Parallel Kriging Analysis for Large Spatial Datasets

Wei Zhuo*, Prabhat‡, Chris Paciorek†, Cari Kaufman† and Wes Bethel‡
*Georgia Institute of Technology
wzhuo3@cc.gatech.edu
†University of California, Berkeley
paciorek, cgk@stat.berkeley.edu
‡Lawrence Berkeley National Lab
Prabhat, ewbethel@lbl.gov

***Abstract*—We investigate the problem of kriging analysis for estimating quantities at unknown locations given a set of observations. Widely known in the geostatistical community, kriging bases spatial prediction on a closed-form model for the spatial covariances between observations, deriving interpolation parameters that minimize variance.**

While kriging produces predictions with high accuracy, a standard implementation based on maximum likelihood involves repeated covariance factorization, forward-solve, and inner product operations. The resulting computational complexity renders the method infeasible for application to large datasets on a single node. To facilitate large-scale kriging analysis, we develop and implement a distributed version of the algorithm that can utilize multiple computational nodes as well as multiple cores on a single node.

We apply kriging analysis for making predictions from a medium-sized weather station dataset, and demonstrate our parallel implementation on a much larger synthetic dataset consisting of 65536 points using 512 cores.

***Keywords*-Spatial data estimation, Kriging analysis, Parallel algorithms**

I. Introduction

The field of spatial statistics provides a range of important statistical models for analyzing spatially-indexed data. These techniques can be used for the characterization of the spatial structure of environmental variables, distinguishing signal from noise, and prediction at locations without measurements. With observational datasets, such as weather station readings or satellite images from remote sensing, one may want to predict at arbitrary locations or extrapolate to regular grids for model comparison, numerical simulation, and visualization. In all of these cases, it is important to fit the data using a spatial model that can borrow strength from available observations in a way that makes use of the spatial structure of the data. It is equally important to calculate uncertainties associated with these predictions.

Kriging is a powerful statistical technique which can be applied to all of these problems [4] [6]. The calculations involved in kriging lie at the core of advanced models which are used to analyze a wide variety of spatial as well as spatio-temporal data [6]. This broad class of methods is widely used for analyzing climate data and other environmental datasets [12], and the calculations also lie at the core of statistical approaches to uncertainty quantification [9]. Kriging can also be viewed as estimation based on an underlying Gaussian process representation of the unknown spatial field; this construct tends to be used in many problems in machine learning. Therefore, parallel frameworks that allow large scale kriging analysis are potentially applicable to a broad suite of statistical and machine learning methods based on Gaussian process methodology.

Modern approaches to kriging maximize the likelihood of the observations to obtain estimates of model parameters that determine predictions under the kriging approach. The likelihood maximization step in kriging relies heavily on matrix operations such as factorization and forward-solve. The covariance matrices used in these operations represent the pairwise relationships among training locations, hence they are of $O(N^2)$ storage complexity, where N refers to the number of training data points. The computational complexity is dominated by the $O(N^3)$ cost of dense cholesky factorization of these matrices. Due to the lack of distributed implementations, researchers are typically limited to processing matrices that can fit in memory on a single workstation. This problem is exacerbated by the ever-increasing size of both simulation and observational datasets that are now becoming available. For instance, climate simulations routinely produce datasets with $O(1M)$ points. Consequently, spatial statisticians adopt approximation techniques [11] [8] that consider subsets of training data local to the query locations, thereby losing fine-grained information of scientific interest. The goal of this paper is to enable exact kriging on large datasets by designing and implementing a high performance, parallel version of kriging.

We address the following challenges in this paper:

- How do we design a framework for enabling parallel kriging computations?
- How do we utilize distributed memory nodes for tackling large scale datasets?
- How do we utilize modern multi-core architectures?
- What are the performance and scalability for such a framework?

In this paper, we first present an expository account of kriging for the benefit of readers who are not familiar with the technique. We contrast this approach to possibly more familiar techniques such as inverse distance-weighted interpolation and least squares regression [10][1]. We then introduce our parallel algorithm for kriging analysis. We apply the implementation to a precipitation anomaly dataset for illustrative purposes, and conduct weak scaling experiments on upto 512 cores for a synthetic dataset with 65,536 entries. Finally, we conclude with some thoughts on limitations and future work.

II. Kriging Model

Let us consider the set of N training locations $\{S_i\}$, with a measurement available at each location S_i, denoted by $y(S_i)$. The process of estimating the target function y at unknown locations uses the following two steps:

1) *Interpolation*: which constructs the estimate as a weighted sum of training observations. Usually, the weight is a function of distances between the training locations and the query location and contains unknown parameters that must be estimated.
2) *Parameter estimation*: this estimates the parameters by minimizing a objective function O that reflects the discrepancies between the training data and estimates at the training locations.

In absence of a statistical model, there could be many ways to interpolate the target function. For illustrative purposes, we will consider the inverse distance weighted interpolation technique that has been well developed in the machine learning community [10] [1].

A. Inverse Distance Weighted Estimation

Given a query location S_q, the inverse distance-weighted interpolation approximates the target function $y(S_q)$ by the following linear equation [10]:

$$\hat{y}_{idw}(S_q) = \sum w_i y(S_i) \tag{1}$$

The interpolation coefficients are of the form $w_i = \lambda(\frac{1}{d(S_i,S_q)})^\rho$, where λ is the inverse of the sum of all coefficients, and ρ is a positive weighting power, meant to weight the contribution of a observation by its closeness to the query location. In other words, the inductive bias in distance-weighted interpolation is that the contribution of a observation is inversely proportional to a power function of its distance from the query location.

In the estimation step, we choose ρ and λ that minimize the squared error summed over the set of training examples

$$E = \sum \left(y(S_i) - \sum_{i \neq j} \lambda(\frac{1}{d(S_i,S_j)})^\rho y_j \right)^2 .$$

The objective function that we are trying to minimize is as follows:

$$O_{idw} = (Y - WY)^T (Y - WY)$$

where Y is the column vector of measurements at training locations and W is the inverse distance matrix

$$W_{ij} = \begin{cases} \lambda(\frac{1}{d(S_i,S_j)})^\rho & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases}$$

B. Kriging Estimation

The Kriging model approximates the target function by linear interpolation with a different form:

$$\hat{y}(S_q) = g(S_q) + \sum k_i (y(S_i) - g(S_i))$$

where g represents the existing knowledge of the spatial field and is generally either a constant or linearly varying over space. We consider a specific example where the observation y is the anomaly measured from the ground truth, therefore g vanishes. The interpolation equation can be further written as:

$$\hat{y}_{krig}(S_q) = \sum k_i y(S_i) \tag{2}$$

Kriging interpolation minimizes the mean squared error of prediction [4]; the interpolator can be shown to correspond to $K = \Omega\Sigma^{-1}$, where K is the row vector representation of $\{k_i\}$, Ω is the covariance vector between the query and the training data, and Σ is the covariance matrix of the N training data points. There are a variety of covariance models that are used; two common ones involve exponential and squared exponential decay with respect to the distance between observations. Predictions can be written in terms of the covariance between observations, $\Sigma_{ij} = \lambda k_\rho(d(S_i, S_j))$, where k_ρ is the radial basis function corresponding to the given covariance model[1], with parameter(s) ρ that need to be estimated. Note that this approach assumes that nearby observations share more similarities than observations that are far apart.

In kriging, we choose λ and ρ that maximize the likelihood of the training data:

$$P = \frac{1}{(2\pi)^{\frac{N}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2} Y^T \Sigma^{-1} Y}$$

The process is the same as minimizing the negative log-likelihood, written as the objective function O_{krig}:

$$O_{krig} = log|\Sigma| + Y^T \Sigma^{-1} Y \tag{3}$$

A example of kriging and inverse distance-weighted estimation is shown in Section IV-A.

[1]A radial basis function $k_\rho(d)$ decreases from 1 to 0 as d increases. For example, $k_\rho(d)$ can be $exp(-d/\rho)$ (exponential), or $exp(-d^2/\rho^2)$ (Gaussian), or $1 - \frac{3d}{2\rho} + \frac{d^3}{2\rho^3}$ (spherical)

C. Steps involved in performing Kriging analysis

The first step of kriging analysis is to construct the symmetric $N \times N$ distance matrix from training locations:

$$D_{ij} = d(S_i, S_j) \tag{4}$$

Then we calculate the covariance matrix Σ given λ, ρ as follows

$$\Sigma_{ij} = \lambda k_\rho(D_{ij}) \tag{5}$$

To compute the objective function in Eq.3 the following linear algebra operations are performed:

1) *Cholesky decomposition* computes the lower triangular matrix L by factorizing Σ:

$$LL^T = \Sigma \tag{6}$$

2) *forwardsolve* solves for X with L as the coefficient matrix for a system of linear equations and Y as the constant term:

$$X = L^{-1}Y \tag{7}$$

3) the last routine computes the *inner product* of X and the *log sum* of L's diagonal elements:

$$Q = X^T X \tag{8}$$

$$log(\mid L \mid) = \sum_{i=1}^{N} log(L_i). \tag{9}$$

Finally the objective function is simply the sum $2log|\Sigma|+Q$.

In summary, our kriging analysis implementation consists of the following key phases:

- *Data preparation*: initialization of training locations S and observations Y, construction of distance matrix (Eq.4)
- *Objective evaluation*: includes calculation of the covariance matrix (Equation 5), Cholesky factorization (Equation 6), forward solve (Equation 7), inner product (Equation 8), diagonal log sum (Equation 9) in sequence.
- *Objective function minimization*: this is equivalent to likelihood maximization, with the above procedure carried out iteratively for varying λ, ρ, μ in a optimizer, where the gradient and Hessian of the objective function is evaluated.
- *Kriging interpolation and uncertainty calculation*: this phase consists of a set of linear algebra routines similar to objective evaluation. This step constructs distance and covariance matrix from query and training locations similar to Eq. 4 and 5. The uncertainty calculation consists of factorization and forward solve similar to Eq. 6 and 7.

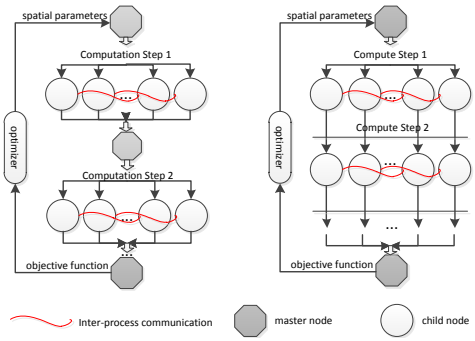


Figure 1. Left figure shows the "distribute-compute-collect" mode. Right figure shows our approach that allows local reduction.

III. Parallel Implementation

As the number of spatial data points increase, the associated $N \times N$ covariance and distance matrices become quadratically larger. In practice we have observed that exact likelihood computation becomes infeasible for more than $\sim$10K spatial locations, even on some high-end workstations. This motivated us to perform all operations involved in evaluating the objective function in a fully distributed fashion. In our implementation, child nodes perform all steps of objective evaluation: construction of covariance matrix, factorization, forward solve, inner product and diagonal sum, in parallel. Master node controls the optimization: it pulls local results in order to evaluate the gradient and Hessian of the objective function and updates the learned spatial parameters. Our design prevents the memory bottleneck; we explicitly avoid situations wherein the master node processes a large matrix and distributes the data to child processes. For instance, the master process is not tasked with covariance construction or collection of intermediate factorization results. This is key difference between our implementation and RScaLAPACK [15], which farms out data for processing to child processes and collects the results back on a single node. We illustrate this key difference in Fig. 1.

A. Dependency Graphs

Our R implementation breaks matrices and vectors into roughly even sized pieces for individual computational task. Once the matrix pieces are distributed across nodes, the factorization and forward solve tasks now begin to compute local result and communicate with each other at different points in time. This is illustrated in Figure 2, which shows the dependency graph of the flow of data. On a distributed memory architecture, we need to explicitly move the data using MPI. We note that our distributed algorithm for the covariance factorization step is not novel, ScaLAPACK [14] [5] implements similar algorithms. Our distributed algorithm for factorization is motivated by previous literature on distributed Cholesky factorization [2] [7].

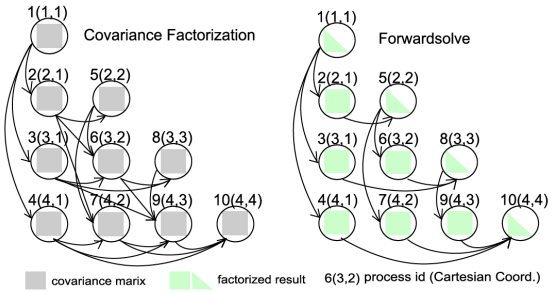


Figure 2. Dependency Graph for computational tasks during factorization and forward solve.

An important difference between our software framework, and accelerated linear algebra kernels provided by libraries such as ScaLAPACK, is that we are able to do an end-to-end analysis of our specific use case (i.e. kriging analysis operations being composed of a factorization followed by a forward solve) to optimize the execution of the distributed algorithm. We use classic parallel programming optimizations such as local reduction and pipelining across these heterogeneous linear algebra computations to overlap the computation and communication tasks effectively. We identify that ScaLAPACK is constrained to optimizing individual tasks (i.e. factorize, forward solve, etc), hence cannot compose a sequence of computations in an optimal fashion.

We use Rmpi [13] for interprocess communication for explicitly moving data in accordance with the dependency graph. For utilizing multi-core processor architectures, we link R to a threaded Intel Math Kernel Library (MKL) implementation of BLAS, which is optimized for Intel processors [16]. The highly tuned MKL implementation takes into account various aspects of shared-memory parallel computations (i.e. multiple cache hierarchies, NUMA layout, memory bandwidth, etc). Therefore, our software is able to fully utilize all available cores on a single node in a hybrid parallel fashion. We note that our software design can enable us to utilize heterogeneous computing hardware in the future, i.e. by utilizing CUDA BLAS for graphics accelerator hardware.

B. Node-level parallel algorithms

Alg. 1 and Alg. 2 are pseudo codes for our node-level parallel algorithms for factorization and forwardsolve. We have shown pictorial representations of these algorithms in Fig. 2, where 10 tasks operate on a generic square matrix. The tasks in a column are cyclic distributed to child nodes so as to allow load balancing.

We note that these two computational steps have the most complex dependency relationships as compared to computing the inner product and the diagonal sum. This is because that once the factorization result L and the solution vector X are available, following steps can be computed by the diagonal processes in a fully parallel fashion with minimal interprocess communication. We now focus on explaining the distributed algorithms for factorization and forwardsolve in detail.

Before the factorization step, we assume the submatrix C_{ij} is available on a child node. Here C_{ij} represents a submatrix of C of size $\frac{N}{P} \times \frac{N}{P}$, where P is the partition number, i, j are indices from 1 to P. Since the covariance matrix is symmetric, we only need to store the lower half of the matrix, hence, the row index i is always greater than the column index j in our construction.

When factorization is completed, child nodes store the local factorization result L_{ij}, the submatrix of the global factorization result L. The master process does not collect this intermediate result. L_{ij} serves as the input to the distributed forwardsolve, as shown in Alg. 2, where Y_i is the sub-vector of observations Y. After distributed forwardsolve, Task (i, j) holds the i-th sub-vector X_i of the solution vector X.

Alg. 1 and 2 represent the core tasks in kriging regression as mentioned in Section II-C. The pseudocode is executed at a task-level by our implementation. We note that implementations for key kernels *chol, forwardsolve, crossprod* are provided by the MKL implementation of the BLAS interface.

Alg. 1 Distributed Factorization for Task(i, j): $C \Rightarrow LL^T$

Input: C_{ij}
Output: L_{ij}
  $L_{ij} := C_{ij}$
  if $i = j$ **then**
    for k $= 1 : j - 1$ **do**
      inmsg:= recv from Task(i, k)
      L_{ij}:= L_{ij} - crossprod (inmsg)
    end for
    L_{ij} = chol(L_{ij})
    broadcast L_{ij} to Task(k, j), $k \in [i + 1, P]$
  end if
  if $i > j$ **then**
    for k $= 1 : j - 1$ **do**
      inmsg1 := recv from Task(j, k)
      inmsg2 := recv from Task(i, k)
      L_{ij}:= L_{ij} - crossprod (inmsg1, inmsg2)
    end for
    inmsg:= recv from Task(i, i)
    L_{ij}:=forwardsolve(inmsg, L_{ij})
    broadcast L_{ij} to Task(i, k), $k \in [j + 1, i]$
    broadcast L_{ij} to Task(k, j), $k \in [i + 1, P]$
  end if

IV. Results

A. Predicting precipitation anomalies

We first show a real-life application of kriging to a scientific problem. We consider training data consisting of

Alg. 2 Distributed Forwardsolve at Task(i, j): $X \Leftarrow L^{-1}Z$

Input: Y_i
Output: X_i
  if $i = j$ **then**
    for $k = 1 : i - 1$ **do**
      inmsg:= recv from Task(i, k)
      $Y_i := Y_i - inmsg$
    end for
    X_i = forwardsolve(L_{ij}, Y_i)
    broadcast X_i to Task(k, i), k $\in$ [i+1, P]
  end if
  if $i > j$ **then**
    X_i:= recv from Task(j, j)
    outmsg:= crossprod(L_{ij}, X_i)
    send outmsg to Task(i, i)
  end if

yearly precipitation anomalies (measured from the mean values) by 7352 weather stations across US (see Fig. 3(a)). Our goal is to make predictions at new spatial locations.

We assume that the covariance between two observations is a exponential radial basis function (presented in Section II). To establish a baseline, we also apply the inverse-distance-weighted method to the same problem, in which the weighting power ρ is chosen to minimize the squared error summed over all training data. We refer to this non-statistical method as *optimal-IDW*.

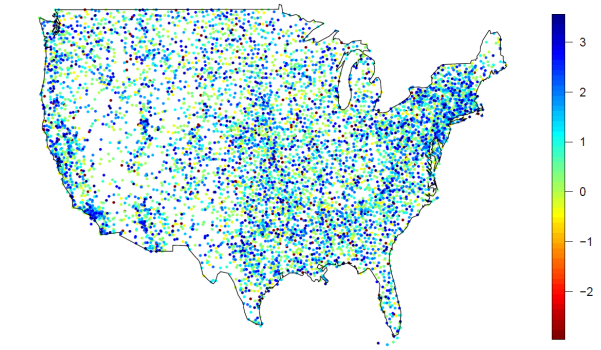
We show the estimation results of kriging and optimal-IDW for prediction values at 30 randomly chosen query points (distinct from the training locations). The resulting predictions are plotted in Fig.3(b), where the annotation on the x-axis specifies the longitude and latitude of each query location. The plot is meant to provide an illustration of the range of errors produced by both techniques. More quantitatively, we report a RMSE of 2.79 for kriging and 3.45 for optimal-IDW over the 30 query locations. The results indicate that kriging works better than optimal-IDW for this test.

B. Parallel Performance

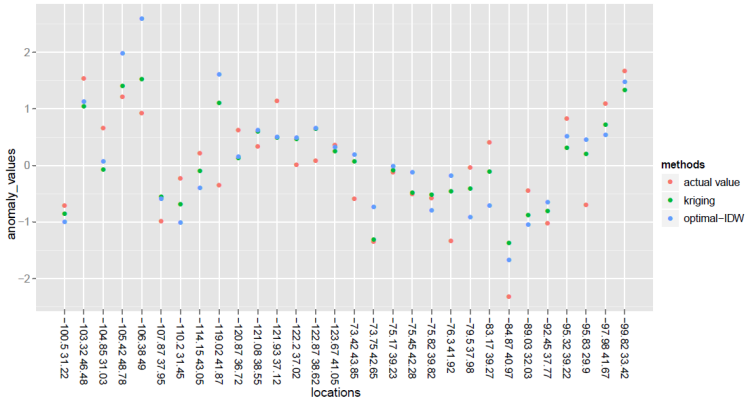
We now present the parallel performance on a much larger dataset, as one of the goals of this paper is to present a parallel processing framework for kriging analysis.

1) Dataset: To put our parallel implementation to the test, we randomly generated $65,536$ spatial data values on a 256×256 regular grid, as shown in Fig. 4(a). Our performance results are based on this synthetic dataset.

2) Hardware: All of our profiling experiments were conducted on Carver, an IBM iDataplex system at NERSC. Each Carver node consists of 2 quad-core Intel Nehalem 2.67GHz processors, 10GB of usable memory and Infiniband 4X QDR interconnect.



(a) Precipitation anomalies from 1962 at US weather stations.

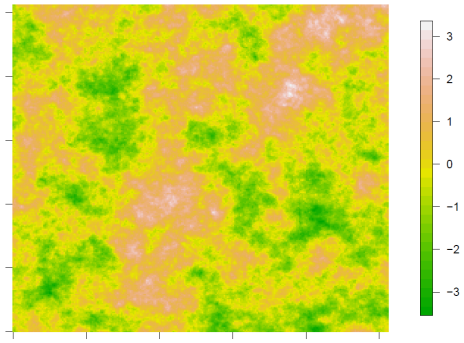


(b) A test involving both kriging and optimal-IDW for 30 query locations from the anomaly dataset.

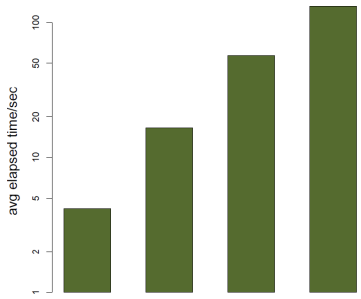
Figure 3. Estimating precipitation anomalies

3) Multi-core Performance on single node: We started our experiments with a baseline single-threaded program in R to perform kriging computations. The average time for the objective evaluation step for 8k data points is 29.3 seconds. We then linked our implementation to the threaded-MKL BLAS implementation. The runtime of objective evaluation reduced to 4.2 secs. Specifically, the time spend on covariance construction, factorization, forwardsolve are 0.26s, 2.8s, 0.96s respectively. Hence, the multi-threaded implementation provided us with a speedup factor of 7.0 with respect to the single-threaded implementation on a single Carver node.

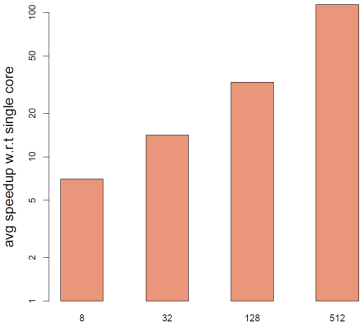
4) Distributed Performance: We now design a weak scaling experiment to test the performance of our node-level parallel implementation linked with threaded MKL, For this experiment, we vary the number of spatial data points across 8K, 16K, 32K and 64K points. Since the resulting covariance matrices are quadratically related to number of points N, this results in problems with 64M, 256M, 1024M and 4096M double-precision entries. We run these problems on 8, 32, 128 and 512 cores respectively and report the elapsed time. Recall that weak scaling experiments involve scaling the size



(a) 64k randomly generated spatial data on regular grid



(b) weak-scaling performance



(c) speedup ratio

Figure 4. Performance profiling

of the dataset (in this case the matrix size) proportionally with the number of computational cores. In other words, we keep $\frac{N^2}{n}$ constant, where n is the number of cores.

Figure 4(b) shows results from our weak scaling study. Overall, we observe a linear relationship between the elapsed time t_n and the number of cores n. This can be explained with the following series of observations: the computational complexity is dominated by an $O(N^3)$ term corresponding to factorization. t_1, the sequential running time is proportional to $O(N^3)$; n is proportional to $O(N^2)$ from the weak scaling design. t_n, the elapsed time is proportional to $\frac{t_1}{n}$, which turns out to be $O(N)$ in our case. Consequently, the elapsed time increases linearly as we make the problem size larger.

Figure 4(c) shows the speedup $S_n = \frac{t_1}{t_n}$ of our parallel implementation. Our results indicate that we can get fairly substantial performance improvement (≈ 120) over a single core for a substantially larger problem. We believe that our results are a positive step forward for applications that involve large scale kriging on massive spatial datasets.

V. Conclusion

In this paper, we explore kriging analysis for spatial prediction at query locations given a set of observations. Kriging provides estimates with high accuracy, however it involves computationally demanding matrix operations that limit the technique's applicability to large spatial datasets. We develop and implement parallel framework for improving the scalability and applicability of kriging analysis. Our framework performs all matrix operations in a fully parallel fashion. The implementation utilizes distributed memory nodes, as well as multi-core CPUs on a single node. We demonstrate the application of our software on a real-world scientific problem, and report weak scaling results on up to 64kx64k covariance data and 512 cores.

As a final note, statisticians routinely use a wide variety of methods based on kriging that rely on composition of the same core calculations considered in this paper. Therefore, our framework can be applied to various extension of kriging and and other Gaussian process-based methodologies (e.g., [3], [6]) which are now widely-used in estimating spatial and spatio-temporal data.

References

[1] Roger S. Bivand, Edzer J. Pebesma, and V. Gmez-Rubio, *Applied Spatial Data Analysis with R*, Springer, 2008.

[2] Ballard, Grey and Demmel, James and Holtz, Olga and Schwartz, Oded, *Communication-optimal parallel and sequential Cholesky decomposition*, Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures, 2009.

[3] Banerjee, S. and B.P. Carlin and A.E. Gelfand, *Hierarchical Modeling and Analysis for Spatial Data*, Chapman & Hall, 2004.

[4] N. Cressie, *Statistics for Spatial Data*, Wiley, 1993.

[5] K. Dackland and E. Elmroth. *Parallel block matrix factorizations for distributed memory* multicomputers, 1992.

[6] A. E. Gelfand, *Handbook of Spatial Statistics*, Taylor and Francis, 2010.

[7] Gustavson, Fred G. and Karlsson, Lars and Kågström, Bo, *Distributed SBP Cholesky factorization algorithms with near-optimal scheduling*, ACM Trans. Math. Softw., 2009.

[8] Cari Kaufman, *Covariance Tapering for Likelihood-Based Estimation in Large Spatial Data Sets*, Journal of American Statistical Association, 2008.

[9] M.C. Kennedy, and A. O'Hagan, *Bayesian calibration of computer models*, Journal of the Royal Statistical Society Series B, 63, 2001.

[10] T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[11] Michael L. Stein, Zhiyi Chi, Leah J. Welty, *Approximating likelihoods for large spatial data sets*, *Journal of the Royal Statistical Society Series B*, 2004.

[12] M.P. Tingley, and P. Huybers, *A Bayesian algorithm for reconstructing climate anomalies in space and time. Part I: Development and applications to paleoclimate reconstruction problems*, Journal of Climate, 23, 2010.

[13] Rmpi: R wrapper for mpi, http://www.stats.uwo.ca/faculty/yu/Rmpi/

[14] ScaLAPACK, http://www.netlib.org/scalapack/

[15] RScaLAPACK: ScaLAPACK for R, http://cran.r-project.org/web/packages/RScaLAPACK/

[16] Intel math kernel library, http://software.intel.com/en-us/articles/intel-mkl/