# Distributed Likelihoods Computation for Large Spatial Datasets

Wei Zhuo[†] Prabhat[‡] Cari Kaufman[§] Chris Paciorek[§]

[†]Georgia Institute of Technology [‡]Lawrence Berkeley National Laboratory [§]UC Berkeley, Dept. of Statistics

## Abstract

We investigate the problem of fitting geospatial models to large spatial datasets. The process of fitting a model involves efficient computation of likelihoods. An exact solution of the problem for $n$ observations requires computing the determinant and inverse of the $n \times n$ covariance matrix, which can be expensive for large n. We examine two modes of parallelization to overcome these limitations: multi-threaded (within single node) and distributed (across multiple nodes).

On a single node, we used the multi-threaded BLAS implementation to achieve significant performance gain over the single threaded implementation.

For a cluster of compute nodes, we implemented a distributed likelihood algorithm using RMPI. The resulting computation utilized all available cores on a single node, as well as multiple nodes on the cluster.

## Introduction

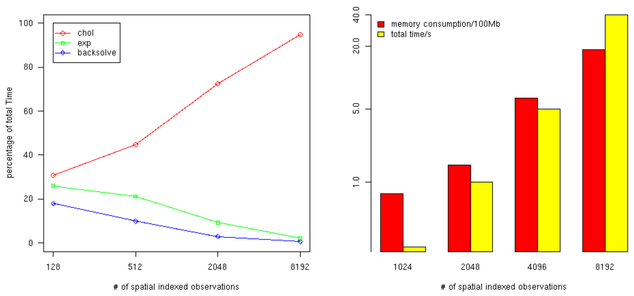
- Traditional geostatistical model assumes the covariance between two spatially indexed observations $Z_i$ and $Z_j$ at location $S_i$, $S_j$ has the following simplified form

$$\Sigma[i,j] = \kappa e^{-|S_i - S_j|/\rho} \quad (1)$$

- from which the joint likelihood of $n$ observations is

$$2\pi^{-n/2} \mid \Sigma \mid^{-1/2} e^{-\frac{1}{2}(Z-\beta)^T \Sigma^{-1}(Z-\beta)} \quad (2)$$

- Current statistical methods use EM algorithm to evaluate important spatial constants in the covariance model, which iteratively compute the joint likelihood up to thousands of times.

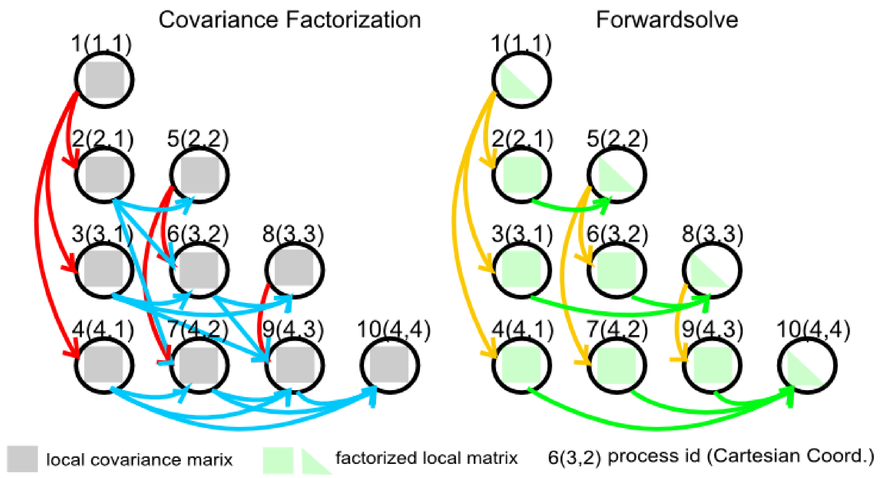



Major steps in the EM algorithm/likelihood computation on a single node are

- *exp*: generating pairwise covariances
- *chol*: factorizing the covariance matrix ($\Sigma = LL^T$)
- *forwardsolve*: solving a lower triangular linear system ($L^{-1}(Z - \beta)$)

The growing trend indicates that computing the joint probability of 65k spatially indexed data would take at least 4 hours, and evaluating spatial constants would easily take several months!

## Algorithm Overview




- **Data Distribution**: The master process distributes the observation $Z$ and location $S$ data to a group of slave process. Each process computes a local covariance matrix from the local location vector. Only the lower half of the global covariance matrix is actually constructed by the slave processes.
- **Covariance Factorization ($\Sigma = LL^T$)**: Each slave process locally performs one of the *matrix-matrix* operations *Cholesky*, *Forwardsolve* and *Crossproduct* based on received messages and its cartesian coordinate. After the factorization, processes residing on the diagonal compute their local determinant and report it to the master.
- **Global Forwardsolve ($X = L^{-1}(Z - \beta)$)**: Each slave process locally performs one of the *matrix-vector* operations *Forwardsolve* and *Crossproduct*. Then the processes residing on the diagonal compute the local crossproducts of their solution vectors, and report the result to the master.
- **Data Collection** The master process calculates the determinant $\mid \Sigma \mid$ by multiplying every local determinants collected in the Covariance Factorization step, and calculates the crossproduct $\mid X^T X \mid$ by adding the local values collected in Global Forwardsolve.

## Analysis of Communication Cost

Our framework for spatial likelihood computation has the following advantages over the ScaLAPACK implementation.

- eliminate the need for large memory node by distributing the covariance matrix construction to multiple nodes.
- reduce the communication between the slave processes and the master by performing local reduction on the diagonal processes.

Following table summarized the IPC cost, assuming $P(P+1)/2$ slave processes

| | Data Distribution | Factorization | Forwardsolve | Data Collection |
|---|---|---|---|---|
| msg count | $(3P^2+3P)/2$ | $(P^3-P)/2$ | $P^2-P$ | $2P$ |
| msg type | vector | matrix | vector | scalar |

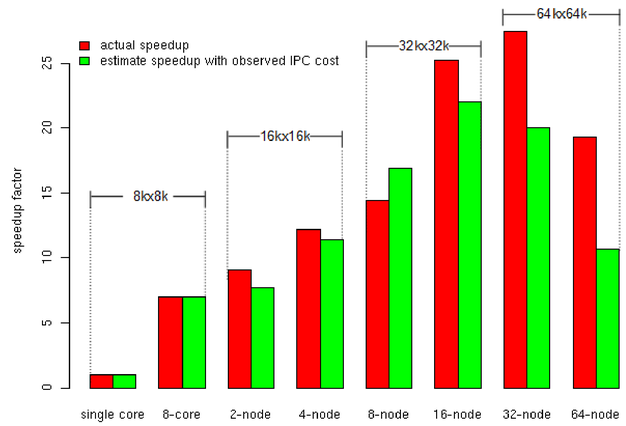
During the experiment, we measure the time required for initial data distribution and estimate the total communication cost, from which we derive the estimated speedup with IPC cost.

Acknowledgment

- This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## Experimental Results

- We run the experiment on Carver, an IBM iDataplex system at NERSC. Each Carver node consists of 2 quad-core Intel Nehalem 2.67GHz processors.
- We show scaling performance of our distributed code from 1 to 64 compute nodes as well as the speedup between single-core and multi-core.
- The speedup is approximately 7 with multi-thread implementation on an single Carver node, and an overall speedup of 28 on a cluster of 32 compute nodes.
- The largest covariance matrix we factorized is $64k \times 64k$, which contains $4B$ nonzero doubles. The best wall-clock time for factorizing $32k \times 32k$ and $64k \times 64k$ covariance matrices are 100s and 474s respectively.




We plot the estimate speedup that takes into account the communication cost. The observed results generally agree with the estimation.

## Implementation Issues

- *Dynamic Scheduling*: There is no strict order on independent message updates. This dynamic scheduling is achieved by coloring messages according their types in stead of their source coordinates.
- *Load Balancing*: We use distribution blocking. In particular, processes within the same column are distributed to different nodes to improve concurrency.

## Conclusion

- We have successfully factorized $64k$x$64k$ covariance matrix for likelihood computation. These results are very encouraging for researchers in spatial statistics who can now perform large scale computations.
- We have Demonstrated a total speedup factor of 28 with respect to a single core, when running on a cluster of 32 multi-core nodes.
- As expected, the interprocess communication cost is the major bottleneck in improving scaling performance of the implementation.