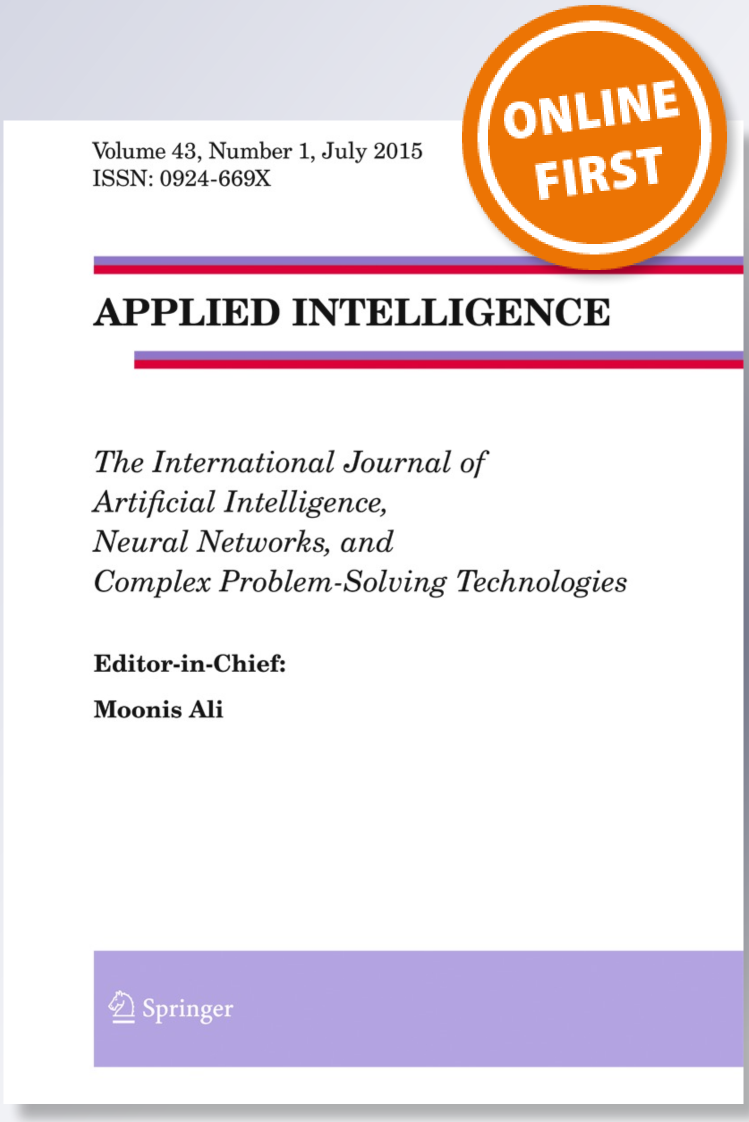
Gaussian process for predicting CPU utilization and its application to energy efficiency

Dinh-Mao Bui, Huu-Quoc Nguyen, YongIk Yoon, SungIk Jun, Muhammad Bilal Amin & Sungyoung Lee

Applied Intelligence
The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies

ISSN 0924-669X

Appl Intell
DOI 10.1007/s10489-015-0688-4


Volume 43, Number 1, July 2015
ISSN: 0924-669X
ONLINE FIRST
APPLIED INTELLIGENCE
The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies
Editor-in-Chief:
Moonis Ali
Springer

Appl Intell
DOI 10.1007/s10489-015-0688-4

Gaussian process for predicting CPU utilization and its application to energy efficiency

Dinh-Mao Bui[1] · Huu-Quoc Nguyen[1] · YongIk Yoon[2] · SungIk Jun[3] · Muhammad Bilal Amin[1] · Sungyoung Lee[1]

Abstract For the past ten years, Gaussian process has become increasingly popular for modeling numerous inferences and reasoning solutions due to the robustness and dynamic features. Particularly concerning regression and classification data, the combination of Gaussian process and Bayesian learning is considered to be one of the most appropriate supervised learning approaches in terms of accuracy and tractability. However, due to the high complexity in computation and data storage, Gaussian process performs poorly when processing large input dataset. Because of the limitation, this method is ill-equipped to deal with the large-scale system that requires reasonable precision and fast reaction rate. To improve the drawback, our research focuses on a comprehensive analysis of Gaussian process performance issues, highlighting ways to drastically reduce the complexity of hyper-parameter learning and training phases, which could be applicable in predicting the CPU utilization in the demonstrated application. In fact, the purpose of this application is to save the energy by distributively engaging the Gaussian process regression to monitor and predict the status of each computing node. Subsequently, a migration mechanism is applied to migrate the system-level processes between multi-core and turn off the idle one in order to reduce the power consumption while still maintaining the overall performance.

Keywords Proactive prediction · Bayesian learning · Gaussian process · Parallel computing · Energy efficiency · CPU utilization

✉ Sungyoung Lee
sylee@oslab.khu.ac.kr

Dinh-Mao Bui
mao.bui@khu.ac.kr

Huu-Quoc Nguyen
quoc@khu.ac.kr

YongIk Yoon
yiyoon@sm.ac.kr

SungIk Jun
sijun@etri.re.kr

Muhammad Bilal Amin
mbilalamin@oslab.khu.ac.kr

[1] Department of Computer Engineering, Kyung Hee University, Suwon, Republic of Korea

[2] Department of Multimedia Science, SookMyung Women's University, Seoul, Republic of Korea

[3] HPC System Research Section/Cloud Computing Department, ETRI, Daejeon, Republic of Korea

1 Introduction

Gaussian process is a stochastic process that exists in many research fields such as data communication, networking and computer science. It is widely used as a non-parametric and probabilistic approach to model the characteristics of the target system. Rather than determining the parameters of the model from scratch, Gaussian process helps adapt these parameters to represent the actual underlying function. As such, Gaussian process is a suitable choice for noisy, corrupted or erroneous data. In fact, there are numerous applications that adopt Gaussian process comprising various machine learning solutions such as image classification, dimensional reduction, unsupervised learning problems [1] and time series prediction. Compared to other famous methods including linear regression, k-nearest neighbor, multivariate regression splines, multi-layer perceptron and support vector regression, the Gaussian process regression (GPR) outperforms them in terms of both accuracy and flexibility [2]. Furthermore, the output of GPR is also simple. It comprises a set of mean and variance from the Gaussian distribution, which represents the predictive value and the confidence of the prediction, respectively.

Because of these properties, GPR should be a major solution for inference and interpolation purposes. However, this technique does have one significant drawback. In a standard implementation, GPR costs $O(n^3)$ for computational complexity and $O(n^2)$ for storage complexity when calculating n training points of dataset [3]. Theoretically, these results come from the matrix inversion and the log determinant calculation. Furthermore, for a large-scale system, the problem is even worse in terms of reaction rate.

There has been a lot of research aiming at overcoming these limitations. Some of these research focus on utilizing the well-known mathematical methods (like Cholesky decomposition and reduced rank covariance matrix) to reduce the complexity to $O(cn^2)$ and $O(cn)$ for computation and storage, respectively c is a constant being equal to $1/6$ and n is the number of training point of dataset. These parameters together express the cost of Newton iterations which is described in detail in [4]). Nevertheless, it is still not fast enough to satisfy the strict requirement of large-scale system, which prefers the solution to be as simple as possible. Others research is related to the optimization techniques, which seem to be effective. Nevertheless, these techniques also encounter the reliability issues and will still be slow in comparison with the mathematical methods. Further detail about these approaches is included in the Related Works section.

In this paper, we propose a solution for the complexity problem in Gaussian process regression which is particularly related to time series prediction in the periodic spatial-temporal dimension. Theoretically, this approach constructs a combination of mathematical modeling, convex optimization and parallel computing to drive the complexity to $O(nlogn)$ for the hyper-parameter learning, $O(n/n_p)$ for the training phase and $O(n)$ for the storage, where n is the number of training point calculating on n_p computing nodes. These improvements are applicable to the large dataset and can be suitably integrated into the large-scale system. In order to verify the proposed idea, an application aiming at proving the effectiveness of the proposed method is devised. Basically, this application predicts the multi-core CPU utilization and issues the process migration between the cores in order to achieve the overall energy efficiency while still maintaining the high performance.

This paper is organized as follows. In Section 2, we provide the related works that are relevant to the topic. In Section 3, we present the motivation behind the proposed idea and the energy saving application. We detail the regression framework in Section 4. In Section 5, we conduct the performance evaluation of the demonstrative application. Our conclusion and direction for future work are summarized in Section 6.

2 Related works

Gaussian process has been in use for a long time with a multitude of successful applications. As previously mentioned, the related research and applications mainly focus on classification and regression, particularly for time series modeling [5]. In more specific, most of the research examines the theory and conducts the experiments to clarify the role of covariance matrix, mean function and hyper-parameters in Gaussian process. In addition, various kernel functions such as Radial Basis Function (RBF), Matérn Kernel Function, Rational Quadratic Kernel, Multi inputs & outputs, and Periodic & Quasi-periodic kernels [6] are also studied to show the flexibility of Gaussian process in modeling many kinds of dataset.

In the research concerning the usability of Gaussian process for predicting chaotic time series [7, 8], the evolving Gaussian process method has been proposed to identify chaotic time series events in the system as an on-line method for modeling. In this method, the information is received in the streaming mode and the hyper-parameters are then adjusted to adapt the prediction model to the data. In other words, an on-line sparse method for Gaussian process modeling has been introduced to sequentially process the continuous bulk of the information (which is contained in the streaming data) and accommodate the hyper-parameter values with the incoming data. Studying the hyper-parameters on-line is interesting; however, in the optimization phase, this method only engages the Conjugate Gradient (CG) iterative technique for minimizing the negative log likelihood function and Cholesky decomposition for updating the covariance matrix . For more information, CG is the famous optimization method to solve sparse systems of linear equation, especially the sparse matrices [9]. Similarly, Cholesky decomposition is popularly used to avoid directly inverting the matrix in solving linear systems because of the stability and convenience in calculation. However, as shown above, these techniques only reduce the computational complexity to $O(cn^2)$ and obviously not much improve the overall performance of the system.

Another approach for on-line hyper-parameter learning is introduced in [10], where an adaptive controller is constructed. In this research, the optimization method is proposed to enhance the on-line hyper-parameters estimation. The optimization technique engaged in the estimation is the Stochastic Gradient Ascent which is empirically suitable for on-line learning. Furthermore, an uncorrelated point removal scheme is also included to reduce the dataset size. In this scheme, only sufficient data points are kept when recalculating the hyper-parameters and the kernel function values. The computational complexity of this approach is $O(n^2)$ for n training points, which is still high when dealing with the large dataset.

To better handle big volume of dataset, the research in [11] proposes the use of the random projection method to reduce the dimension of the data point. This technique is applied to matrix approximation to reduce the rank of covariance matrix and consequently reduce the computational complexity from $O(n^3)$ to $O(n^2m)$, where n is the number of data point and m is the best rank approximation for the covariance matrix. This method improves the matrix inversion and also generalizes the random projection algorithm. Comparing to other optimization methods, this approach is not much impressive in terms of the results; however, it still shows the potential for clustering and classifying the general data matrix.

Other research on Gaussian process for Big Data [12] focuses on the Stochastic Variational Inference (SVI) method for constructing Gaussian process model. One interesting advantage of this approach is the independent complexity with regard to the dataset. This method allows variational inference performing on a very large dataset and shows impressive performance in terms of the speed of inference. Nonetheless, the SVI only works properly in the special system in which the observational and latent variables are globally factorized. Unfortunately, Gaussian process does not have that kind of global variable. In addition, because of the input-independent property, the SVI-oriented methods are affected by large variance which results in less precise solution.

Another approach utilizes the KD-Tree on fast Gaussian process regression [13] which enhances the speed of the hyper-parameter learning phase. Theoretically, the computational complexity of this particular phase decreases from $O(n^3)$ to $O(n)$ for n training points which is a significant improvement. In this method, the dataset is recursively partitioned into many subsets and assigned to corresponding nodes of a binary tree. In addition, the cached information of the weighted sum for each node is also included in order to accelerate the summation procedure. Although the KD-Tree technique expedites the hyper-parameters estimation, the training phase still tolerates low performance by using the Conjugate Gradient iterative method when processing large dataset. Furthermore, the computational cost of $O(nlogn)$ is also added, since the algorithm must invest in advance to build build the KD-Tree. Because of the extra computation, this approach is less than ideal for the problem of numerous data points.

In contrast with the aforementioned KD-Tree technique, the Improved Fast Gauss Transform (IFGT)[14], which is a Matrix-Vector Multiplication (MVM) method, performs better in both the hyper-parameter learning and the training phase. This approach improves upon the Fast Gauss Transform (FGT) (which is originally derived from the well-known Fast Multipole Method [15]) by adaptively choosing the precision parameter during the approximation process. In addition, the IFGT shares the divide-and-conquer mechanism that is similar to the one used in the KD-Tree technique. In essence, the IFGT also partitions the domain by k-centers clustering, then caches the sum of contribution in each level for accelerating purpose. Despite this notable improvement, the IFGT still has a critical drawback because this method mostly relies on the FGT which is only effective if the objective function can be expanded to the Gaussian-type potential (the definition of *potential* or *layer potential*, which is originally derived from the *potential theory* [16], firstly used to solve linear constant-coefficient parabolic partial differential equations by transforming the original kernel formula to many layers for computational convenience. Each layer is named as a *potential*. In that sense, the Gaussian-type potential is actually the Gauss transform of the kernel in both physical and Fourier domains to layers). Unfortunately, this requirement has not always been satisfied in the hyper-parameter learning phase. This problem results in unreliable calculation for the hyper-parameters estimation. Further elaboration of this issue and potential solution will be addressed later in this research.

Although there has been a significant amount of research focusing on Gaussian process regression, not much progress has been made in terms of performance improvement, especially the optimization. Furthermore, due to the fact that large dataset is popular in the current distributed system, it is necessary to develop a low-complexity and reliable method to process the data and give the reliable predictive information in acceptable processing speed.

3 Motivation

3.1 Domain analysis

Rather than discussing the theory of Gaussian process regression (GPR), it is more productive to develop a solution based on a popular problem in order to showcase the methodology. In this research, we attempt to predict the workload of each core in the CPU. In the first step, we would like to discuss the domain of the problem of interest. Intuitively, the computation is mostly based on the monitoring statistic of CPU multi-core. This statistic is periodic, noise-free, and consists of two quantities: the arrival rate of the incoming processes and the service rate of each core with regard to the time epoch. In light of the queuing theory, we model these quantities *via* $M/M/s$ Markov Chain (Fig. 1). For more information, $M/M/s$ Markov Chain is a stochastic process in which the first M stands for the Poisson arrival rate of customer, the second M is the exponential service rate of the server and the s generally represents the number of server providing service in the system. In this model, the CPU comprises s - the number of core with identical processing capability. Each core is represented by the service rate μ. The system processes, which follow the arrival rate λ, continuously come to the processor to be served. Subsequently, the departure of completed task begins and the complementary resources are cleaned. Depending on these parameters, the CPU utilization denoted by ρ is formulated as follows:

$$\rho = \frac{\lambda}{s\mu} \tag{1}$$

Typically, the nature of inter-arrival time of incoming processes and the service time of the CPU core is exponential, which is similar to the other arrival rates in communication and service providing. Therefore, the counting procedure in this case is truly a Poisson process with regard to the time. Due to these aforementioned reasons, the supervised machine learning technique (or regression technique in detail) is the most suitable for handling the problem of uncertainty and time series forecasting [6].

3.2 Approaches

According to the predefined domain, our goal is to predict the desired information from the input value. This means that the predictive utilization of the CPU core in the future time is expected after investigating the input data. Likewise, the relationship between the input data and the output interpolation should be re-constructed. In other words, the underlying function producing an output from the input is needed to analyze. To do that, there are two common approaches: restriction bias and preference bias. While the former approach restricts the class of function to consider, the latter approach takes into account the probability of every possible function. The higher probability the function achieves, the more likely it is chosen for implementation. For each of these methods, there are some drawbacks that should be investigated.

The first approach generates the poor prediction if the chosen function and the underlying function are mismatched. Moreover, increasing the number of chosen function to adapt to the training data might lead to an issue of over-fitting. Although the trained algorithm would be able to work properly with the training data, the accuracy would be dramatically decreased when coping with the real-world data.

In contrast to the restriction bias, the preference bias provides the flexibility in terms of choosing the function. However, this approach still encounters some critical obstacles leading to the poor performance. As there are numerous sets of possible function to examine, evaluating all of them in a finite amount of time is almost impossible. Therefore, it is necessary to determine a method which is capable of achieving the necessary flexibility and obtaining the targeted output at a faster rate.

The anticipation on arrival rate of incoming process over time shapes a time series function [17]. For instance, a time series $y(t)$ with regard to the time t is actually a non-deterministic function y of the independent variable t. Modeling this kind of variable by applying the regression technique reveals the relationship between the aforementioned arrival rate in the present and the future:

$$y(t) = f(t) + \varepsilon \tag{2}$$

in which, $f(t)$ is the underlying function and ε is a zero mean additive noise process. Due to the domain of interest, it is known that both the input and output are noise-free. As a result, the parameter ε must be equal to zero.

With an input-output pair being considered as an observation, the problem of interest is the probability of Y at given time t. Fortunately, the expression of this problem also implies the solution, or at least the proper method to determine the solution. Theoretically, this is similar to the Bayes' rule, which is used in probability inference [18]. Therefore, a suitable inference technique should engage the Bayesian approach [5, 19] as follows:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{3}$$

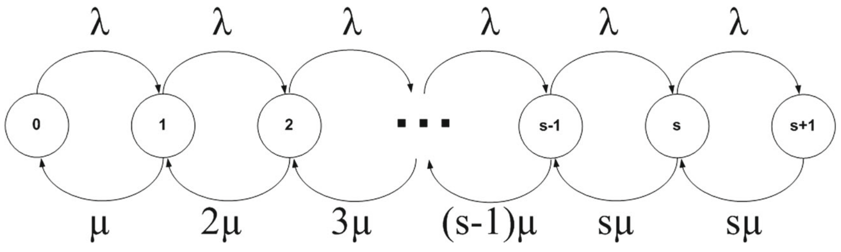


Fig. 1 Markov chain representation for processing activity in CPU cores

At this point, it is beneficial to understand the characteristics of the arrival process. As briefly mentioned in the Domain analysis section of this paper, the probability density of the counting process for the arrival is the Poisson probability density, which is represented by the following function:

$$f(k, \lambda) = P(Y = k) = \frac{\lambda^k \exp(-\lambda)}{k!} \tag{4}$$

According to the Central Limit Theorem [18], when the arrival rate is large (as the arrival rate of incoming process in large-scale system would be), the Poisson distribution converges to the Gaussian distribution [12]. As a result, it makes sense to integrate the Gaussian process framework into the Bayesian learning approach [5, 20]. Comparing to the other regression methods [10], Gaussian process framework is one of the most suitable method for dealing with regression problem, since it is powerful, non-parametric, and flexible [2, 21]. In addition, this framework is also capable of uncertainty estimation using the training data, without the requirement for an explicit declaration from the prediction function.

4 Proposed method

4.1 Target process

To optimize the Gaussian process regression in the spatial-temporal dimension, we choose to use an energy saving application for demonstration. The energy efficiency architecture for the multi-core CPU is proposed in this section and described in Fig. 2. Basically, the purpose of this architecture is to pro-actively reduce the energy consumption of the CPU. Thus, the main functionality of this application is to empty the workload of the CPU cores and then deactivate or stand-by the idle cores to save the energy. To do this effectively, the migration procedure on the Migrator component requires the predictive information of the CPU core utilization to determine the source and the destination in

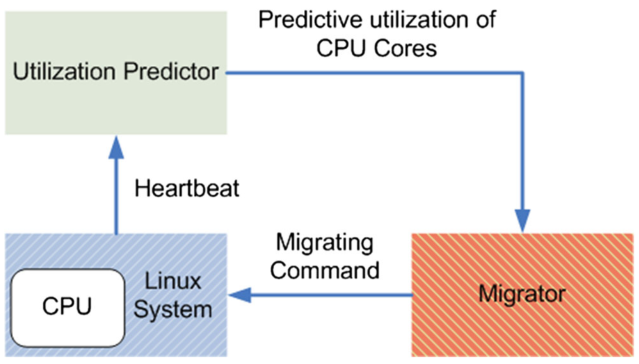


Fig. 2 The architecture of energy saving application

order to migrate the target process (from this point, the system process considered for migrating is known as the target process). Primarily based on the current value of the CPU core's utilization, which is known as the heartbeat, the predictive utilization is calculated in the Utilization Predictor and plays a critical role in decision making. For dealing with the rapid change in the utilization of the CPU cores, the energy saving application is implemented as a background daemon in each computing node. This background daemon is responsible for migrating the system processes between the local CPU cores.

4.2 Source and destination

Based on the predictive information coming from the Utilization Predictor, the Migrator reaches halfway to the final goal. In fact, the predictive utilization of the CPU cores can be employed to decide which core would be the best suited as a source core to extract the target process. Typically, the core with the lowest utilization is chosen to be the source core. Then, it is necessary to choose the destination core for placing the extracted target process. Depending on the basic knowledge of the operating system, as well as the queuing theory, various critical conditions must be considered in order to judge the destination. First, the destination core should not be the same as the source core. Moreover, the destination core should not be blocked (the core should not be too busy, as it would fail to accept the target process). Finally, when the first two conditions are matched, the core with the highest estimation of utilization is selected as the next destination core.

Assuming that an arriving process comes to a specified CPU core, that process might 'see' an average number of system process which are being held in progress. Because the arrival counting follows the Poisson process, the Poisson Arrival See Time Averages (PASTA) theory [22] is applicable when estimating the desired average number of system process. Following the PASTA theory, the average arrival time is considered to be equal to the time average or the expectation of waiting time in the CPU core. By applying the Pollaczek-Khintchine formula [22], the expectation waiting time $\mathbf{E}_j$ of CPU core j can be calculated as follows:

$$\mathbf{E}_j(W) = \frac{\lambda_\tau 1/\mu^2}{2(1 - \lambda_\tau 1/\mu)} \tag{5}$$

in which, λ_τ is the arrival rate of the incoming process at time τ, μ is the service rate of the CPU core and W is the expected queuing delay in CPU core [22]. After estimating the arrival average, the blocking rate of the CPU core can be obtained by dividing this value by the service rate of the core:

$$\mathbf{BR}_j = \frac{\mathbf{E}_j(W)}{\mu} \tag{6}$$

By evaluating the blocking rate $\mathbf{BR}_j$ of each CPU core j, it is easy for the Migrator to determine the destination for migrating the target process. If there is no available core for destination promotion, the migration procedure is canceled until there is at least one unblocked core. Finally, in the worst case scenario in which all of the on-line cores (including the CPU core being considered) are blocked, a re-activation procedure is issued in order to re-activate the stand-by cores one by one. This strategy helps the CPU maintain an acceptable performance when dealing with the incoming processes.

4.3 Prediction model

In our application, the object of prediction is to anticipate the utilization of CPU cores. Bayesian learning and Gaussian process regression are employed as the inference technique and probability framework, respectively. Because the input data for this model is the time series utilization, curve-fitting is preferred over function mapping for the mapping approach. It is important to note that the curve-fitting is more flexible with regard to the time series data and non-stationary model.

Assuming that the input data is a limited collection of time location $x = [x_1, x_2, x_3, \cdots x_n]$, a finite set of random variable $y = [y_1, y_2, y_3, \cdots y_n]$ represents the corresponding joint Gaussian distribution of incoming processes with regard to the time order. This set over the time constraint actually forms up the Gaussian process:

$$f(y|x) \sim \mathcal{GP}\left(m(x), k(x, x')\right) \tag{7}$$

with

$$m(x) = \mathbb{E}\left(f(x)\right) \tag{8}$$

$$k(x, x') = \mathbb{E}\left(\left(f(x) - m(x)\right)\left(f(x') - m(x\prime)\right)\right) \tag{9}$$

in which, $m(x)$ is the mean function, evaluated at the time location variable x, and $k(x, x')$ is the covariance function, also known as the kernel function [23]. By definition, the kernel function is a positive-definite function which is used to define the prior knowledge of the underlying relationship. Basically, the kernel function is only a mandatory requirement when there is lack of finite dimensional form of the feature space. Otherwise, it can be dropped by directly calculating the sample. However, this feature space dimension is frequently infinite, which means that the kernel function cannot be directly calculated. For this reason, the kernel function technique is often chosen to tackle the Gaussian process regression. In addition, the kernel function comprises some special parameters that specify its own shape. These parameters are referred to as the hyper-parameters. Because the input data comes to the Predictor as a set of n time locations, the kernel should be engaged in the matrix form.

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix} \tag{10}$$

Generally, the Square-Exponential (SE) kernel, also known as the Radial Basis Function (RBF) kernel, is chosen as the basic kernel function. In reality, the SE kernel is favored in most of the Gaussian process applications, because it requires the calculation for only few parameters. Moreover, there is a theoretical reason to choose this method, as it is an appropriately universal kernel for any continuous function whenever enough data is given. The formula for SE kernel is described as follows:

$$k_{SE}(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{11}$$

in which, σ_f is the output-scale amplitude and l is the time-scale of the variable x from one moment to the next. l also stands for the bandwidth of the kernel and the smoothness of the function. In addition, l also plays the role of judgment for Automatic Relevance Detection (ARD) to discard the irrelevant input dimension. Although the SE kernel has many computational benefits, its drawbacks include unreasonable smoothness assumption and underestimating the variance of prediction. Due to this reason, the Matérn kernel may be considered as a good alternative. The formula for Matérn kernel is described as follows:

$$\begin{aligned} k_M(x, x') &= \sigma_f^2 \frac{1}{\Gamma(\nu)2^{\nu-1}} \left(\sqrt{2\nu}\frac{|x - x'|}{l}\right)^{\nu} \\ &\quad \times \mathbb{B}_\nu\left(\sqrt{2\nu}\frac{|x - x'|}{l}\right) \end{aligned} \tag{12}$$

where σ_f is the output scale, l is the time-scale between the moment of variable x and x', $\Gamma()$ is the standard Gamma function and $\mathbb{B}_\nu$ is the modified second order Bessel function. An extra hyper-parameter ν specifies the roughness, also known as the degree of differentiation. For machine learning purposes, ν regularly equals to $3/2$ or $5/2$ [4]. Since ν should not be too high or it will distort the shape of the prediction function at the edges. One disadvantage of the Matérn kernel is computationally intensive due to the complexity of solving the Bessel and Gamma functions. Among these kernels, it is necessary to rely on the learning strategy to decide the suitable kernel as the covariance function. In this research, the SE kernel is chosen to characterize the input training data in order to balance between the accuracy and the reaction rate.

In the next step, we evaluate the posterior distribution of the Gaussian process. Assuming that the incoming value of the input data is (x_*, y_*), the joint distribution of the training output is y, and the test output is y_* as shown below:

$$p\left(\begin{bmatrix} y \\ y_* \end{bmatrix}\right) = \mathcal{GP}\left(\begin{bmatrix} m(x) \\ m(x_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(x,x')\mathbf{K}(x,x_*) \\ \mathbf{K}(x_*,x)\mathbf{K}(x_*,x_*) \end{bmatrix}\right) \tag{13}$$

here, $\mathbf{K}(x_*, x_*) = k(x_*, x_*)$, $\mathbf{K}(x, x_*)$ is the column vector made from $k(x_1, x_*), k(x_2, x_*) \cdots , k(x_n, x_*)$. In addition, $\mathbf{K}(x_*, x) = \mathbf{K}(x, x_*)^T$ is the transposition of $\mathbf{K}(x, x_*)$. Subsequently, the posterior distribution over y_* can be evaluated with the below mean m_* and covariance C_*.

$$m_* = m(x_*) + \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}(y - m(x)) \tag{14}$$

$$C_* = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{15}$$

Then

$$p(y_*) \sim \mathcal{GP}(m_*, C_*) \tag{16}$$

The best estimation for y_* is the mean of this distribution:

$$\overline{y}_* = \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}y \tag{17}$$

In addition, the uncertainty of the estimation is captured in the variance of the distribution as follows:

$$var(y_*) = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{18}$$

4.4 Hyper-parameter learning phase

The proposed model invokes a set of hyper-parameter $\theta = [\sigma_f, l]$ which exists in covariance and mean function. In definition, these hyper-parameters are supposed to be evaluated through the marginalization process. By using the Bayes' rule, the (16) can be rewritten as:

$$p(y_*|y) = \frac{\int p(y_*|y,\theta)p(y|\theta)p(\theta)\mathrm{d}\theta}{\int p(y|\theta)p(\theta)\mathrm{d}\theta} \tag{19}$$

In this equation, the marginal likelihood $p(y) = \int p(y|\theta)p(\theta)\mathrm{d}\theta$ is the main point of interest. Theoretically, the Maximum A Posteriori (MAP) estimation of θ can be obtained when $p(\theta|y)$ reaches its maximum [24]. From the first moment of inference process, the prior $p(\theta)$ must be assigned for the hyper-parameters to reflect the domain knowledge. Based on the input data, this task is not an obstacle. In addition, according to the Bayes' rule, the probability $p(\theta|y)$ is known to be proportional to $p(y|\theta)$. Therefore, the optimization step only involves maximizing the log $p(y|\theta)$ or minimizing the negative log $p(y|\theta)$ [25].

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\log|\mathbf{K}| + \frac{n}{2}\log(2\pi) \tag{20}$$

The partial derivative of this negative log marginal likelihood with regard to each hyper-parameter is known as:

$$-\frac{\partial}{\partial\theta_i}\log p(y|\theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_i}\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}tr(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_i}) \tag{21}$$

As previously stated, estimating the hyper-parameter set θ can be achieved by minimizing the negative log marginal likelihood. In other words, the hyper-parameter set θ is evaluated by taking the first order partial derivatives of the negative log marginal likelihood and setting them to zero. This is the classic, trivial method to construct the learning scheme for the hyper-parameters. Nonetheless, solving these derivative equations drives the whole computation into the state of Worst Case Execution Time (WCET). The reason is the high complexity in re-calculating the matrix inversion and partial derivative matrix for every hyper-parameter. Therefore, it is necessary to determine another approach for hyper-parameter learning phase. Instead of putting effort into minimizing the negative log marginal likelihood, this heavy-load-job can be done faster by minimizing the upper bound of this term approximately [26]. In the (20), the dominant computation focuses on two terms: the data-fit term [4] which is denoted by $\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y}$, and the complexity penalty term or the log determinant log $|\mathbf{K}|$. Before going further, the (20) should be simplified to decrease the complexity. To do that, the asymptotic estimation for the log determinant and the corresponding error term is considered in [27]. To find these terms, the law of log determinant, which is originated in [28], is engaged on the sample covariance matrix. Each element $X_i, i = 1...n$ of the training dataset can be seen as a random sample of the Gaussian distribution. Because the interval time for collecting samples is non-overlapped, periodic and identical, these random samples are independent and identically distributed (i.i.d.). This means the dimension can be reduced from the spatial-temporal to the spatial only [29]. Therefore, the sample covariance matrix can be constructed as:

$$\hat{\mathbf{K}} = \frac{1}{n}\sum_{i=1}^{n+1}(X_i - \bar{X})(X_i - \bar{X})^T \tag{22}$$

One more constant, namely Bias Correction, should be calculated in advance.

$$\tau_n := \frac{\partial \log\Gamma(x)|_{x=n/2}}{\partial x} - \log\left(\frac{n}{2}\right) \tag{23}$$

After determining these two important values, the log determinant $L = \log |\mathbf{K}|$ calculating on the covariance matrix $\mathbf{K}$ can be estimated as:

$$\hat{L} = \log |\hat{\mathbf{K}}| - \tau_n \tag{24}$$

In addition, the Mean Squared Error (MSE) of this log determinant estimation is also evaluated as follows:

$$\mathbb{E}\left(\hat{L} - L\right)^2 \leq -2\log\left(1 - \frac{1}{n}\right) + \frac{10}{3n(n-1)} \tag{25}$$

The log determinant estimation in the (24) is proven in [28] to achieve both the optimal convergence rate as well as the asymptotically optimal constant in low dimensional space. Therefore, this estimation can be adopted as a replacement for the original log determinant. Consequently, the (20) is simplified to:

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\hat{L} + \frac{n}{2}\log(2\pi) \tag{26}$$

in which, in addition to the constant $\frac{n}{2}\log(2\pi)$, when the process runs for a long time, the term $\hat{L}$ also converges to a constant. This means that the partial derivative of negative log marginal likelihood in the (21) is mutually simplified by dropping these constants:

$$-\frac{\partial}{\partial\theta_i}\log p(y|\theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_i}\mathbf{K}^{-1}\mathbf{y} \tag{27}$$

This derivative indicates that the negative marginal log likelihood should only involve minimizing the following reduced negative log marginal likelihood estimation (rMLE):

$$-\log p(y|\theta)_{rMLE} = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} \tag{28}$$

Traditionally, dealing with this task concerns inversing the covariance matrix $\mathbf{K}$. This matrix operation normally costs $O(n^3)$, which is very computationally expensive. Nonetheless, this effect can be mitigated by applying an appropriate solution. Motivated by the application in the stochastic frontier model [24] and Kriging problem [30], the Fast Fourier Transform (FFT) is a promising tool for mitigating this complexity. As previously mentioned, the kernel function is a positive-definite function. Thus, the FFT makes it possible to transform this kernel in order to bring the computation from the spatial-temporal domain (or previously reduced spatial domain) into the frequency domain. After that, the most expensive task is not the matrix inversion, but rather calculating the power spectrum (the quantity shows how much of the signal is at the frequency ω) which only costs $O(nlogn)$. This cost is much less and can be computed faster than the aforementioned traditional approach.

To achieve this advantage, first the Squared Exponential kernel $k_{SE}(x, x')$ (or $k_{SE}(x)$ in general) in the (11) needs to be rewritten in Fourier Transform representation [31] as shown below:

$$\mathcal{F}_{SE}(\omega) = l\sigma_f^2\sqrt{2\pi}exp(-2\pi^2\omega^2l^2) \tag{29}$$

in which, ω is the frequency representation of the time location x in the periodic domain. To accelerate the optimization procedure, the non-uniform Fast Fourier Transform (NUFFT) is applied. According to the reduced negative log marginal likelihood minimization, assuming that Φ is the function that generates $\tilde{\mathbf{K}} = \mathbf{K}^{-1}$. Under the periodic assumption, the Parseval theorem [30] can be applied to derive the Fourier Transform for the (28)

$$\mathcal{F}_{rMLE}(\theta) = \mathcal{F}\left(-\log p(y|\theta)_{rMLE}\right) = \frac{1}{2n}\hat{\mathbf{y}}^T\widehat{\Phi * \mathbf{y}_\circ} \tag{30}$$

in which, the hat sign from $\hat{\mathbf{y}}$ denotes a Fourier transform of $\mathbf{y}$ and $\mathbf{y}_\circ$ denotes the data vector in the periodic domain. In the next step, by continuing to apply the convolution theorem with regard to the constraint $\Phi\mathcal{F}_{SE} \equiv 1$, the final form of the Fourier Transform for the rMLE can be represented, as shown below.

$$\mathcal{F}_{rMLE}(\theta) = \frac{1}{2n}\sum_i \hat{\Phi}_i * \hat{y}_i^2 = \frac{1}{2n}\sum_i \frac{\hat{y}_i^2}{\mathcal{F}_{SE}(\omega_i)} \tag{31}$$

With this form of the (31), it is no longer expensive to determine the set of hyper-parameter by using gradient-based optimizing techniques. In this research, we choose the Stochastic Gradient Descent (SGD), because this technique is suitable for the large dataset, faster than other gradient techniques, and critically less sensitive to the local minima [32]. To integrate the SGD into hyper-parameter learning phase, the partial derivatives of the (31) with regard to each hyper-parameter are required for calculating. These equations are given by:

$$\frac{\partial}{\partial l}\mathcal{F}_{rMLE} = \hat{y}_i^2\exp\left(2\pi^2l^2\omega^2\right)\left(\frac{2\sqrt{2}\pi^{3/2}\omega^2}{\sigma_f^2} - \frac{1}{\sqrt{2\pi}l^2\sigma_f^2}\right) \tag{32}$$

And

$$\frac{\partial}{\partial\sigma_f}\mathcal{F}_{rMLE} = -\frac{\sqrt{\frac{2}{\pi}}\hat{y}_i^2\exp\left(2\pi^2l^2\omega^2\right)}{l\sigma_f^3} \tag{33}$$

After getting the partial derivatives, an updating scheme is issued to update the hyper-parameters. This scheme is as follows.

$$l^{(k)} \leftarrow l^{(k-1)} + \alpha(k)\frac{\partial}{\partial l^{(k-1)}}\mathcal{F}_{rMLE} \tag{34}$$

$$\sigma_f^{(k)} \leftarrow \sigma_f^{(k-1)} + \alpha(k)\frac{\partial}{\partial \sigma_f^{(k-1)}}\mathcal{F}_{rMLE} \tag{35}$$

in which, $\alpha(k)$ is the decay function with regard to the k^{th} iteration. We opt to use the decay function instead of the exact line search or backtracking line search. It is mainly due to the performance issue. For the ease of calculation, a Robbins-Monroe sequence [33] is employed to construct the decay function $\alpha(k) = 1/(k+1)$. In fact, the Robbins-Monroe sequence is popularly used, since it is sufficient to ensure the convergence of the optimization algorithm [34], especially in the SGD method.

To govern the number of iteration for the optimization algorithm (in this case, the SGD), an error function is defined based on the Root Mean Square Error (RMSE) method to measure the convergence. It is important to note that the RMSE method is stricter than the frequently-used Mean Square Error (MSE) method. By using this error function, the error gap between the current iteration value and the previous one can be evaluated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\mathcal{F}_i^{(k)} - \mathcal{F}_i^{(k-1)}}{n}} \tag{36}$$

in which, $\mathcal{F}_i^{(k)}$ and $\mathcal{F}_i^{(k-1)}$ respectively stand for the k^{th} and $k-1^{th}$ iteration values of the rMLE at the target location i. Theoretically, the RSME threshold is limited to 10^{-11} which produces a solution very close to the real one. The purpose of this optimization procedure is to conduct all the steps in the periodic domain. It means that the optimization can be done with no matrix inversion. Additionally, the vector of dual weight $\mathbf{y}_0 \approx \Phi * \mathbf{y}_\circ$ can also be easily estimated in the spectral domain. In the end of this hyper-parameter learning phase, the set of hyper-parameter is ready for the training phase. The hyper-parameter learning algorithm is also described in Algorithm 1 and Fig. 3.

Algorithm 1: Hyper-parameter learning phase

```
   Data: Utilization Statistic of CPU Core. This is the latest history of utilization of
         each core with regard to time step
   Result: Hyper-parameters array $\theta^{(*)} = [l^{(*)}, \sigma_f^{(*)}]$
1  Initialize value for $\theta^{(0)} = [l^0, \sigma_f^0], \omega_{[]}, \epsilon_{RSME}$;
3  /* Fast Fourier Transform of input data                                 */
4  $\hat{y}$= nufft1d1(y);
5  for $k$=1 to sizeof($\hat{y}$) do
7      /* step_size is equivalent to $\alpha$ in the equation (34) and (35)      */
8      step_size=decay_function($k$);
9      $j$=random(1,sizeof($\hat{y}$));
11     /* partial derivative of $\mathcal{F}_{rMLE}$ w.r.t $l$                   */
12     $\nabla l$ = partial_l($\hat{y}_{[j]}, \omega_{[j]}, l^{(k-1)}, \sigma_f^{(k-1)}$);
14     /* partial derivative of $\mathcal{F}_{rMLE}$ w.r.t $\sigma_f$            */
15     $\nabla\sigma_f$ = partial_$\sigma_f$($\hat{y}_{[j]}, \omega_{[j]}, l^{(k-1)}, \sigma_f^{(k-1)}$);
17     /* update hyper-parameters                                          */
18     $l^{(k)} = l^{(k-1)}$+step_size*$\nabla l$;
19     $\sigma_f^{(k)} = \sigma_f^{(k-1)}$+step_size*$\nabla\sigma_f$;
20     Compute $\mathcal{F}_{rMLE}^{(k)}(\theta^{(k)})$;
21     Compute $RMSE^{(k)} = RMSE(\mathcal{F}_{rMLE}^{(k)})$;
22     if ($RMSE^{(k)} \leq \epsilon_{RSME}$) then
23         break();
24     end
25 end
26 return $\theta^{(*)} = [l^{(*)}, \sigma_f^{(*)}]$;
```

4.5 Training phase

In the training phase, most of the computation involves determining the mean value in the (17). In this equation, once the kernel matrix is known, the matrix inversion becomes the main problem. In fact, dealing with the matrix inversion is one of the most intensive computing tasks in optimization. Although the Cholesky decomposition is usually employed to avoid doing the matrix inversion directly, the computational complexity is still $O(n^3)$, where n is the number of training point of dataset. In addition, $O(n^2)$ is also taken into account for matrix storage. This issue is a significant bottleneck for the system. Although the FFT is still effective in this case, because of the change in the objective target, a more suitable technique is used to calculate the prediction.

Firstly, the (17) can be rewritten as:

$$\overline{y}_* = \mathbf{K}(x_*, x)\xi \tag{37}$$

in which

$$\xi = \mathbf{K}(x, x')^{-1} y \tag{38}$$

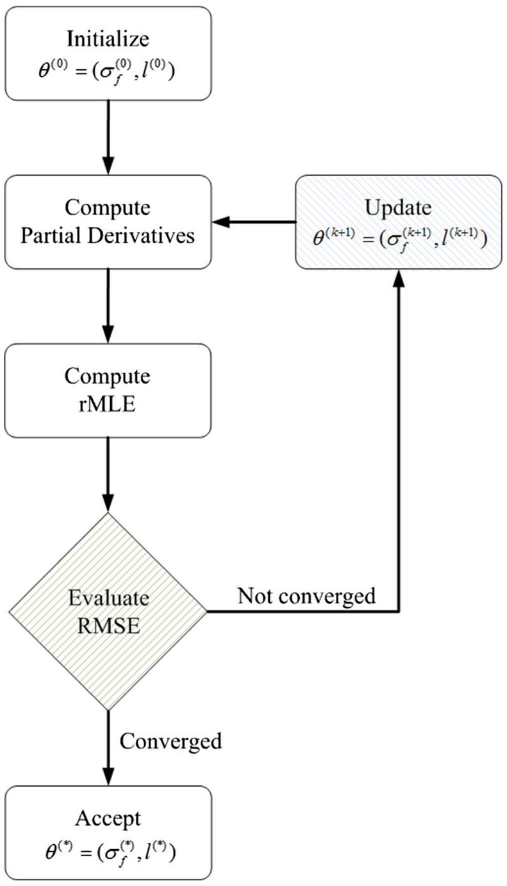

Initialize
$\theta^{(0)}=(\sigma_f^{(0)},l^{(0)})$
Compute
Partial Derivatives
Update
$\theta^{(k+1)}=(\sigma_f^{(k+1)},l^{(k+1)})$
Compute
rMLE
Evaluate
RMSE
Not converged
Converged
Accept
$\theta^{(*)}=(\sigma_f^{(*)},l^{(*)})$

Fig. 3 Hyper-parameter learning algorithm

Multiplying both sides with $K(x, x')$

$$y = \mathbf{K}(x, x')\xi \tag{39}$$

Since the matrix $\mathbf{K}(x, x')$ is symmetric and positive definite, the Conjugate Gradient [9] iterative method is engaged to solve the linear problem in the (39). In this method, a starting point is chosen and a series of steps are created to converge upon the approximate solution ξ_i which is adjacent to the real one ξ. The best solution to this process can be given by the inequality below:

$$\frac{\|\xi-\xi_i\|_{\mathbf{K}}}{\|\xi-\xi_0\|_{\mathbf{K}}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2i} \tag{40}$$

in which, the constant $\kappa = \lambda_{max}/\lambda_{min}$ is the ratio of the largest to the smallest eigenvalue of matrix $\mathbf{K}$, and the $\mathbf{K}-norm$ is calculated as $\|z\|_{\mathbf{K}} = z^T\mathbf{K}z$ with z is any arbitrary vector. The tolerance parameter ζ is also given such that $0 < \zeta < 1$. This parameter is a bound for the practical Conjugate-Gradient scheme.

$$\frac{\|y-\mathbf{K}\xi_i\|_2}{\|y-\mathbf{K}\xi_0\|_2} \leq \zeta \tag{41}$$

in which, at the end of the i^{th} iteration, $\|y-\mathbf{K}\xi_i\|_2$ is obtained as the residual in the Euclidean norm. Regularly, the starting point of the iteration process is ξ_0. Then, the relative error is also obtained as shown below:

$$\frac{\|y-\mathbf{K}\xi_i\|_2}{\|y-\mathbf{K}\xi_0\|_2} \leq \sqrt{\kappa}\frac{\|\xi-\xi_i\|_{\mathbf{K}}}{\|\xi-\xi_0\|_{\mathbf{K}}} \leq 2\sqrt{\kappa}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2i} \tag{42}$$

In fact, a number of iteration is done to achieve the given tolerance parameter ζ. This number can be evaluated as shown below.

$$i \geq \frac{\ln\left(\frac{2\sqrt{\kappa}}{\zeta}\right)}{2\ln\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)} \tag{43}$$

The complexity of the Conjugate Gradient method is $O(in^2)$, in which i is the number of iteration, n is the number of training point of dataset. The storage cost is $O(n)$, due to the fact that the matrix-vector product is able to do the calculation without retaining the whole matrix. The cost of computation and storage in this method is less than the aforementioned direct matrix operation. However, it still does not satisfy the computational intensity of the large-scale system. In fact, the system usually works on an enormous number of dataset. In this case, the quadratic complexity algorithm could deteriorate the overall performance rapidly. Therefore, the Conjugate Gradient needs to be coupled with the Improved Fast Gauss Transform (IFGT) method [35] to achieve even faster calculation. The IFGT technique is derived from the Fast Gauss Transform (FGT) [36–38], which is an $\epsilon - exact$ approximate algorithm. Theoretically, in the j^{th} step of the Conjugate Gradient, with m is the number of target point and n is the number of source point, the FGT firstly expands the j^{th} conjugacy (the A-orthogonal multiplication [9]) $G(x_j)$ into a plane-wave expansion of the previous Square-Exponential kernel (the Matérn kernel might be expanded similarly) as shown below:

$$G(x_j) = \sum_{i=1}^{N} q_i \exp\left(-\frac{\|x_j-x_i\|^2}{2l^2}\right) \tag{44}$$

in which, x_j is the target point with $\{x_j \in \mathbb{R}^2\}_{j=1,\ldots,m}$, q_i is the source weight with $\{q_i \in \mathbb{R}\}_{i=1,\ldots,n}$, x_i is the source point with $\{x_i \in \mathbb{R}^2\}_{i=1,\ldots,n}$, and l is the bandwidth with $\{l \in \mathbf{R}^+\}$. This Gaussian-type expansion can be calculated approximately by using the discrete Fourier Transform [37, 39]:

$$G(x_j) \approx \sum_{|\alpha|\leq p} \mathcal{F}(\alpha) w_\alpha \exp\left(\frac{i\alpha L(x_j-x_i)}{\sqrt{2}pl}\right) \tag{45}$$

with $\mathcal{F}(\alpha)$ and w_α are given by

$$\mathcal{F}(\alpha) = \frac{1}{2^3\sqrt{\pi}} \exp\left(-\frac{L^2|\alpha|^2}{4p^2}\right) \tag{46}$$

$$w_\alpha = \left(\frac{L}{p}\right)^2 \sum_{y\in U} f(y) \exp\left(\frac{i\alpha L(c^U - y)}{\sqrt{2}pl}\right) \tag{47}$$

in which, $\alpha = (\alpha_1, ...\alpha_d)$ is the multi-dimensional index which stands for a d-tuple of non-negative integers (in this context, $d = 2$), p is the number of plan-wave coefficient required per dimension to obtain the desired precision ϵ and L is the truncation error term (the detail configurations of p and L can be found in [39]). Assuming that the domain Ω of interest is a unit square $[0, 1]^2$ because of the spatial-temporal dimension of the domain (if a value stays out of the range, shifting and rescaling have to be performed), by partitioning Ω into uniform squares U of size $\sqrt{2}l$, the FGT might compute the desired result in three steps: **S2W**, **W2L** and **L2T**. Before explaining these terms, the definition of 'interaction list' should be firstly addressed. This list is denoted by $\mathcal{I}[U]$ which describes a specific set of neighbor for U. Basically, this set supports the kernel at the center of U. In the beginning, the FGT algorithm starts with the **S2W** step. This step sequentially calculates the (45) for each square U. Subsequently, the plane-wave expansion, which is created in **S2W**, propagates to all of the elements V of $\mathcal{I}[U]$ as a local expansion. The step **W2L** plays its role by modifying the specified expansion as shown below:

$$w^*_\alpha = w_\alpha \exp(\frac{i\alpha L(c^V - c^U)}{\sqrt{2}pl}) \tag{48}$$

In the last step **L2T**, the conjugacy $G(x_j)$ is computed at x_j using the 'local' expansion from the box containing it:

$$G(x_j) = \sum_{|\alpha|\le p} \mathcal{F}(\alpha) w^*_\alpha \exp\left(\frac{i\alpha L(x - c^V)}{\sqrt{2}pl}\right) \tag{49}$$

For acceleration purposes, the *sweeping* algorithm in [40] is implemented with the FGT method. As previously mentioned, the FGT helps reduce the computational costs to $O(mn)$ with m is the number of target point and n is the number of source point. However, this result suffers a decreasing in accuracy due to the ϵ parameter which also critically influences the p and L parameters. To overcome this drawback, the IFGT proposes a strategy to adaptively select the ϵ parameter without the loss of accuracy. This strategy is based on an improvement to the Krylov subspace method [41] for the symmetric positive definite matrix. In this research, ϵ is chosen using the following inequality:

$$\epsilon_i \le \frac{\delta}{n} \frac{\|y - K\xi_0\|}{\|\check{r}_{i-1}\|} \tag{50}$$

in which, δ is the bound determined by the subtraction of the i^{th} iteration's residual to the corresponding residual of the approximate matrix-vector product: $\|\check{r}_i - r_i\| \le \delta$, and n is the number of training data point. With this enhancement, the complexity in training phase now drops to $O(n)$ which satisfies the reaction rate.

There has been an issue related to whether it is possible to apply the IFGT coupling with the Conjugate Gradient (hereinafter, IFGT-CG) in hyper-parameter learning phase. As shown above, when doing the Conjugate Gradient iteration to solve the linear system in the (39), it requires the Matrix-Vector Multiplication (MVM) which costs $O(in^2)$ where i is the number of iteration and n is the number of training point of dataset. Together, the IFGT-CG reduces this complexity to only $O(n)$ by applying IFGT to the MVM part. Due to this reason, it sounds suitable to engage this combination to the hyper-parameter learning phase to achieve better reaction rate. Unfortunately, the answer is 'yes' for the Conjugate Gradient and 'no' for the IFGT. First, the IFGT, which is derived from the FGT technique, works properly only if the objective function can be represented in the potential form [42] (far field or near field [15]). This strict requirement is impossible for the rMLE in the (28) as well as the partial derivative in the (27). To make this point more clear, assume that $F(x)$ is a scalar or vector field. For a fixed target point y, depending on the location of source point with a predefined range r, the field $F(x-y)$ for x inside the range r is called *near field*. For x out of the range r, this field is defined as *far field*. The problem is that there is no way to transform the objective functions in (27) and (28) to the Gaussian-type potentials. Secondly, the randomized method proposed in the original equation of IFGT [43] requires significant additional computation [3]. It means that minimizing the rMLE is problematic when using the IFGT. In this case, the standalone Conjugate Gradient solves the hyper-parameter learning at the computational complexity of $O(n^2)$ with n is the number of training point of dataset. This complexity is worse than the $O(nlogn)$ of the FFT technique introduced in the previous section. However, the IFGT-CG is still effective when dealing with the optimization step in training phase, especially with the matrix inversion.

4.6 Parallel training phase

Although the optimization procedure benefits immensely from the IFGT and the Conjugate Gradient, the training phase can be further improved by implementing the

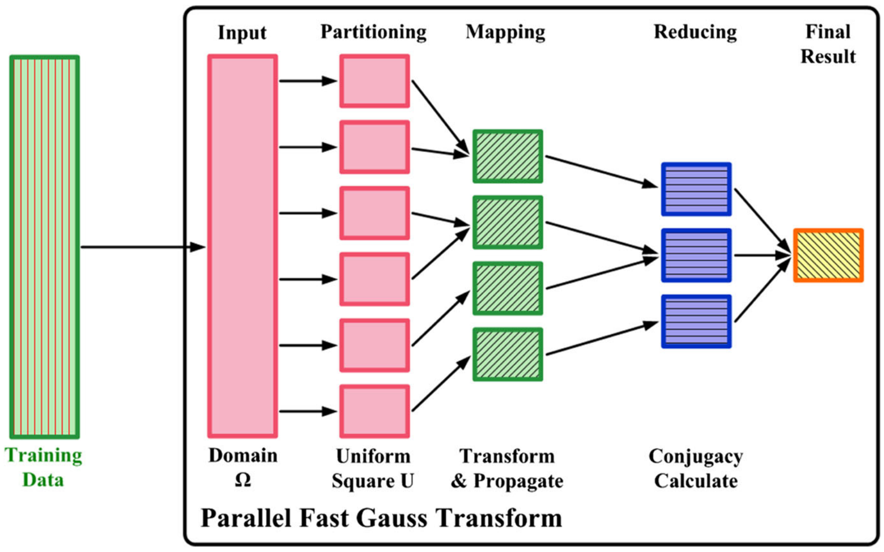


Fig. 4 MapReduce Implementation of Parallel Fast Gauss Transform

parallelism. While maintaining the notion of dynamically choosing the precision parameter ϵ, the FGT operation can be adjusted to enable the parallel computing. As a consequence, the IFGT, which engages the FGT, also shows the improved performance. When examining the structure of the FGT method, it seems natural to combine these three steps of conjugacy (**S2W**, **W2L** and **L2T**) to improve theefficiency of the concurrent calculation. Since **S2W** and **W2L** are closely related to each other, these steps can be merged into one **S2L** step. After merging, there are only two steps for calculation.

In the main part, unlike the parallel method introduced in [40], the idea of parallelism here is to invoke the MapReduce method to take advantage of a robust and flexible parallel implementation. This method can be implemented straightforwardly resulting in the Parallel IFGT which accelerates the training phase (Fig. 4). Because there are two calculation steps, **S2L** can be considered to be the *Map* phase and **L2T** is mapped onto the *Reduce* phase. Prior to this, in the *Map* phase, a grid of separated U squares is partitioned. Then, these squares are distributed to the computing node as the regular input. Depending on the data, the tasks for the Fourier Transform of each square U are created. These tasks follow the (45) and can be controlled by the Task Tracker of the MapReduce framework. After *Map* phase completion, the outcome of each task is propagated to all the members in the interaction list of each square U as a 'local' expansion. Subsequently, the *Reduce* phase receives the updated data from the *Map* phase and creates the task for the **L2T** step. By executing these **L2T** tasks during the *Reduce* phase, the results can be achieved at a much faster rate in comparison with the original FGT method.

Although it is hard to analyze the complexity of the MapReduce operation, the overall complexity of the Parallel FGT performing on n source points and m target points can be estimated as $O(mn/1n_p)$. The reason for this estimation is that the computing tasks with the same complexity are now divided and simultaneously processed at n_p computing nodes of the MapReduce framework. Consequently, this improvement can help reduce the computational complexity of the whole training process to $O(n/n_p)$. The comparison of complexity between the proposed method and the others can be found in Table 1. Finally, the algorithm of parallel FGT is described in Algorithm 2.

Algorithm 2: MapReduce Implementation of Fast Gauss Transform

```
   Data: Utilization Statistic of CPU Core. This is the latest history of utilization of
         each core with regard to time step
   Result: $G(x_j)$ over the square $U$
1  Partition the domain $\Omega$ into square $U$ of size $\sqrt{2}l$;
2  Calculate $G(x_j)$ on each square $U$;
4  /* Execute the S2L step                                              */
5  Distribute square $U$ to n$Map$ job;
6  On $Map$ phase:
7     Execute the 'local' expansion;
8     Propagate the contribution to the interaction list;
10 /* Execute the W2L step                                              */
11 On $Reduce$ phase:
12    Execute the conjugacy;
```

Table 1 Computation cost of proposed method

	Direct method	CG	Proposed method
Hyper-parameter learning phase	$O(n^3)$	$O(n^2)$	$O(nlogn)$
Training phase	$O(n^3)$	$O(n^2)$	$O(n/n_p)$

5 Performance evaluation

5.1 Experiments

For the performance evaluation, our experiments focus on investigating the performance of the proposed application in terms of energy efficiency and execution time. In the initial experiment, the workload is generated *via* the CPU intensive benchmark for one hour to determine the energy saving. In this test, in order to easily control the number and the intensiveness of the workload, the *stress*-1.0.4 benchmark software is used to simulate the incoming processes. Otherwise, in the second experiment, ten bunches of ten concurrent *gzip* jobs (totally one hundred instances of *gzip* command issuing on 256KB of test data) are pushed into the system to test the execution time. To aggregate the results, the *powerstat* and the *sysstat* software are used to log the power consumption and the workload statistics, respectively. All of the information of the benchmarking system is described in Table 2.

5.2 Implementation

The predictive utilization of each core is anticipated using a combination of *Python* script and *C++* library. *Python* is chosen because of the light-weight feature in comparison with *Matlab* [44]. In fact, *Python* possesses a small core of command which is equipped with all the functionality that the researcher would require. Besides, *Python* interpreter is free and available for all of the operating systems. In this combination, in addition to the hyper-parameters estimation source code implemented directly in *Python* for the ease of environmental parameters tuning, the core of Parallel IFGT is implemented in *C++* and wrapped by *ctypes* as a library for compatibly running with the *Python*. The main reason for this particular implementation is related to the performance. In addition to the aforementioned implementation of the proposed method, three other algorithms, namely the original IFGT coupling with the Conjugate Gradient (IFGT-CG), the pure Conjugate Gradient (in short, CG) and the Direct method (the Gauss-Jordan elimination), are also applied for matrix inversion in the hyper-parameter learning phase and training phase for purposes of comparison.

5.3 Metrics

The proposed architecture is measured on two levels: the algorithm level and the application level. In the algorithm level, the metric of interest is the completion time of prediction. In the application level, as previously mentioned, the energy efficiency and the execution time are the metrics of interest. If the daemon application is able to save the energy consumption as well as maintaining an acceptable execution time, the energy efficiency of the CPU would be significantly improved.

5.4 Results

Application level - energy efficiency evaluation as seen in the Fig. 6, both systems begin with the stand-by mode which costs 91.49 watts to maintain. Each system runs the simultaneous stress tests for the duration of 60 minutes. Subsequently, the system with energy saving enabled ends the benchmark test with the power consumption of 154.93 watts, in comparison with 177.96 watts of the regular

Table 2 System configuration

	Configuration
Platform	64bit
CPU	Intel®Core^{TM} i7-3770, 3.40GHz, Quad core
Storage	800GB
Memory	16GB
OS	CentOS 6.5 (final)
	Kernel:2.6.32-431.el6.x86_64
Benchmark	*stress*-1.0.4
Power stat	*powerstat*-0.01.30-1
System stat	*sysstat*-9.0.4-27.el6
Gzip-test-data	text file (256KB)

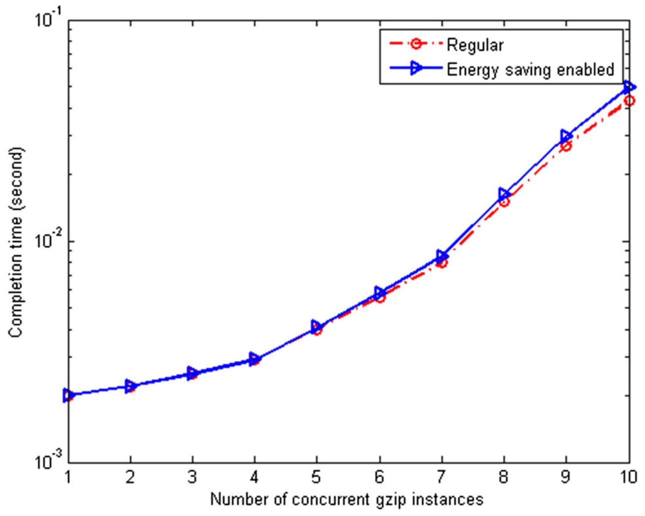


Fig. 5 Execution time comparison between the system with and without energy saving application (lower is better)

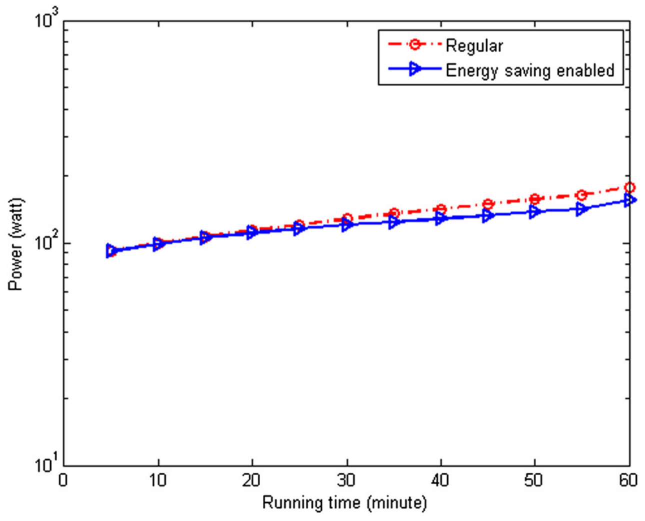


Fig. 6 Power consumption over one hour running time (lower is better)

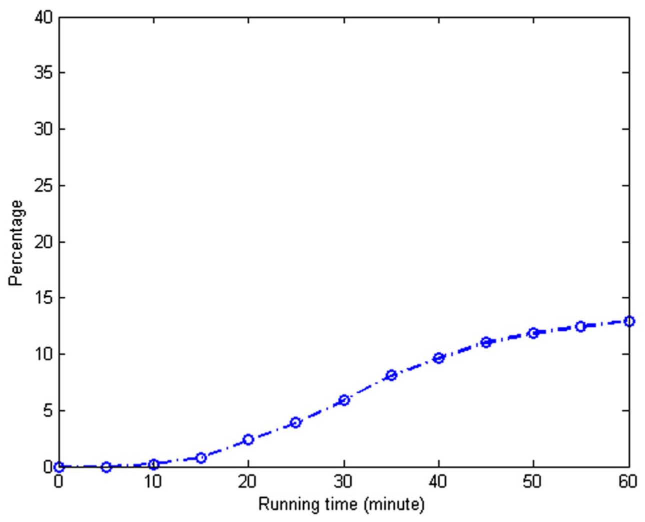


Fig. 7 Power saving over one hour running time

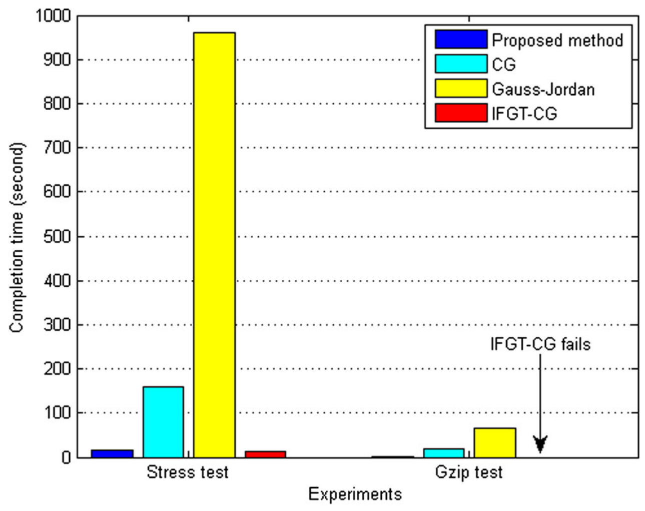


Fig. 8 Hyper-parameter learning speed evaluation (lower is better)

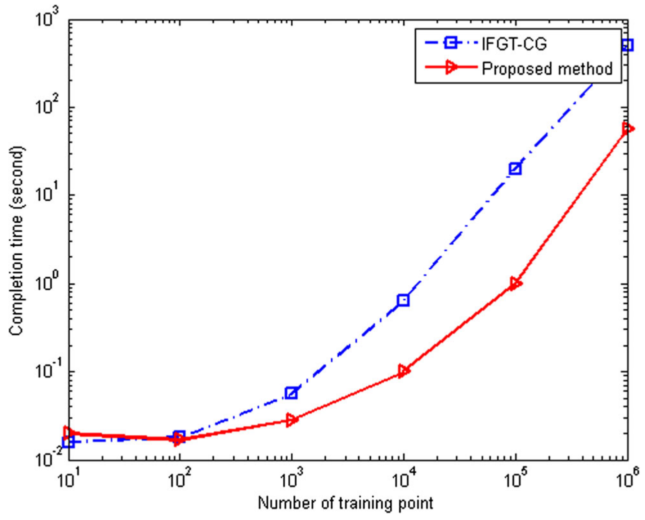


Fig. 9 Training speed evaluation on stress test (lower is better)

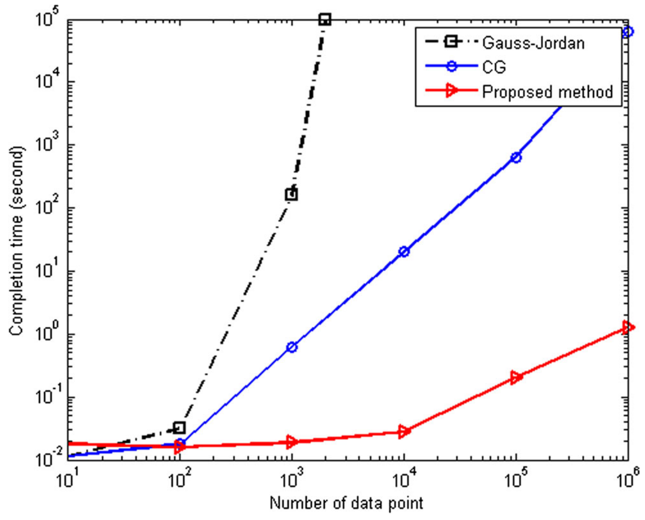


Fig. 10 Overall prediction speed evaluation (lower is better)

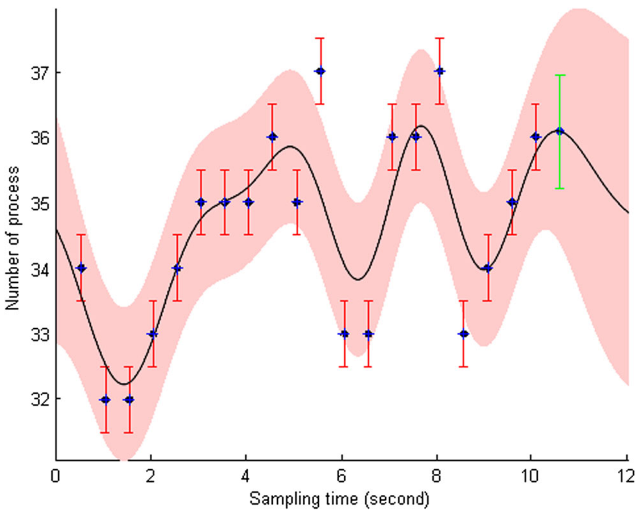


Fig. 11 Mean prediction and error bar of proposed method given 20 training points on stress test

system. Therefore, 23.03 watts are saved (which is equivalent to an energy saving of 12.94 % in Fig. 7). In processor architecture, an energy saving of 12,94 % is significant.

Application level - Execution time evaluation: in the *gzip* experiment, the system engaging the energy saving application is slightly slower. In comparison with the regular system, the energy saving enabled system takes longer running time to finish the equivalent number of task. This extra amount of completion time is measured so as from 2 % to 14 %. In essence, this delay time mostly comes from both predicting the utilization as well as migrating the processes and be considered as the *context switching* cost [45]. In the worst case, despite increasing by 14 %, the time gap between two systems is just 6.43*10^{-3} seconds which is infinitesimal and acceptable (Fig. 5).

Algorithm level - prediction performance as seen in Fig. 8, within the same error bound ($\epsilon = 10^{-11}$) and the same training dataset (around 10^3 points), the proposed method takes 17 seconds to finish estimating the hyper-parameters on the stress test, whereas the CG and Gauss-Jordan elimination cost 160 seconds and 960 seconds, respectively. For a different training dataset (100 target points in *gzip* test), the proposed method spends approximately 1.7 seconds to finish estimating the hyper-parameters, while the CG and Gauss-Jordan elimination cost 20 seconds and 66 seconds, respectively. Particularly for this small test, the original IFGT algorithm tolerates more failure during the computation. This is predominant due to the difficulty in applying the Gaussian-type potential for maximum likelihood estimation which has been discussed in detail in [3]. For this reason, the IFGT-CG is excluded from the hyper-parameter learning estimation procedure. Meanwhile, the proposed method still copes well with the same learning data. Thus, the hyper-parameters estimated by the proposed method are shared to the IFGT-CG algorithm to continue conducting the performance evaluation in the training phase. Moreover, for the evaluation in the training phase as well as the overall prediction, which is described in Figs. 9 and 10, when the number of training point increases, the proposed method continues to significantly outperform the other methods in terms of reaction rate. Finally, for the reliability measurement, because the proposed method also partially relies on the IFGT, which defines the precision of $\epsilon = 10^{-11}$ in advance, the accuracy requirements is always satisfied. In Fig. 11, when doing the accuracy benchmark on 20 consecutive testing points in the stress experiment, the mean prediction is able to adapt well to the testing data, with 95 % confidence maintained by the variance.

6 Conclusion

The proposed method proves the capability in improving the power consumption of the computing node. To do that, the strategy is to predict the utilization of CPU cores, migrate the target processes and stand-by the idle cores to save the energy. To sum up, this method has two major contributions to the research field. First, based on the knowledge of queuing theory, stochastic process, and optimization, the approach reduces the complexity of the hyper-parameter learning phase of Gaussian process regression from $O(n^3)$ to $O(nlogn)$. As a result, the prediction on periodic time series event performs faster, more stably, and more reliably. Second, by applying MapReduce parallelism to the Fast Gauss Transform, we are able to reduce the complexity of the training phase from $O(n^3)$ to $O(n/n_p)$. This improvement continuously increases the reaction rate of the prediction method, makes it possible to deal with the large-scale system. For further development, as previously mentioned, some parts of the source code that are developed to test this method would be made available under the terms of the GNU general public license (GPL). In the near future, we would like to apply the proposed method in Big Data system to improve the performance in terms of processing time and fault tolerance. Besides, implementing the parallelism to the hyper-parameter learning phase of the proposed method is also considered and hopefully would be reflected soon to achieve even faster speed of prediction.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) NRF-2014R1A2A2A01003914.

This work was supported by the IT R&D program of MSIP/IITP. [14-000-09-001, Development of General-Purpose OS and Virtualization Technology to Reduce 30 % of Energy for High-density Servers based on Low-power Processors].

Conflict of interests The authors declare that they have no conflict of interests.

Compliance with Ethical Standards
Research involving Human Participants and/or Animals. The authors declare that this research does not involve any Human Participants and/or Animals.

References

1. Lawrence ND (2004) Gaussian process latent variable models for visualisation of high dimensional data. Advances in neural information processing systems 16:329–336
2. Rasmussen CE (1997) Evaluation of gaussian processes and other methods for non-linear regression, Ph.D. dissertation, Toronto, Ont., Canada, Canada, aAINQ28300
3. Chalupka K, Williams CKI, Murray I (2012) A framework for evaluating approximation methods for gaussian process regression, CoRR. arXiv:1205.6326
4. Rasmussen C, Williams C (2005) Gaussian processes for machine learning, ser. adaptive computation and machine learning, MIT Press. http://www.gaussianprocess.org/gpml/chapters/
5. Brahim-Belhouari S, Vesin J (2001) Bayesian learning using gaussian process for time series prediction. In: Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on, pp 433–436
6. Roberts S, Osborne M, Ebden M, Reece S, Gibson N, Aigrain S (2012) Gaussian processes for time-series modeling. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 371(1984)
7. Petelin D, Filipič B, Kocijan J Optimization of gaussian process models with evolutionary algorithms. In: Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms - Volume Part I, ser. ICANNGA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 420–429. [Online]. Available:, http://dl.acm.org/citation.cfm?id=1997052.1997098
8. Petelin D, Kocijan J. (2014) Evolving gaussian process models for predicting chaotic time-series. In: Evolving and Adaptive Intelligent Systems (EAIS), 2014 IEEE Conference on. IEEE, pp. 1–8.
9. Shewchuk JR (1994) An introduction to the conjugate gradient method without the agonizing pain, Pittsburgh, PA, USA, Tech. Rep
10. Grande R, Chowdhary G, How J (2013) Nonparametric adaptive control using gaussian processes with online hyperparameter estimation. In: 2013 IEEE 52nd annual conference on decision and control (CDC), , pp. 861867
11. Banerjee A, Dunson D, Tokdar S (2011) Efficient gaussian process regression for large data sets. arXiv:e-prints
12. Hensman J, Fusi N, Lawrence ND (2013) Gaussian processes for big data CoRR. arXiv:1309.6835
13. Shen Y, Ng AY, Seeger M (2005) Fast gaussian process regression using kd-trees. In: NIPS
14. Yang C, Duraiswami R, Davis LS (2004) Efficient kernel machines using the improved fast gauss transform. In: NIPS
15. Beatson R, Greengard L (1997) A short course on fast multipole methods. Wavelets, multilevel methods and elliptic PDEs 1:1–37
16. Kress R, Maz'ya V, Kozlov V (1989) Linear integral equations, vol 82. Springer
17. Cunningham J, Ghahramani Z, Rasmussen CE (2012) Gaussian processes for time-marked time-series data. In: Lawrence ND, Girolami M (eds) AISTATS, ser. JMLR Proceedings, vol 22, pp 255–263. JMLR.org
18. Hastie T, Tibshirani R, Friedman J (2001) The Elements of Statistical Learning, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc
19. Tipping ME (2003) Bayesian inference: An introduction to principles and practice in machine learning. In: Bousquet O, von Luxburg U, Rtsch G (eds) Advanced lectures on machine learning, ser. lecture notes in computer science, vol 3176. Springer, pp 41–62. http://dblp.uni-trier.de/db/conf/ac/ml2003.html
20. Chowdhary G, Kingravi H, How J, Vela P (2014) Bayesian nonparametric adaptive control using gaussian processes. IEEE Transactions on Neural Networks and Learning Systems PP(99):1–1
21. Álvarez MA, Lawrence ND (2011) Computationally efficient convolved multiple output gaussian processes. J Mach Learn Res 12:1459–1500. http://dl.acm.org/citation.cfm?id=1953048.2021048
22. Gallager R (2013) Stochastic Processes: Theory for Applications. Cambridge University Press. http://books.google.co.kr/books?id=CGFbAgAAQBAJ
23. Muller K, Mika S, Ratsch G, Tsuda K, Scholkopf B (2001) An introduction to kernel-based learning algorithms. IEEE Transactions on Neural Networks 12(2):181–201
24. Tsionas EG (2012) Maximum likelihood estimation of stochastic frontier models by the fourier transform. J Econ 170 (1):234–248
25. Bergstra J, Bardenet R, Bengio Y, Kgl B (2011) Algorithms for hyper-parameter optimization. In: Shawe-Taylor J, Zemel RS, Bartlett PL, Pereira FCN, Weinberger KQ (eds) NIPS, pp 2546–2554
26. Rodner E, Freytag A, Bodesheim P, Denzler J (2012) Large-scale gaussian process classification with flexible adaptive histogram kernels. In: Fitzgibbon AW, Lazebnik S, Perona P, Sato Y, Schmid C (eds) ECCV (4), ser. Lecture Notes in Computer Science, vol 7575. Springer, pp 85–98
27. Pace R, LeSage JP (2004) Chebyshev approximation of log-determinants of spatial weight matrices. Comput Stat Data Anal 45(2):179–196
28. Cai TT, Liang T, Zhou HH (2013) Law of log determinant of sample covariance matrix and optimal estimation of differential entropy for high-dimensional gaussian distributions. CoRR. arXiv:1309.0482
29. Quinonero-Candela J, Rasmussen CE (2005) A unifying view of sparse approximate gaussian process regression. J Mach Learn Res 6:1939–1959
30. de Baar J, Dwight R, Bijl H (2013) Speeding up kriging through fast estimation of the hyperparameters in the frequency-domain. Comput Geosci 54(0):99–106
31. Sollich P, Williams CKI (2004). In: Winkler J, Niranjan M, N. D. Lawrence (eds) Understanding gaussian process regression using the equivalent kernel. in Deterministic and Statistical Methods in Machine Learning, ser. Lecture Notes in Computer Science, vol 3635. Springer, pp 211–228. http://dblp.uni-trier.de/db/conf/dsmml/dsmml2004.html
32. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press. http://books.google.co.kr/books?id=mYm0bLd3fcoC
33. Robbins H, Monro S (1951) A stochastic approximation method, The annals of mathematical statistics:400–407
34. Robbins H, Siegmund D (1985) A convergence theorem for non negative almost supermartingales and some applications, in Herbert Robbins Selected Papers. Springer, pp 111–135

35. Yang C, Duraiswami R, Gumerov N, Davis L (2003) Improved fast gauss transform and efficient kernel density estimation. In: Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, pp.664–671 vol.1
36. Alecu TI, Voloshynovskiy S, Pun T (2005) The gaussian transform. In: EUSIPCO2005, 13th European Signal Processing Conference, pp.4–8
37. Greengard L, Strain J (1991) The fast gauss transform. SIAM J Sci Stat Comput 12(1):79–94
38. Yamamoto Y (2006) Efficient parallel implementation of a weather derivatives pricing algorithm based on the fast gauss transform. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, pp. 8
39. Spivak M, Veerapaneni SK, Greengard L (2010) The fast generalized gauss transform. SIAM J Sci Comput 32(5):3092–3107. doi:10.1137/100790744
40. Sampath RS, Sundar H, Veerapaneni SK (2010) Parallel fast gauss transform. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '10. Washington, DC, USA: IEEE Computer Society, pp. 1–10. doi:10.1109/SC.2010.39
41. Simoncini V, Szyld DB (2003) Theory of inexact krylov subspace methods and applications to scientific computing. SIAM J Sci Comput 25(2):454–477
42. Greengard L, Rokhlin V (1987) A fast algorithm for particle simulations. J Comp Physiol 73(2):325–348
43. Morariu VI, Srinivasan BV, Raykar VC, Duraiswami R, Davis LS (2008) Automatic online tuning for fast gaussian summation. In: Koller D, Schuurmans D, Bengio Y, Bottou L (eds) NIPS. Curran Associates, Inc, pp 1113–1120
44. Fangohr H (2004) A comparison of c, matlab, and python as teaching languages in engineering. In: Computational Science-ICCS 2004. Springer, pp 1210–1217
45. Wu X (1999) Performance Evaluation, Prediction and Visualization of Parallel Systems, ser. The International Series on Asian Studies in Computer and Information Science. Springer, US. http://books.google.co.kr/books?id=IJZt5H6R8OIC

Dinh-Mao Bui received the B.S. degree in Computer Science from the Computer Engineering Department at Ton Duc Thang University, Vietnam, in 2009 and the M.S. degree in Data Communication and Networking from the Posts and Telecommunications Institute of Technology, Vietnam, in 2012. He is now working toward the PhD degree in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests include Engineering Optimization, Stochastic Process, Parallel computing and Big Data.

Huu-Quoc Nguyen received the B.S. degree in Computer Software from the Computer Engineering Department at Posts and Telecommunications Institute of Technology, Vietnam, in 2014. He is now working toward the Master degree in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests include Cloud Computing, Thin Client, Authentication and Network.

YongIk Yoon is a Professor for Dept. of Multimedia Science in SookMyung Women's University, South Korea. He received M.S and Ph.d. degree from Computer Science of KAIST, in 1985 and 1994. He had researched for 15 years (1983–1997) like a member of senior Researcher of ETRI, in Korea. Also he had worked a visiting professor in University of Colorado at Denver, in USA, for three years (2004–2007). His Research Interests are smart services for future life, middleware for smart/future life, collaboration service model in mobile cloud environment, and intelligent mobile customer application platform. He is now a Member of ACM, IEEE, KIISE (Korean Institute of Information Scientists and Engineers), KIPS (Korean Information Processing Society), and OSIA (Open Standard and Internet Association).

SungIk Jun is a principal member of engineering staff for High-Performance System Research team in ETRI (Electronics Telecommunication Research Institute), South Korea. He received M.S degree from Computer Science of Chung-ang University, in 1987. He had researched for Real-time OS team during 15 years (1987–2001) like a member of senior Researcher of ETRI, in Korea. Also he had worked a team leader Wireless Security Application Research team for eight years (2001.3–2009.4). His research interests are Operating System, Wireless Security, and M2M for future life.

Muhammad Bilal Amin received his MS from DePaul University, Chicago, USA in 2006. He got his PhD. from Kyung Hee University, Korea in 2015. He is currently a Researcher and Team Lead of Cloud Computing Group at Ubiquitous Computing Lab, Department of Computer Engineering, Kyung Hee University, South Korea. He has a working experience of more than 10 years in software industry, working for Fortune 500 companies in USA. His research interests include, cloud computing, parallel programming, distributed systems, software architecture, semantic web, and performance-based ontology matching.

Sungyoung Lee received his Ph.D. degree in Computer Science from Illinois Institute of Technology (IIT), Chicago, Illinois, USA in 1991. He has been a professor in the Department of Computer Engineering, Kyung Hee University, Korea since 1993. Before joining Kyung Hee University, he was an assistant professor in the Department of Computer Science, Governors State University, Illinois, USA from 1992 to 1993. His current research focuses on Ubiquitous Computing, Cloud Computing, Intelligent Computing and eHealth.