A Parallel Formulation of the Spatial Auto-Regression Model for Mining Large Geo-Spatial Datasets

Baris M. Kazar, Shashi Shekhar, David J. Lilja, Daniel Boley
Electrical and Computer Engineering and Computer Science Department
University of Minnesota, Twin-Cities MN 55455
{kazar,shekhar,lilja,boley}@cs.umn.edu

Abstract: The spatial auto-regression model (SAM) is a popular spatial data mining technique which has been used in many applications with geo-spatial datasets. However, serial procedures for estimating SAM parameters are computationally expensive due to the need to compute all the eigenvalues of a very large matrix. We propose a parallel formulation of the SAM parameter estimation procedure in this paper using data parallelism and hybrid programming technique. Experimental results on an IBM Regatta show that the proposed parallel formulation achieves a speedup of up to 7 on 8 processors. We are developing algebraic cost models to analyze the experimental results to further improve the speedups.

Keywords: Spatial Auto-regression, Spatial Auto-correlation, Parallel Formulation, Spatial Data Mining

1. Introduction

Explosive growth in the size of spatial databases has highlighted the need of spatial data mining techniques for mining the interesting but implicit spatial patterns within these large databases. Extracting useful and interesting patterns from massive geo-spatial datasets is important for many application domains such as regional economics, ecology and environmental management, public safety, transportation, public health, business, and travel and tourism [2,13,14].

Many classical data mining algorithms such as linear regression assume that the learning samples are independently and identically distributed (IID). This assumption is violated in the case of spatial data due to spatial auto-correlation [14] and classical linear regression yields a weak model with not only low prediction accuracy [13] but also residual error exhibiting spatial dependence. The spatial auto-regression model (SAM) [5,14] is a generalization of the linear regression model to account for spatial auto-correlation. It has been successfully used to analyze spatial datasets in regional economics, ecology [2,13], etc. The model yields better classification and prediction accuracy [2,13] for many spatial datasets exhibiting strong spatial auto-correlation.

However, it is computationally expensive to estimate the parameters of SAM. For example, it can take an hour of computation for a spatial dataset with 10000 points. This has limited the use of SAM to small problems, despite its promise to improve classification and prediction accuracy for larger spatial datasets. Parallel processing is a promising approach to speedup the sequential solution procedure for SAM and this paper focuses on this approach.

The only related work [9] implemented the SAM solution for one-dimensional geo-spaces and used CMSSL [4], a parallel linear algebra library written in CM-Fortran (CMF) for the CM-5 supercomputers of Thinking Machines Corporation, both of which are not available for use anymore. This work was not useful for spatial datasets embedded in spaces of dimensionality two or more.

We propose a parallel formulation for a general exact estimation procedure [6] for SAM parameters that can be used for spatial datasets embedded in multi-dimensional space. We use public domain parallel numerical analysis library to implement steps of serial solution. In addition, we modify the source code of the library to change scheduling, data-partitioning etc. to tune the performance.

We evaluate the proposed parallel formulation on an IBM Regatta. Results of experiments show that the proposed parallel formulation achieves a speedup of up to 7 on 8 processors within a single node of the IBM Regatta. We compare different load-balancing techniques supported by OpenMP [1] for improving the speedup of the proposed parallel formulation of SAM. Affinity scheduling performs best on average. We also evaluate the impact of other OpenMP parameters, i.e. chunk size.

We plan to expand the experimental studies to include a larger number of processors via hybrid parallel programming, and other parameters e.g. degree of auto-correlation. We also plan to develop algebraic cost models to characterize the scalability and understand the performance bottlenecks to further improve speedup.

In the long run, we want to develop parallel formulations for approximate solution procedures for SAM that exploit sparse matrix techniques [8].

Scope: This paper covers parallel formulation for a dense and exact solution to SAM. The parallel SAM solution is portable and can run on today’s distributed shared-memory architecture supercomputers such as IBM Regatta, IBM SP, SGI Origin, SGI Altix, and Cray X1. We do not address non-parallel approaches, e.g. sparse matrix techniques, approximation techniques to speedup the sequential solutions to SAM.

The remainder of the paper is organized as follows: Section 2 presents the problem statement and explains...
the serial exact algorithm for the SAM solution. Section 3 discusses our parallel formulation. Experimental results are presented in Section 4. Finally, Section 5 summarizes and concludes the paper with a discussion of future work.

2. Problem Statement and Serial Exact SAM Solution

We first present the problem statement and then discuss the serial exact SAM solution based on the maximum-likelihood (ML) theory [6].

The problem studied in this paper is defined as follows: Given the serial solution procedure called the "Serial Dense Matrix Approach" described in [9], we need to find a parallel formulation for multi-dimensional geo-spaces to reduce the response time. The constraints are as follows: the spatial auto-regression parameter, \( \rho \), varies in the range \([0,1)\); the error is normally distributed, i.e. \( \varepsilon \sim N(0,\sigma^2) \) IID; the input spatial dataset is composed of normally distributed random numbers with unit standard deviation and zero mean; the parallel platform is composed of an IBM Regatta, OpenMP and MPI; and the size of the neighborhood matrix \( W \) is \( n \). The objective is to implement parallel and portable software whose scalability is evaluated analytically and experimentally.

A spatial auto-correlation term \( \rho y \) is added to the linear regression model in order to model the strength of spatial dependencies among the elements of the dependent variable, \( y \). The resulting equation – which accounts for the spatial relationships – is shown in equation 1 and is called the spatial auto-regression model [5].

\[
y = \rho Wy + \beta \varepsilon
\]

**Eqn 1.** Spatial auto-regression model (SAM)

where \( \rho \) is the spatial auto-regression (auto-correlation) parameter, \( y \) is an \( n \)-by-1 vector of observations on the dependent variable, \( x \) is an \( n \)-by-\( k \) matrix of observations on the explanatory variable, \( W \) is the \( n \)-by-\( n \) neighborhood matrix that accounts for the spatial relationships (dependencies) among the spatial data, \( \beta \) is an \( k \)-by-1 vector of regression coefficients, \( \varepsilon \) is an \( n \)-by-1 vector of unobservable error.

Figure 1. System diagram of the serial exact algorithm for the SAM solution composed of three stages i.e. Stage A, B, and C. Stage A is further composed of three sub-stages (pre-processing, Householder transformation and QL transformation) which are not shown.

Figure 1 highlights the stages of serial exact algorithm for the SAM solution. It is based on maximum-likelihood (ML) theory which requires computing the logarithm of the determinant of large \((1-\rho W)\) matrix. The derivation of the ML theory will not be shown here due to limited space. However, the first term of the end-result of the derivation i.e. equation 2 clearly shows why we need to compute (natural) logarithm of determinant of a large matrix. The \( n \)-by-\( n \) identity matrix is denoted by "I", the transpose operator is denoted by "\( T \)", "ln" denotes logarithm operator and \( \sigma^2 \) is the common variance of the error in equation 2.

\[
\begin{align*}
\ln(L) &= \ln \| I - \rho W \| - \frac{n \ln(2\pi)}{2} - \frac{n \ln(\sigma^2)}{2} - SSE \\
where \\
SSE &= \frac{1}{2\sigma^2} \left\{ y^T (1 - \rho W)^T (1 - x(x^T x)^{-1} x^T) (1 - x(x^T x)^{-1} x^T) [1 - x(x^T x)^{-1} x^T] (1 - \rho W)y \right\}
\end{align*}
\]

**Eqn 2.** The end-result of ML theory

Therefore, Figure 1 can be viewed as an implementation of the ML theory. This section describes each stage. Stage A is composed of three sub-stages: pre-processing, Householder transformation [11], and QL transformation [3]. The pre-processing sub-stage not only forms the row-standardized neighborhood matrix \( W \), but also converts it to its symmetric eigenvalue-equivalent matrix \( \tilde{W} \). The Householder transformation and QL transformation sub-stages are used to find all of the eigenvalues of the neighborhood matrix. The Householder transformation sub-stage takes \( \tilde{W} \) as input and forms the tri-diagonal matrix whose eigenvalues are computed by the QL transformation sub-stage. Computing all of the eigenvalues of the neighborhood matrix approximately takes 99% of the total serial response time as shown in Table 1.

Stage B needs the eigenvalues of the neighborhood matrix to calculate the determinant of \((1-\rho W)\) at each step of non-linear one-dimensional parameter optimization using the golden section search [3]. There are around 80 iterations in the optimization for all the input sets and neighborhood matrices used. Thus, it is much more time-saving to compute the eigenvalues of a large dense matrix i.e. \( W \) once than to compute the determinant of a large dense matrix i.e. \((1-\rho W)\) 80 times. Equation 3 expresses this simplification. The optimization is \(O(n)\) complexity.

\[
|1 - \rho W| = \prod_{i=1}^{n} (1 - \rho_{l_i}) \quad \text{taking the logarithm} \quad \ln |1 - \rho W| = \sum_{i=1}^{n} \ln(1 - \rho_{l_i})
\]

**Eqn 3.** The relationship of the eigenvalues of \( W \) to the logarithm of the determinant of \((1-\rho W)\)

Finally, stage C computes the sum of the squared error, i.e., the \( SSE \) term, which is \(O(n^2)\) complex. Table 1 shows our measurements of the serial response times of stages of the exact SAM solution based on ML theory.
Each response time given in this study is the average of five runs. We note that stage $A$ takes a large fraction of total time.

<table>
<thead>
<tr>
<th>Problem size (n)</th>
<th>Machine</th>
<th>Time (sec)</th>
<th>Stage $A$</th>
<th>Stage $B$</th>
<th>Stage $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Computing Eigenvalues</td>
<td>ML Function</td>
<td>Least Squares</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>SGI Origin</td>
<td>78.10</td>
<td>0.41</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM SP</td>
<td>69.20</td>
<td>1.30</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM Regatta</td>
<td>46.90</td>
<td>0.58</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>6400</td>
<td>SGI Origin</td>
<td>1735.41</td>
<td>5.06</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM SP</td>
<td>1194.80</td>
<td>17.65</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM Regatta</td>
<td>798.70</td>
<td>6.19</td>
<td>0.42</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>SGI Origin</td>
<td>6450.90</td>
<td>11.20</td>
<td>1.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM SP</td>
<td>6546.00</td>
<td>66.88</td>
<td>1.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM Regatta</td>
<td>3439.30</td>
<td>24.15</td>
<td>0.93</td>
<td></td>
</tr>
</tbody>
</table>

### 3.1 Stage $A$: Computing Eigenvalues

Stage $A$ can be parallelized using parallel eigenvalue solvers [7,10,12]. If the source code of the parallel eigenvalue solver is available, one may be able to modify it to tune the performance by changing the parameters, e.g., scheduling technique, chunk size i.e., the number of iterations per scheduling step, etc.

We use a public domain parallel eigenvalue solver from the Scalapack Library [12]. This library is available on MPI based communication paradigm. Thus, we use hybrid programming technique to exploit this library within OpenMP, a shared memory programming model which is preferred within each node of IBM Regatta. In future work, we will use OpenMP within node and MPI across nodes of IBM Regatta. We modified the source code of the parallel eigensolver within Scalapack to allow evaluation of different design decisions including the choice of scheduling techniques and chunk sizes. OpenMP provides a rich set of choices for scheduling techniques i.e., static, dynamic, guided, affinity scheduling techniques.

Another important design decision relates to the partitioning of data items. We instructed OpenMP to partition the neighborhood matrix across processors.

### 3.2 Stage $B$: Fitting the Autoregression Parameter

The golden section search algorithm itself is left un-parallelized since it is very fast in serial format and the response time may increase due to the communication overhead. The serial golden section search stage has linear complexity. However, the golden section search needs to compute the logarithm of the maximum-likelihood function, all of whose constant (spatial statistics) terms are computed in parallel.

### 3.3 Stage $C$: Least Squares

Once the estimate for the autoregressive parameter $\hat{\rho}$ is computed, the estimate for the regression coefficient $\hat{\beta}$, which is a scalar in our spatial auto-regression model, is calculated in parallel. The formula for $\hat{\beta}$ is derived from ML theory. The estimate of the common variance of the error term $\hat{\sigma}^2$ is also computed in parallel to compare with the actual value. The complexity is reduced to $O(n^2/p)$ from $O(n^2)$ due to the parallelization of this stage.

### 4. Experimental Design

Our experimental design answers four important questions:

1. Which load-balancing method provides best speedup?
2. How does problem size ($n$) affect speedup?
3. How does chunk size ($B$) affect speedup?
4. How does number of processors ($p$) affect speedup?

Figure 2 summarizes the factors and their parameter domains. The most important factors are load-balancing technique, problem size, chunk-size and number of processors. These factors determine the performance of the parallel formulation. The neighborhood structure could be extended to 8 and more neighbors even though we used 4 neighbors in this study. The load-balancing techniques of OpenMP can be grouped in four major classes:

1. Static Load-Balancing (SLB)
   - Contiguous Scheduling: Since $B$ is not specified, the iterations of a loop are divided into chunks of $n/p$ iterations each. We refer this scheduling as "static $B=n/p$".
   - Round-robin Scheduling: The iterations are distributed in chunks of size $B$ in a cyclic fashion. This scheduling is referred to as "static $B=\{1,4,8,16\}$".

2. Dynamic Load-Balancing (DLB)
   - Dynamic Scheduling: If $B$ is specified, the iterations of a loop are divided into chunks containing $B$ iterations each. If $B$ is not specified, then the chunks consist of $n/p$ iterations. The processors are assigned these chunks on a "first-come, first-do" basis. Chunks of the remaining work are assigned to available processors.
   - Guided Scheduling: If $B$ is specified, then the iterations of a loop are divided into progressively smaller chunks until a minimum chunk size of $B$ is reached. The default value for $B$ is 1. The first chunk contains $n/p$ iterations. Subsequent chunks consist of $n/p$ iterations divided by $p$.
iterations. Available processors are assigned chunks on a "first-come, first-do" basis.

3. Quasi-Dynamic Load-Balancing composed of both static and dynamic (QDLB).
   • Affinity Scheduling: The iterations of a loop are initially divided into \( p \) chunks, containing \( \frac{n}{p} \) iterations. If \( B \) has been specified, then each processor is initially assigned to a chunk, and is then further subdivided into chunks containing \( B \) iterations. If \( B \) has not been specified, then the chunks consist of half of the number of iterations remaining iterations. When a thread becomes free, it takes the next chunk from its initially assigned partition. If there are no more chunks in that partition, then the thread takes the next available chunk from a partition initially assigned to another thread.

4. Mixed Load-Balancing:
   • Mixed1 and Mixed2 Scheduling: Composed of both static and round-robin scheduling.

The first response variable is speedup and is defined as the ratio of the serial execution time to the parallel execution time. The other response variable is parallel efficiency and it is a metric that is defined as the best speedup number obtained on 8 processors divided by 8. The standard deviation of five runs reported in our experiments for problem size 10000 is 16% of the average run-time (i.e. 561 seconds). It is 9.5% (i.e. 76 seconds) of the average run-time (i.e. 5.2 seconds) for problem size 2500.

<table>
<thead>
<tr>
<th>Factor Name</th>
<th>Parameter Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>f77 w/ OpenMP &amp; MPI</td>
</tr>
<tr>
<td>Problem Size</td>
<td>(2500,6400,10000)</td>
</tr>
<tr>
<td>Neighborhood Structure</td>
<td>2-D w/ 4-neighbors</td>
</tr>
<tr>
<td>Method</td>
<td>Maximum Likelihood for exact SAM</td>
</tr>
<tr>
<td>AR Parameter</td>
<td>(0,1)</td>
</tr>
<tr>
<td>Load-Balancing</td>
<td>Contiguous (Ann): Static B=8, Dynamic B=8, Affinity B=8, Guided B=8</td>
</tr>
<tr>
<td></td>
<td>Combined (Ann): Static B=8, Dynamic B=8, Affinity B=8, Guided B=8</td>
</tr>
<tr>
<td>Machine</td>
<td>IBM Regatta (Renesas)</td>
</tr>
</tbody>
</table>

Figure 2. The experimental design

4.1 Which load-balancing method provides the best speedup?

Experimental Setup: The response variables are speedup and parallel efficiency. Even though the neighborhood structure used is the 4-nearest neighbors for multi-dimensional geo-spaces, our method can solve for any type of neighborhood matrix depending on different structures.

Trends: Figure 3 summarizes the average speedup results for different load-balancing techniques. For each problem size, affinity scheduling appears to be the best. For example, in Figure 3(a) affinity scheduling with chunk-size 1 provides best speedup. In Figure 3(b) for problem size 6400, affinity scheduling with chunk-size 8 provides best speedup. Affinity scheduling with chunk-size \( \frac{n}{p} \) provides best speedup in Figure 3(c) for problem size 2500. The main reason is that affinity scheduling does scheduling both at compile time and run-time, allowing it to adapt quickly to the dynamism in the program without much overhead. The best parallel efficiency obtained for problem size 10000 is 93.13%; for problem size 6400 it is 83.45%; and for problem size 2500 it is 66.26%. It is then seen that the parallel efficiency increases as the problem size increases. This is due to the fact that as the problem size gets larger, the ratio of parallel time spent in the code to the serial time spent increases.

Figure 3. The best speedup results from each class of load-balancing techniques (i.e. mixed1, mixed2, static w/o \( B \), with \( B=\{1,4,8,16\} \); dynamic with \( B=\{n/p, 1, 4, 8, 16\} \); affinity w/o \( B \), with \( B=\{1,4,8,16\} \); guided w/o \( B \) or with \( B=\{n/p\} \), and \( B=\{4,8,16\} \) for problem sizes (n) a)
10000, b) 6400 and c) 2500 on 1, 4 and 8 processors. Mixed1 scheduling uses static with $B=4$ (round-robin with $B=4$) for non-uniform workload and static $w/o B$ (contiguous) for uniform workload. Mixed2 scheduling uses static with $B=16$ (round-robin with $B=16$) for non-uniform workload and static $w/o B$ (contiguous) for uniform workload.

4.2 How does problem size impact speedup?

**Experimental Setup:** The number of processors is fixed at 8 processors. The best and worst load-balancing techniques namely affinity scheduling and guided scheduling are presented as two extremes. Speedup is the response variable. The chunk-sizes and problem sizes are varied.

*Trends:* The two extremes for the load-balancing techniques are shown in Figure 4. The speedup increases in a linear fashion with problem size in the case of affinity scheduling. In the case of guided scheduling, even though the increase in speedup is not linear as the problem size increases, there is still some speedup.

4.3 How does chunk-size affect speedup?

**Experimental Setup:** The response variable is the speedup. The number of processors is fixed at 8 processors. The problem sizes and the chunk sizes are varied. We want to compare two load-balancing techniques, static scheduling and dynamic scheduling. Static scheduling is purely arranged at compile time, while dynamic scheduling is arranged purely at runtime.

*Trends:* Figure 5 presents the comparison. As can be seen, there is a value of chunk size between 1 and $n/p$ that results in the highest speedup for each load-balancing scheme. The dynamic scheduling reaches the maximum speedup when chunk size is 16, while static scheduling reaches the maximum speedup at chunk size 8. This is due to the fact that dynamic scheduling needs more work per processor in order to beat the scheduling overhead. There is a critical value of the chunk size for which the speedup reaches the maximum. This value is higher for dynamic scheduling to compensate for the scheduling overhead. The workload is more evenly distributed across processors at the critical chunk size value.

4.4 How does number of processors affect speedup?

**Experimental Setup:** The chunk size is kept constant at 8 and 16. Speedup is the response variable. The number of processors is varied i.e. {4, 8}. The problem size is fixed at 10000.

*Trends:* As Figure 6 shows, the speedup increases as the number of processors goes from 4 to 8. The average speedup across all scheduling techniques is 3.43 for the 4-processor case and it is 5.91 for the 8-processor case. Affinity scheduling shows the best speedup, on average 7 times on 8 processors. Therefore,
the speedup increases as the number of processors increases.

![Figure 6. Analyzing the effect of number of processors on speedup when problem size is 10000 using 4 and 8 processors. Mixed1 scheduling uses static with $B=4$ (round-robin w/ $B=4$) for non-uniform workload and static w/o $B$ (contiguous) for uniform workload. Mixed2 scheduling uses static with $B=16$ (round-robin w/ $B=16$) for non-uniform workload and static w/o $B$ (contiguous) for non-uniform workload.](image)

**5. Conclusions and Future Work**

Linear regression is one of the best-known classical data mining techniques. However, it makes the assumption of independent identical distribution (IID) in learning data samples, which does not apply to geospatial data. In the spatial auto-regression model (SAM), spatial dependencies within data are taken care of by the auto-correlation term and the linear regression model thus becomes a spatial auto-regression model. Incorporating the auto-correlation term enables better prediction accuracy. However, computational complexity increases due the logarithm of the determinant of a large matrix, which is computed by finding all of the eigenvalues of another matrix. Parallel processing helps provide a practical solution to make SAM computationally efficient.

In this study, we developed a parallel formulation for a general exact estimation procedure for SAM parameters that can be used for spatial datasets embedded in multi-dimensional space, which can be used for location prediction problems. We studied various load-balancing techniques allowed by the OpenMP API. The aim was to distribute the workload as uniformly as possible among the processors. The results show that our parallel formulation achieves a speedup up to 7 using 8 processors. We are developing algebraic cost models to analyze the experimental results to further improve the speedups. We will expand our experimental studies to include a larger number of processors via hybrid parallel programming and other parameters e.g. degree of auto-correlation. In the long-run, we plan to develop parallel formulations for approximate solution procedures for SAM that exploit sparse matrix techniques.

**Acknowledgments**

This work was partially supported by Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under contract number DAAD19-01-2-0014. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. The authors would like to thank the members of the Spatial Database Group for valuable discussions and Minnesota Supercomputing Institute for using their computing resources. The authors thank Kim Koffolt for helping improve the readability of this paper.

**References**

Appendix: 
Algebraic Cost Model

This section concentrates on the analysis of ranking of the load-balancing techniques. The clock speed is 1.3GHz. The cache line of IBM Regatta is 128 bytes long i.e. 16 floating-point numbers.

The contiguous and guided scheduling techniques can be categorized as the poorest techniques. The round-robin scheduling technique comes next in the ranking. The \( \alpha \) coefficient for the cost of local work term \( (c_{LW}) \) in eqn4 implies this selection. The \( \alpha \) coefficient is dependent on how granular the chunk-size is spread across the processors. The finer selection is achieved by the cost of load-balancing term \( (c_{SLB}) \). In other words, \( c_{SLB} \) represents the synchronization overhead in the parallel program. This term is zero for a serial program. Figure 7 abstracts any parallel program parallelized at finer granularity level in terms of these two major terms. Here, the main assumption for this cost model is that the time spent in the parallel region is much more than the time spent in the serial region and \( B \ll n/p \), both of which hold for our formulation.

Figure 7. Generic parallel program parallelized at finer granularity level

\[
c_{\text{total}} = \alpha c_{\text{LW}} + c_{\text{SLB}}
\]

Eqn 4. Algebraic cost model expression for parallel programs parallelized at with finer granularity level

Since affinity scheduling does an initial scheduling at compile time, it will have less overhead than the dynamic scheduling. The partial ranking is shown in Figure 8.a. The worst point in this space is at the right-upper corner where synchronization cost (i.e. \( c_{SLB} \)) is too high and the load-imbalance (i.e. \( c_{LW} \)) reaches the most non-uniform state. The best point is at the left-lower corner where synchronization cost is the lowest and the load-imbalance is almost zero, which is hypothetical. The real scenarios lie between these two extremes. So, the decision tree in Figure 8.b summarizes the total ranking of the load-balancing techniques used in this study.

Figure 8. a) Partial Ranking and b) Total Ranking of load-balancing techniques

![Figure 8](image-url)