24] Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)

[25] Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2001)

[26] Golub, G., Loan, C.V.: Matrix Computations, second ed. The Johns Hopkins University Press (1989)

[27] Cowan, G.: Statistical Data Analysis. Oxford University Press (1998)

[28] Courrieu, P.: Fast computation of moore-penrose inverse matrices. Neural Information Processing – Letters and Reviews 8(2), 25–29 (2005)

[29] Nabney, I.: NETLAB: Algorithms for Pattern Recognition. Advances in Pattern Recognition. Springer (2002)

[30] Brown, K.: Diversity in neural networks ensembles. Ph.D. thesis, School of Computer Science, University of Birmingham, UK (2004)

[31] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, second ed. McGraw-Hill (2002)

[32] Gropp, W.: Learning from the success of mpi. In: HiPC '01: Proceedings of the 8th International Conference on High Performance Computing, pp. 81–94. Springer-Verlag, London, UK (2001)

[33] Hillis, W.D., Steele Jr., G.L.: Data parallel algorithms. Commun. ACM 29(12), 1170–1183 (1986). DOI http://doi.acm.org/10.1145/7902.7903

[34] Cantu-Paz, E., Goldberg, D.: Efficient parallel genetic algorithms: theory and practice. Computer Methods in Applied Mechanics and Engineering 186(1), 221–238 (2000)

[35] Amdahl, G.: Readings in computer architecture, chap. Validity of the single processor approach to achieving large scale computing capabilities, pp. 79–81. Morgan Kaufmann Publishers Inc. (2000)

[36] Luo, X.Q., Gregory, E.B., Yang, J.C., Wang, Y.L., Chang, D., Lin, Y.: Parallel com- puting on a pc cluster. CoRR cs.DC/0109004 (2001)

[37] K.S.S., Jung, K.: Gpu implementation of neural networks. Pattern Recognition 37(6), 1311 – 1314 (2004). DOI DOI: 10.1016/j.patcog.2004.01.013

[38] Luo, Z., Liu, H., Wu, X.: Artificial neural network computation on graphic process unit. In: Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on, vol. 1, pp. 622–626 vol. 1 (2005). DOI 10.1109/IJCNN.2005.1555903. URL http://dx.doi.org/10.1109/IJCNN.2005.1555903

[39] Brandstetter, A., Artusi, A.: Radial basis function networks gpu based implementation. IEEE Transaction on Neural Network 19(12), 2150–2161 (2008)

[40] NVIDIA: NVIDIA CUDA Programming Guide. NVIDIA (2009). http://www.nvidia.com