

# Scalable Hyper-parameter Estimation for Gaussian Process Based Time Series Analysis

Varun Chandola  
Oak Ridge National Labs  
1 Bethel Valley Road  
Oak Ridge, TN  
chandolav@ornl.gov

Ranga Raju Vatsavai  
Oak Ridge National Labs  
1 Bethel Valley Road  
Oak Ridge, TN  
rrvatsavai@ornl.gov

## ABSTRACT

Gaussian process (GP) is increasingly becoming popular as a kernel machine learning tool for non-parametric data analysis. Recently, GP has been applied to model non-linear dependencies in time series data. GP based analysis can be used to solve problems of time series prediction, forecasting, missing data imputation, change point detection, anomaly detection, etc. But the use of GP to handle massive scientific time series data sets has been limited, owing to its expensive computational complexity. The primary bottleneck is the handling of the covariance matrix whose size is quadratic in the length of the input time series. In this paper we propose a scalable method that exploits the special structure of the covariance matrix for hyper-parameter estimation in GP based learning. The proposed method allows estimation of hyper parameters associated with GP in quadratic time, which is an order of magnitude improvement over standard methods with cubic complexity. Moreover, the proposed method does not require explicit computation of the covariance matrix and hence has memory requirement linear to the length of the time series as opposed to the quadratic memory requirement of standard methods. To further improve the computational complexity of the proposed method, we provide a parallel version to concurrently estimate the log likelihood for a set of time series which is the key step in the hyper-parameter estimation. Performance results on a multi-core system show that our proposed method provides significant speedups as high as 1000, even when running in serial mode, while maintaining a small memory footprint. The parallel version exploits the natural parallelization potential of the serial algorithm and is shown to perform significantly better than the serial faster algorithm, with speedups as high as 10.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—  
*Data Mining*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD-LDMTA'10, July 25, 2010, Washington, DC, USA.  
Copyright 2010 ACM 978-1-4503-0215-9/10/07 ...\$10.00.

## General Terms

Algorithms

## Keywords

Gaussian process, Time Series, Scalability

## 1. INTRODUCTION

Gaussian process (GP) [14, 15]<sup>1</sup> is increasingly becoming popular as kernel machine learning tool for non-parametric regression and classification [10]. If the time indices are used as the inputs, one can use a GP as a forecasting or prediction model for time series data [1, 8, 2]. Besides prediction, GP based models can also be used for other time series analysis tasks such as change detection, anomaly detection, missing data imputation, noise removal, etc.

While GP has emerged as a popular learning method, its application to large scale data sets has been limited owing to the inherent  $O(t^3)$  computational complexity as well as  $O(t^2)$  memory storage requirements, where  $t$  is the input data size. When dealing with time series,  $t$  is the length of the time series, which can be large (and growing) for applications such as remote sensing, astronomy, electro-cardiograph (ECG) analysis, etc. For example, remote sensing satellites are continuously collecting data at global level at high temporal resolution. For each spatial location, the length of the time series is continuously growing and hence standard GP analysis methods cannot scale to such long time series.

The computational bottleneck for GP based analysis is further compounded by the fact that often one needs to simultaneously handle multiple time series, for example, multiple remote sensing time series collected from a spatial neighborhood. In the case of remote sensing, the number of time series in a spatial neighborhood are ever increasing with the spatial resolution of the satellite instruments. The standard GP analysis methods have a  $O(pt^3)$  computational complexity and a  $O(t^2 + p)$  memory footprint for handling  $p$  time series simultaneously. While multi-threaded or parallel programming can alleviate the issue of handling  $p$  time series simultaneously, the quadratic memory requirements are a significant bottleneck, especially in emerging heterogeneous computing architectures, hybrid of multi-core and Graphical Processing Units (GPUs), in which movement of data is expected to be the biggest computational bottleneck.

In this paper we exploit the special structure of the covariance matrix associated with the GP analysis to make the algorithms scale to massive data sizes. We make use of several

<sup>1</sup>Henceforth, referred to as GP.

existing results derived for matrices in mathematics literature [9, 17] and extend these to adapt them for GP analysis for time series. We present an algorithm to compute the exact Marginal Log Likelihood (MLL) and the derivative of the MLL, which are the key steps in estimating the GP hyper-parameters via gradient based optimization. The computational complexity of the proposed algorithm is  $O(t^2)$  and requires  $O(t)$  memory. We also present a parallel version of the algorithm to simultaneously process multiple time series to compute the MLL and its derivatives. We present results on artificial data to demonstrate the speedups achieved the proposed algorithm running in serial as well as in parallel (multi-threaded) mode on a multi-core system.

## 2. RELATED WORK

As noted earlier, GP based methods typically scale as  $O(t^3)$  with the size of the input data with a memory requirement of  $O(t^2)$ . This makes them impractical in domains that encounter massive data sizes such as remote sensing, ECG analysis, etc. Several approximation based methods have been proposed in the literature [19, 5, 7] to scale GP to such large datasets (See [15, Chapter 8] for a detailed overview). These methods fall under the general purview of sparse and approximate kernel methods. All of these methods use matrix approximation techniques to efficiently manipulate the covariance matrix (inverse computation, Cholesky factorization, solving system of equations). Several papers [19, 5] approximate the covariance matrix using lower rank approximation techniques, such as the Nyström approximation, for faster but approximate results. Several papers have used a “subset of regressors” approach [16, 7] that uses only  $m$  out of  $t$  regressors and hence entail  $O(m^2t)$  complexity. In this paper we focus on scaling the GP analysis such that we obtain the exact solution and hence we do not compare our approach to the existing approximate methods.

While scalability has been a key issue for data mining applications, only a few existing techniques make use of the available concurrency from high performance computing hardware and software to address this issue in the context of GP analysis. Keane et al [12] proposed a data parallel approach for likelihood estimation in GP regression, but their method estimate the log likelihood locally, and hence the final outcome is not guaranteed to be the same as a sequential algorithm.

In this paper we make use of the fact that the covariance matrix encountered with GP for time series is a symmetric Toeplitz matrix and hence several solutions that have been proposed in literature [9, 11] can be utilized to make the hyper-parameter estimation algorithm computationally as well as memory efficient. Specifically, there have been many  $O(t^2)$  algorithms developed to invert a Toeplitz matrix as well as solve a Toeplitz system of equations [17, 13, 6]. We adapt the algorithms by Trench [17, 20, 21] for the problem for scalable hyper-parameter estimation as its stability has been well studied in literature for large Toeplitz matrices [3].

## 3. BACKGROUND

In this section we will provide a background on Gaussian process based learning and its application to time series modeling. We will then provide the algorithm for computing the marginal log likelihood and its derivative with respect

to the associated hyper-parameters for a given set of observations.

### 3.1 Gaussian Process

A GP is a generalization of a Gaussian distribution and is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution [15, Chapter 2]. A GP describes a distribution over a (potentially infinite) set of functions and is completely specified by its mean function  $m(\mathbf{x})$  and covariance function  $k(\mathbf{x}, \mathbf{x}')^2$ :

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \quad (1a)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \quad (1b)$$

where  $\mathbf{x}$  is an input or index belonging to an input or index set  $\mathcal{X}$ . Typically the mean function is taken to be zero and the GP is written as:

$$f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}')) \quad (2)$$

Thus the above GP is a collection of random variables, where each random variable is the value of function  $f(\mathbf{x})$  at location  $\mathbf{x}$ . When dealing with time series, the index set  $\mathcal{X}$  is the set of time indices  $\{1, 2, \dots, T\}$ , though a GP can be defined for more general forms of inputs such as  $\mathbb{R}^D$ . For this paper, since we are dealing with time series, we will replace  $\mathbf{x}$  with  $t$  to denote time. The covariance function  $k$  defines the covariance between the function values at two different time points:

$$\text{cov}(f(t_1), f(t_2)) = k(t_1, t_2) \quad (3)$$

Typically, a covariance function is specified using a set of parameters  $\Theta$ , these are considered as the *hyper-parameters* for the GP. For example, a widely used covariance function, called *squared exponential (se)*, can be written as:

$$k(t_1, t_2) = \sigma_f^2 \exp\left(-\frac{\Delta t^2}{2l^2}\right) \text{ where } \Delta t = t_1 - t_2 \quad (4)$$

### 3.2 Prediction Using GP

For GP based regression, it is assumed that the actual observations  $y_t$  are noisy versions of the function values  $f(t)$  and the two quantities are related as:

$$y_t = f(t) + \varepsilon_t \quad (5)$$

where  $\varepsilon_t$  is a noise term that accounts for the noisy component of the observations. Traditionally,  $\varepsilon_t$  is assumed to be a Gaussian noise term  $\sim \mathcal{N}(0, \sigma_n^2), \forall t$ , though we propose an observation dependent covariance structure for  $\varepsilon_t$  which is better suited for handling periodic time series data. For now, we will assume that  $\varepsilon_t \sim \mathcal{N}(0, \sigma_n^2), \forall t$ .

Given a noisy set of observations  $\mathbf{y}_{t-1}$ , the GP prior on  $\{f(1), f(2), \dots, f(t)\}$  (see (2)) and the relationship between  $y_t$  and  $f(t)$  (see (5)), can be used to make a prediction at time  $t$ . The advantage of GP is that the prediction is not a value but a normal distribution  $\sim \mathcal{N}(\hat{y}_t, \hat{v}_t)$ , where the predictive mean  $\hat{y}_t$  and predictive variance  $\hat{v}_t$  are given by:

$$\hat{y}_t = K_{tt-1}^\top (K_{(t-1)(t-1)} + \sigma_n^2 I)^{-1} \mathbf{y}_{t-1} \quad (6)$$

$$\hat{v}_t = k(t, t) - K_{tt-1}^\top (K_{(t-1)(t-1)} + \sigma_n^2 I)^{-1} K_{tt-1} \quad (7)$$

where  $K_{(t-1)(t-1)}$  is a  $|\mathbf{t} - \mathbf{1}| \times |\mathbf{t} - \mathbf{1}|$  kernel matrix such that  $K_{(t-1)(t-1)}[i][j] = k(i, j)$ . Similarly,  $K_{tt-1}$  is a  $(t - 1)$  length vector such that  $K_{tt-1}[i] = k(t, j)$ .

<sup>2</sup>In this paper we will denote matrices with capital letters ( $K$ ), vectors with small bold letters ( $\mathbf{x}, \mathbf{s}_i$ ), and scalars with small letters ( $t, x_i$ ).

The marginal log-likelihood of  $\mathbf{y}_t$ , denoted as  $l_t$ , can be calculated as:

$$l_t = \log p(\mathbf{y}|\Theta) = -\frac{1}{2}\mathbf{y}^\top (K_{(t-1)(t-1)} + \sigma_n^2 I)^{-1} \mathbf{y} \quad (8)$$

$$-\frac{1}{2} \log |(K_{(t-1)(t-1)} + \sigma_n^2 I)|$$

$$-\frac{n}{2} \log 2\pi$$

Equations (6) and (7) allow one to use GP for time series prediction as well as other analysis tasks such as outlier/anomaly detection, noise removal, and change detection. But as can be observed in (6), (7), and (8), handling the large covariance matrix,  $(K_{tt} + \sigma_n^2 I)$  (henceforth referred to as  $K$  for simplicity) is the key bottleneck for computing as well as memory resources.

### 3.3 Hyper-parameter Estimation Using Gradient Descent

The hyper-parameters  $\Theta$  associated with the covariance function can be calculated by minimizing the marginal log likelihood ( $l_t$ ) function in (8) using a gradient based optimization algorithm. The derivative of  $l_t$  with respect to a given hyper-parameter  $\theta \in \Theta$  can be computed as ([15, Chapter 5]):

$$\frac{\partial l_t}{\partial \theta} = -\frac{1}{2}\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(K^{-1} \frac{\partial K}{\partial \theta}) \quad (9)$$

The computational complexity of the log-likelihood calculation as well as the derivative of log-likelihood is  $O(t^3)$  where  $t$  is the length of the time series  $\mathbf{y}$  if standard inversion or Cholesky decomposition based methods are used. Moreover, the calculations require keeping the  $O(t^2)$  matrix in the memory.

## 4. EFFICIENT HYPER-PARAMETER ESTIMATION

In this section we will present a scalable algorithm to compute the log-likelihood and its derivative (See (8) and (9)). For this we assume that the covariance function used in the GP is *stationary* and only depends on the absolute difference between the inputs, i.e,  $k(t_1, t_2) = k(|\Delta_t|)$ . Many widely used covariance functions, such as the *squared exponential* function in (4) and *Matern* class of functions fall under this criterion.

### 4.1 Characteristics of Covariance Matrix

We first note that the covariance function discussed above will result in a symmetric *Toeplitz* matrix,  $K$ , as shown below:

$$K = \begin{pmatrix} k_0 & k_1 & k_2 & \dots & k_{t-1} \\ k_1 & k_0 & k_1 & \dots & k_{t-2} \\ k_2 & k_1 & k_0 & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{t-1} & k_{t-2} & \dots & \dots & k_0 \end{pmatrix} \quad (10)$$

Moreover it can be shown that such functions result in a positive semi-definite covariance matrix while adding a  $\sigma_n^2$  noise to the diagonal results in a positive definite covariance matrix. One can straightaway note that  $K$  in (10) can be represented using just the first row (or column) of  $K$ .

---

### Algorithm 1 ToeplitzInverseSolve( $\mathbf{k}, \mathbf{y}$ )

---

```

if  $k_1 \neq 1$  then
   $\mathbf{k} \leftarrow \mathbf{k}/k_1, \mathbf{y} \leftarrow \mathbf{y}/k_1$ 
end if
 $\mathbf{z}_1 \leftarrow y_1, \mathbf{g}_1 \leftarrow -k_2, \lambda_1 \leftarrow 1 - k_2^2$ 
for  $i = 1$  to  $t - 2$  do
   $\theta_i \leftarrow y_{i+1} - \mathbf{z}_i^\top \mathbf{k}_{2:i+1}$ 
   $\gamma_i \leftarrow -k_{i+2} - \mathbf{g}_i^\top \mathbf{k}_{2:i+1}$ 
   $\mathbf{z}_{i+1} \rightarrow \begin{bmatrix} \mathbf{z}_i + \frac{\theta_i}{\lambda_i} \hat{\mathbf{g}}_i \\ \frac{\theta_i}{\lambda_i} \hat{\mathbf{g}}_i \end{bmatrix}$ 
   $\mathbf{g}_{i+1} \rightarrow \begin{bmatrix} \mathbf{g}_i + \frac{\gamma_i}{\lambda_i} \hat{\mathbf{g}}_i \\ \frac{\gamma_i}{\lambda_i} \hat{\mathbf{g}}_i \end{bmatrix}$ 
   $\lambda_{i+1} \rightarrow \lambda_i - \frac{\gamma_i^2}{\lambda_i}$ 
end for
 $\theta_{t-1} \rightarrow y_t - \mathbf{z}_{t-1}^\top \mathbf{k}_{2:t}$ 
 $\mathbf{z}_t \rightarrow \begin{bmatrix} \mathbf{z}_{t-1} + \frac{\theta_{t-1}}{\lambda_{t-1}} \hat{\mathbf{g}}_{t-1} \\ \frac{\theta_{t-1}}{\lambda_{t-1}} \hat{\mathbf{g}}_{t-1} \end{bmatrix}$ 
return  $\mathbf{z}_t$ 

```

---

This characteristic straightaway provides a way to reduce the memory requirements of the algorithms involving  $K$ .

### 4.2 Using Toeplitz Matrix Operations

Several  $O(t^2)$  algorithms have been proposed for Toeplitz matrix inversion which make use of the special matrix structure to compute the inverse [17, 13, 6]. But one can observe that a direct inversion of the covariance matrix  $K$  is not required to calculate the log-likelihood and its derivatives in (8) and (9). Instead, one only needs to calculate the following quantities:

1.  $\mathbf{y}^\top K^{-1} \mathbf{y}$
2.  $\log |K|$
3.  $\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y}$
4.  $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta})$

One can use Cholesky decomposition and solve a system of equations using the Cholesky decomposition to compute each of these quantities but that will have  $O(t^3)$  complexity to compute the decomposition and  $O(t^2)$  memory requirement for the covariance matrix  $K$ .

We will show that each of these four quantities can be computed in a computational as well as memory efficient manner:

#### 4.2.1 Computing $\mathbf{y}^\top K^{-1} \mathbf{y}$

Algorithm 1 shows how one can compute  $K^{-1} \mathbf{y}$ , i.e., solving a Toeplitz system of equations. This algorithm was originally proposed by Trench [18, 21] for Toeplitz matrices and we simplify it for the symmetric case. This algorithm takes the first row of the covariance matrix,  $\mathbf{k} = \{k_1, k_2, \dots, k_t\}$ , as input and returns the solution vector. In the algorithm  $\hat{\mathbf{x}}$  denotes a vector obtained by reversing the vector  $\mathbf{x}$ . A portion of a vector is denoted as  $\mathbf{x}_{i:j}$ . Note that this algorithm is  $O(t^2)$  and requires  $O(t)$  memory only.

#### 4.2.2 Computing $\log |K|$

It has been shown that the determinant of the matrix  $K$  can be computed as a by-product of the Algorithm 1 by

---

**Algorithm 2** ToeplitzDiagonalSums( $\mathbf{k}$ )

---

```

alpha  $\leftarrow$  ToeplitzInverseSolve( $\mathbf{k}_{1:t-1}, \mathbf{k}_{2:t}$ )
 $\gamma \leftarrow \frac{1}{k_1 + k_{2:t} \alpha}$ 
nu  $\leftarrow [\gamma \hat{\alpha} \gamma]^\top$ 
for  $k = 0$  to  $t - 1$  do
   $s_k \leftarrow \frac{1}{\gamma} \sum_{j=1}^{t-k} (2i + k - n + 1) \nu_i \nu_{i+k}$  { $s_k$  is the sum of
  the  $k^{\text{th}}$  diagonal starting from main diagonal ( $k = 0$ ).}
end for
return  $\mathbf{s}$ 

```

---

simply taking the product of  $\lambda_i$ s [20], i.e.,

$$\log |K| = t \log k_1 + \sum_{i=1}^{t-1} \log \lambda_i \quad (11)$$

Thus  $\log |K|$  can be computed in linear time without requiring any additional memory.

### 4.2.3 Computing $\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y}$

Algorithm 1 computes  $K^{-1} \mathbf{y}$ . The vector  $K^{-1} \mathbf{y}$  can be multiplied with  $\frac{\partial K}{\partial \theta}$  in  $O(t^2)$  time and the resulting vector can be multiplied with  $\mathbf{y}^\top$  in linear time. Note that since  $\frac{\partial K}{\partial \theta}$  is Toeplitz, it can be multiplied using only one representative row of the matrix, i.e., with  $O(t)$  space requirements.

### 4.2.4 Computing $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta})$

Let  $L = K^{-1}$  and  $P = \frac{\partial K}{\partial \theta}$ . We are interested in computing  $\text{tr}(LP) = \text{tr}(PL)$  where  $P$  is a symmetric Toeplitz matrix and  $L$  is the inverse of a symmetric Toeplitz matrix. We can write:

$$\begin{aligned} \text{tr}(PL) &= \sum_{i=1}^t \sum_{j=1}^t p_{ij} l_{ij} \\ &= \sum_{i=1}^t \sum_{j=1}^t p_{|i-j|} l_{ij} \\ &= \sum_{k=-t+1}^{t-1} p_{|k|} \sum_{j=k+1}^t l_{j-k,j} \\ &= p_0 \sum_{j=1}^t l_{jj} + 2 \sum_{k=1}^{t-1} p_k \sum_{j=k+1}^t l_{j-k,j} \end{aligned}$$

Note that each summation  $\sum_{j=k+1}^t l_{j-k,j}, \forall k = 0 \dots t-1$  is nothing but the sum of the  $k^{\text{th}}$  diagonal of  $L$ . Given the diagonal sums for  $L (= K^{-1})$  we can compute  $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta})$  in linear time. An  $O(t^2)$  algorithm for the computation of the diagonal sums is shown in Algorithm 2. The proof of correctness of the algorithm was given by Dias and Leitao [4].

The computational complexity involved with computing  $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta})$  using Algorithm 2 is  $O(t^2)$  with  $O(t)$  memory required.

## 5. HANDLING MULTIPLE TIME SERIES

In many scenarios one needs to estimate the GP hyper-parameters with respect to multiple time series. Let  $\mathbf{Y} = [\mathbf{y}_1 \mathbf{y}_2 \dots \mathbf{y}_p]$  be a set of input time series. The total marginal log likelihood for all time series will be equal to the sum of marginal log likelihoods for individual time series using (8),

---

**Algorithm 3** GPFast( $\mathbf{k}, \frac{\partial \mathbf{k}}{\partial \theta}, \mathbf{y}$ )

---

```

 $K^{-1} \mathbf{y}, \log |K| \leftarrow$  ToeplitzInverseSolve( $\mathbf{k}, \mathbf{y}$ )
Compute  $\mathbf{y}^\top K^{-1} \mathbf{y}$  and  $\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y}$  using above.
 $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta}) < -$  ToeplitzDiagonalSums( $\mathbf{k}$ )
return Compute  $l_t$  and  $\frac{\partial l_t}{\partial \theta}$  using (8) and (9).

```

---

i.e.,

$$\log p(\mathbf{Y}|\Theta) = \sum_{i=1}^p \log p(\mathbf{y}_i|\Theta) \quad (12)$$

Same holds for the computation of the derivative of the total marginal log likelihood with respect to a hyper-parameter. One can compute these two quantities in a loop using the results in Section 4 resulting in a  $O(pt^2)$  complexity. In order to improve the complexity in the multiple time series scenario we present a parallel version of the algorithm.

The final algorithm for computing the marginal log-likelihood and its derivatives for a given time series uses the four quantities computed above and combines them as shown in (8) and (9). The steps are sketched in Algorithm 3. The algorithm takes the first row of the covariance matrix and its derivative as inputs. The algorithm assumes that there is only one hyper-parameter though it can be directly extended for multiple hyper-parameters.

## 5.1 Parallel Version of Hyper-parameter Estimation

For the parallel version, a straightforward approach would be to send the task of computing the log-likelihood and its derivatives for each input time series, *GPFast*, to a different processing unit. Thus in a system with  $O(p)$  concurrency, one can achieve a  $O(t^2)$  complexity. We further reduce the cost of the algorithm by noting that two of the four key quantities required for the computation of the log-likelihood and its derivatives are independent of the input time series, and only quantities that depend on  $\mathbf{Y}$  are  $\mathbf{y}^\top K^{-1} \mathbf{y}$  and  $\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y}$ .

In the parallel version, we invoke *ToeplitzInverseSolve* for each  $\mathbf{y}_i \in \mathbf{Y}$  and also compute the quantities  $\mathbf{y}^\top K^{-1} \mathbf{y}$ ,  $\mathbf{y}^\top K^{-1} \frac{\partial K}{\partial \theta} K^{-1} \mathbf{y}$ , and also  $\log |K|$  (as by-product of *ToeplitzInverseSolve*) from the resulting solution. The final results are passed back to the master node which then combines these to compute the log-likelihood and its derivatives for the hyper-parameter estimation algorithm. The master node, after delegating tasks to the children nodes, computes the quantity  $\text{tr}(K^{-1} \frac{\partial K}{\partial \theta})$  and then waits for the children nodes to join. The data required by each child node is  $O(t)$ .

For our experiments we implemented a multi-threaded (using JAVA native threads) implementation of the parallel algorithm but the same algorithm can be implemented using MPI for multi-processor architectures or using *MapReduce* for cloud based computing architectures. The linear data size required by each child node is especially attractive for the latter, where the amount of data transferred between nodes can be a significant bottleneck.

## 6. EXPERIMENTAL RESULTS

In this section we compare the computational performance of the proposed algorithm *GPFast* against a standard algorithm (*GPStandard*) for computing log-likelihoods and deriva-

tives. We also investigate the performance of the multi-threaded version of *GPFast*, referred to as *GPFastP*. All experiments are done on time series with varying lengths. All algorithms were implemented in JAVA using the JAMA<sup>3</sup> package. The *GPStandard* algorithm used cholesky decomposition provided with the JAMA package to solve the system of equations and compute the inverse, etc.

All experiments were run on an SGI Altix ICE 8200 cluster called Frost<sup>4</sup>. Frost is currently configured with 128 compute nodes each having 16 virtual cores and 24GB of memory, infiniband interconnects, and a gigabit ethernet network. Each node is capable of supporting 32 threads. The serial algorithms were run with a single thread while the parallel algorithm was run on a single node using multiple threads. We measured the speedup achieved by the improvements in the serial algorithm presented in this paper as well as by using the available concurrency.

### 6.1 Performance of *GPFast* vs. *GPStandard*

We first compare the performance of the computational and memory efficient *GPFast* algorithm against the *GPStandard* algorithm. Table 1 shows the performance of the two algorithms on single time series with varying lengths of the time series. Note that the *GPStandard* algorithm requires a  $O(t^2)$  space in the memory and hence could not run for time series more than 5000 length. Each data point in Table 1 is an average of measurements from 10 runs with same configuration. Results indicate that the *GPFast* algorithm

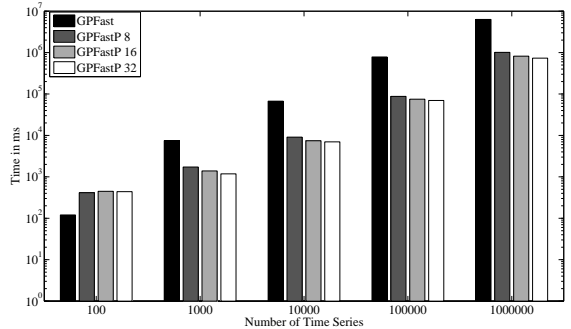
Length	GPFast	GPStandard	SpeedUp
10	3	4	1.33
20	4	7	1.75
50	8	39	4.88
100	16	101	6.31
500	134	1962	14.64
1000	374	25803	68.99
5000	5071	5985433	1180.33
10000	15717	—	—
100000	150696	—	—

**Table 1: Computation times and speedup of *GPFast* vs. *GPStandard* (All times in msec).**

achieves significant speedups over the standard algorithm, with speedup over 1000 for time series of length 5000. The *GPStandard* algorithm runs out of memory for time series longer than 5000 while the *GPFast* algorithm has no memory related issue in dealing with time series as long as 1000000.

### 6.2 Performance of *GPFastP*

To study the performance of the parallel version of *GPFastP* we ran it using varying number of threads. We fixed the length of the time series at 5000 and varied the number of time series used for hyper-parameter estimation. Figure 1 compares the performance of *GPFastP* for different number of threads. The speedups attained by using multiple threads over the serial algorithm (*GPFast*) are summarized in Table 2. For small number of time series, the parallel algorithm does not provide much improvement over the serial algorithm, but the improvement is clear for larger data



**Figure 1: Performance Comparison of *GPFastP*. Y-axis is on logscale.**

# Time Series	$n = 8$	$n = 16$	$n = 32$
100	0.29	0.27	0.27
1000	4.33	5.38	6.33
10000	7.38	9.02	9.59
100000	8.92	10.44	11.21
1000000	6.24	7.72	8.60

**Table 2: Speedups achieved by *GPFastP* over serial *GPFast* for different number of threads ( $n$ ). Length of the time series is fixed at 500.**

sizes. The improvement from serial to using 8 threads is significant for data sizes larger than 1000, but the improvement from using 8 threads to 32 threads is not exactly linear. The maximum speedup of close to 11 is reached when using 32 threads and processing 100K time series.

## 7. CONCLUSIONS

GP based methods typically scale as  $O(t^3)$  with the size of the input data with a memory requirement of  $O(t^2)$ . This makes them impractical in domains that encounter massive time series data sizes such as remote sensing, ECG analysis, etc. In this paper we have shown how Gaussian process analysis can be scaled to handle massive time series data sets. The proposed algorithms allows computation of the marginal log-likelihood of a given time series or a set of time series in  $O(t^2)$  time while using  $O(t)$  memory space. The proposed serial algorithm *GPFast* can process massive data sizes while providing speedups of as high as 1000 over the traditional serial algorithm. We have utilized several existing known results in the area of Matrix algebra to design a fast algorithm that can scale to massive data sizes often encountered in scientific applications.

The parallelization demonstrated using multiple threads indicates that GP analysis is naturally suited for parallelization and hence can be further scaled by utilizing the available as well as emerging computing architectures such as heterogeneous processing units and cloud computing. The latter options are well suited for the proposed parallel algorithm since the amount of data that needs to be sent to the various computing nodes is linear to the length of the time series.

While the proposed algorithm utilizes the special structure of the underlying covariance matrix to produce an exact solution, in future this algorithm can be combined with the existing work in the area of approximate GP methods to

<sup>3</sup><http://math.nist.gov/javanumerics/jama/>

<sup>4</sup><http://www.nccs.gov/computing-resources/frost/>

achieve further speedups while staying close to the exact solution.

## 8. ACKNOWLEDGMENTS

Prepared by Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, Tennessee 37831-6285, managed by UT-Battelle, LLC for the U. S. Department of Energy under contract no. DEAC05-00OR22725. This research is funded through the LDRD program at ORNL.

## 9. REFERENCES

- [1] S. Brahim-Belhouari and J. Vesin. Bayesian learning using gaussian process for time series prediction. In *Statistical Signal Processing, 2001*, pages 433–436, 2001.
- [2] B. J. Brewer and D. Stello. Gaussian process modelling of asteroseismic data. *Monthly Notices of the Royal Astronomical Society*, 395(4):2226–2233, June 2009.
- [3] G. Cybenko. Round-off error propagation in durbin’s, levinson’s, and trench’s algorithms. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP ’79.*, volume 4, pages 498–501, apr 1979.
- [4] J. Dias and J. Leitao. Efficient computation of trTR-1 for toeplitz matrices. *Signal Processing Letters, IEEE*, 9(2):54–56, feb 2002.
- [5] P. Drineas and M. W. Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.
- [6] J. Durbin. The fitting of time-series models. *Revue de l’Institut International de Statistique / Review of the International Statistical Institute*, 28(3):233–244, 1960.
- [7] L. Foster, A. Waagen, N. Aijaz, M. Hurley, A. Luis, J. Rinsky, C. Satyavolu, M. J. Way, P. Gazis, and A. Srivastava. Stable and efficient gaussian process calculations. *J. Mach. Learn. Res.*, 10:857–882, 2009.
- [8] A. Girard, C. E. Rasmussen, J. Quinonero-Candela, J. Q. N. Candela, M. Modelling, and R. Murray-smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems*, pages 529–536. MIT Press, 2003.
- [9] G. H. Golub and C. F. V. Loan. *Matrix Computations*. Johns Hopkins Press, Baltimore, MD, USA, second edition, 1989.
- [10] G. Jun, R. R. Vatsavai, and J. Ghosh. Spatially adaptive classification and active learning of multispectral data with gaussian processes. In *ICDMW ’09*, pages 597–603, 2009.
- [11] T. Kailath and A. H. Sayed, editors. *Fast reliable algorithms for matrices with structure*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.
- [12] A. J. Keane, A. Choudhury, A. Choudhury, P. B. Nair, P. B. Nair, A. J. K. F. Choudhury, and P. B. Nair. A data parallel approach for large-scale gaussian process modeling. In *in Proc. the Second SIAM International Conference on Data Mining*, 2002.
- [13] N. Levinson. The Wiener RMS error criterion in filter design and prediction. *Journal of Mathematics and Physics*, 25(4):261–278, 1947.
- [14] A. O’Hagan and J. F. C. Kingman. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B*, 40(1):1–42, 1978.
- [15] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2005.
- [16] B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society. Series B (Methodological)*, 47(1):1–52, 1985.
- [17] W. F. Trench. An algorithm for the inversion of finite toeplitz matrices. *SIAM Journal on Applied Mathematics*, 12(3):515–522, 1964.
- [18] W. F. Trench. Weighting coefficients for the prediction of stationary time series from the finite past. *SIAM Journal on Applied Mathematics*, 15(6):1502–1510, 1967.
- [19] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.
- [20] S. Zohar. Toeplitz matrix inversion: The algorithm of W. F. Trench. *J. ACM*, 16(4):592–601, 1969.
- [21] S. Zohar. The solution of a toeplitz set of linear equations. *J. ACM*, 21(2):272–276, 1974.