Proceedings of the *FOSS/GRASS Users Conference* - Bangkok, Thailand, 12-14 September 2004

GRASS-based High Performance Spatial Interpolation Component for Spatial Decision Support Systems

Kun Lu*, Steve Goddard

* Department of Computer Science and Engineering, University of Nebraska-Lincoln, Nebraska, 68588-0115, US, tel. +014024721779, e-mail klu@cse.unl.edu

1 Introduction

Spatial Data Interpolation (SDI) is widely used in GIS (Geographical Information Systems) areas such as natural resource modeling [1] and spatial decision support systems (SDSS) [2][3] to generate continuous surfaces maps. In the real world, it is impossible to exhaustively sample data at every desired location because of time and cost constraints. Spatial Interpolation allows the estimation of attribute values at any location within the region of interest. It is a computational procedure of estimating/predicting surface values for a continuous geo-spatial variable at unmeasured locations within a region where discrete sample values are measured. For example, climate data such as precipitation are collected from a limited number of scattered weather stations. With Interpolation methods, these scattered data can be used to generate precipitation maps covering the whole nation.

There are three typical SDI methods: Inverse Distance Weighting (IDW), Spline, and Kriging. In IDW, the value of an unmeasured location is calculated as a linear weighted combination of sample values within a local neighborhood of the point being estimated. The weight of a sample point is assigned according to the inverse of its distance to the point being estimated. Spline estimates surface values by fitting a minimum curvature surface to the sample data. Kriging, also known as geo-statistics, has more sophisticated algorithms. Comparing with the first two, Kriging is more likely to provide a better estimation [4], but it is also believed to be the most computational intensive method as well.

GRASS provides SDI functionalities through commands such as s.surf.idw, s.surf.rst and, s.surf.krig, etc [5]. However, there are two potential drawbacks when applying these GRASS tools in modern GIS. The first one is the limited support GRASS provides for geo-spatial services in distributed, heterogeneous computing environments. Like most traditional GIS, GRASS functions primarily as a single operator system in which commands are issued by users. This limits its use in component-oriented distributed systems. The second drawback is the poor performance of GRASS SDI tools when used in large-scale models. The term performance here refers to the time it takes to execute GRASS interpolation commands. The execution time for tasks with various sizes could range from seconds to hours. A response time of such a wide range heavily restricts the application of GIS on the Web.

The National Agriculture Decision Support System (NADSS) is a web-based decision support system built on GRASS. It’s a joint project with the U.S. Department of Agriculture (USDA) and the University of Nebraska-Lincoln. SDI is heavily used in the NADSS to generate agricultural-climatic data maps for use in drought risk analysis and assessment. This work, as part of the NADSS project, proposes a component-based spatial data interpolation framework to overcome the two drawbacks stated above. In this framework, the interpolation task could be performed either sequentially through a local

GRASS library call, or, in parallel by invoking a specific interpolation component on a remote High Performance Computer or computer cluster, depending on the task size. The requested computing resource is adapted to the task size; hence the response time could be fixed in an acceptable range. This paper describes the SDI framework and an implementation of a CORBA-based, high-performance Kriging interpolation component. Parallel Kriging is implemented in C using the Message Passing Interface (MPI) library. In Section 2 we introduce the Kriging algorithm used in GRASS. Section 3 describes our proposed high-performance SDI framework and how to integrate GRASS into the framework. Section 4 covers the parallel Kriging algorithm used in our high-performance Kriging component. Finally Sections 5 and 6 present experimental results and conclusions.

2 Kriging

Kriging is a geo-statistical interpolation algorithm developed by Matheron and Krige [10] originally for use in the mining industry. Like IDW, Kriging also uses a linear weighted combination of a number of neighboring sample values to estimate the value at the unmeasured locations. However, Kriging uses a more sophisticated method in weighing the neighboring sample points than that of IDW. It assumes the surface variation over space changes with a rate and expresses this rate in a variogram that shows how the average difference between values at points changes with distance between points. Thus generally there are two steps in a Kriging interpolation [6]. The first step is to construct a variogram, and then, the second step is to compute the estimates using the variogram model.

2.1 Constructing a Variogram

Usually, a variogram is either experimentally constructed from sample observations or an empirical model is used. The most common empirical models [6] are the spherical model, the exponential model, the Logarithmic model, the power model and the Gaussian model. The spherical model is used in this work. A sample semi-variogram using the spherical model is illustrated in Figure 1. The properties of a semi-variogram model include nugget, range and sill.

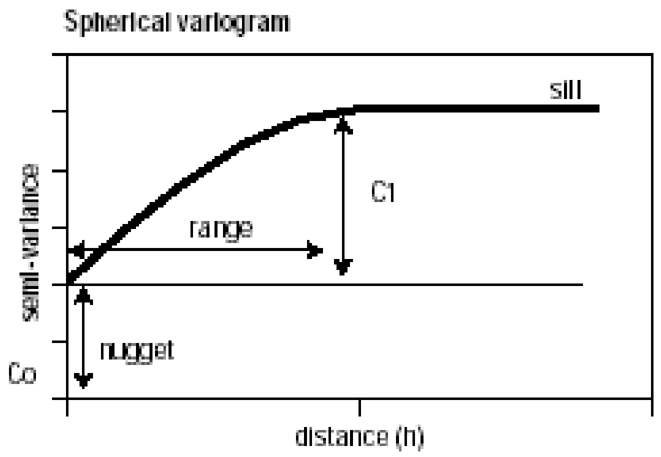

Figure 1: A sample semi-variogram [7].

The nugget is the intercept of the semi-variogram with the vertical axis. The range is the distance beyond which there is no spatial correlation. The sill is the maximum semi-variogram value, which is the plateau in Figure 1. The sill value is reached when the distance between two observations is larger than the range. The semi-variogram values are small at nearby locations, and they increase to a constant value (sill) as the distance *h* increases. Hence small semi-variogram values imply high spatial correlation. The formulation of the spherical mode is given in Equation (2.1) where *a*, C_0 and C_1 are constants; C_0 is the nugget, $C_0 + C_1$ is the sill value, *a* is the range, and *h* is distance, which is the input variable.

$$\text{Spherical model: } \gamma(h) = \begin{cases} C_0 + C_1\left(1.5\frac{h}{a} - 0.5\left(\frac{h}{a}\right)^3\right) & \text{if } \quad |h| \le a \\ C_0 + C_1 & \text{if } \quad |h| > a \end{cases} \tag{2.1}$$

The constants are determined experimentally, and their derivation is beyond the scope of this work. However, typically, *sill = 1* and *nugget = 0*, but range values are application dependent.

2.2 Computing the Estimates

Once the variogram model is constructed, it is used to compute the weights used in the Kriging interpolation. To estimate a value at unknown location s_0, *n* nearest neighbor sample values are weighted according to the variogram. Each weight w_i used in kriging at s_0 can be represented as [8]:

$$\sum_{j=1}^{n} w_j \gamma(h_{ij}) + \mu = \gamma(h_{i0}) \text{, for each } i,\ 1 \le i \le n \tag{2.2}$$

where *n* is the number of neighboring samples, *i* is the index of the sample locations, h_{ij} is the Euclidean distance between location i and *j*. Subscript *0* indicates the unknown location s_0. μ is a constant which is solved later. Therefore, Equation (2.2) contains *n* sub equations with *n + 1* unknowns (the weights and μ). One more constraint is needed to solve all the weights:

$$\sum_{i=1}^{n} w_i = 1 \tag{2.3}$$

Thus, by combining Equations (2.2) and (2.3), we could compute the weights using the following covariance matrix:

$$w = C^{-1}D \tag{2.4}$$

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ \mu \end{bmatrix} \qquad C = \begin{bmatrix} \gamma(h_{11}) & \cdots & \gamma(h_{1n}) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \gamma(h_{n1}) & \cdots & \gamma(h_{nn}) & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} \gamma(h_{10}) \\ \vdots \\ \gamma(h_{n0}) \\ 1 \end{bmatrix}$$

where the matrix C consists of covariance values between sample points, the vector D consists of covariance values between the sample points and the point being estimated, vector *w* consists of *n+1* unknown variables to be solved. In Equation (2.4), the matrix inversion is the most computationally intensive part of the Kriging method, which is computed for each pixel in the output surface since the nearest sample points vary for different pixels.

After weights are calculated, an estimated value can be obtained using a linear combination of the neighboring sample values:

$$\hat{v}(s_0) = \sum_{i=1}^{n} w_i(s_0)v(s_i) \qquad (2.5)$$

where $\hat{v}(s_0)$ is the estimated value at location s_0; $v(s_i)$ is the observed sample value at location s_i; n is the number of points used in calculation; and $w_i(s_0)$ is the weight of the sample data point at s_i for the estimation of the value at location s_0.

3 A GRASS-Based High-Performance SDI Framework

We have developed a high-performance spatial data interpolation framework in which GRASS SDI commands can be transformed into distributed SDI components. Section 3.1 describes a layered architecture on which the GRASS-based SDI framework is constructed. Section 3.2 describes a working model of the high-performance SDI component.

3.1 A layered architecture for Component-based GIS/SDSS

Figure 2 represents the architecture of a component-based GIS/SDSS system in which GRASS is used as a GIS kernel. A layer in the architecture provides services to its upper layer. Each layer is independent from its underlying layer given that the interface does not change.

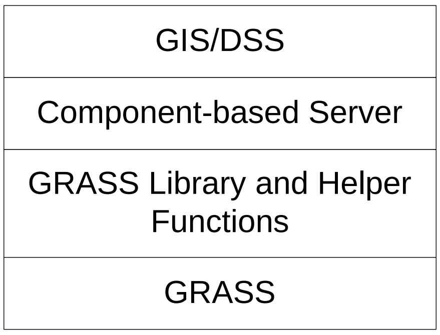

Figure 2: A GRASS-based GIS/SDSS architecture.

GRASS is at the bottom layer in the architecture. This layer provides basic spatial data management and processing functionalities, however it cannot be readily used in a distributed environment since GRASS is a command-oriented GIS. Converting these GRASS commands into object-oriented methods is crucial for a GRASS-based component and is achieved in the second layer.

The second layer provides sophisticated geoprocessing services based on the underlying GRASS layer. The GRASS Library essentially wraps the GRASS commands as a shared library with open APIs [9]. In this way the core spatial data processing functions of GRASS can be provided to an object-oriented component in a distributed environment such as CORBA, DCOM and JAVA RMI. The helper functions provide flexible geo-spatial functions that are not directly provided by GRASS commands. For example, in the GRASS environment, only one active mapset is allowed at a time. This works fine for sequential geo-processing modes such as GRASS input commands. For a distributed GIS like NADSS, multiple objects of distributed components may request geo-processing at the same time. The GIS is expected to be able to assign one mapset exclusively for each request so that multiple requests can be handled simultaneously. With a helper function, component servers in an upper layer can assign each request a free mapset if available.

Component-based servers reside in the third layer in the architecture. This layer provides SDI services, mapping services and other domain-specific services for decision support. Each component in this layer can be deployed on different platforms in a distributed computing environment to optimize resource utilization.

3.2 A high-performance SDI component in NADSS

This section takes Kriging as an example to describe a working model of a high-performance SDI component. The SDI components in NADSS provide high-performance Kriging interpolation services to spatial map servers.

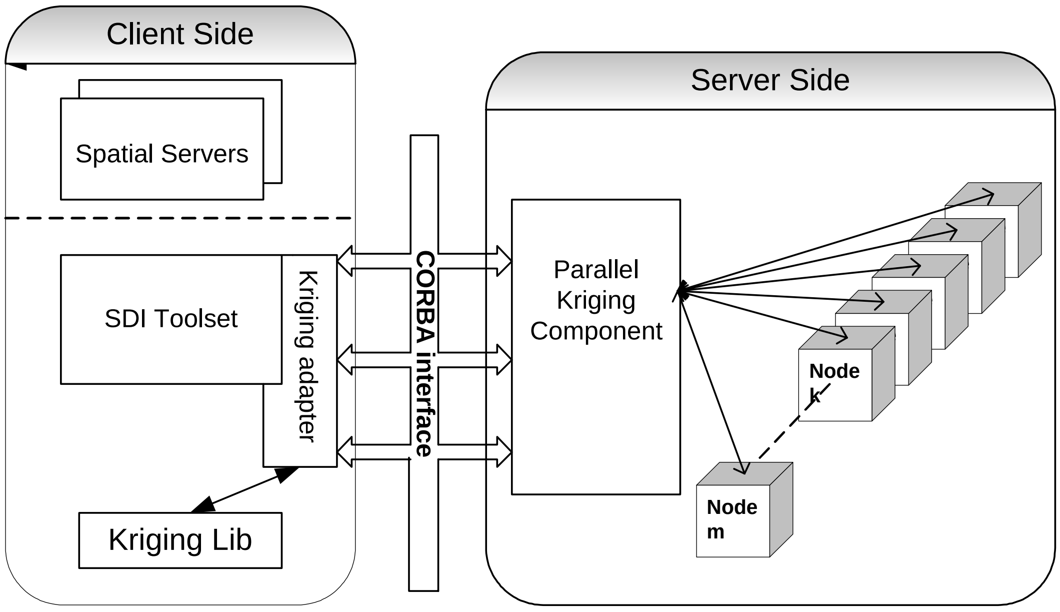

Figure 3: A working model of parallel Kriging component

Figure 3 illustrates how the Kriging component works in NADSS. The client side applications run on a low-end platform while the server side components are running on a high-performance platform such as a parallel computer or a Beowulf computer cluster. Detailed structure on the client side exhibits the layered structure. The spatial servers belong to the third layer in the architecture described in Section 3.1. It provides mapping services to the upper layer and acquires SDI services from the underlying layer. The SDI toolset and Kriging Lib belong to the second layer. Each interpolation tool in the SDI toolset is associated with an interpolation adapter. The purpose of the interpolation adapters is to direct the interpolation request to an appropriate SDI handler. A SDI handler here could either be a GIS library call or an interpolation component server. The Kriging adapter deals with Kriging requests. It can either perform the Kriging interpolation locally through a GRASS library call, or invoke a remote Kriging component to carry out the interpolation in parallel, depending on the problem size, to achieve an optimal performance. The Kriging adapter communicates with a remote Kriging component using a CORBA interface. On the server side, the parallel Kriging component is a CORBA-based server implemented in C++ with MPI (the Message Passing Interface). The Kriging Server receives input GRASS site data and Kriging parameters from the client, initializes the GRASS environment, assigns jobs to multiple CPUs, assembles the result and sends it back to the client. The output of the MPI Kriging is saved in the form of GRASS raster data, which is a group of files in the GRASS mapset. Before the Kriging component can send the results to the client, the raster data is first converted into a single ASCII file. This conversion is accomplished by calling proper GRASS library functions in the underlying layer. Therefore both client side and server side applications in the working model are conforming to the layered architecture. The scalability of this high-performance SDI model is achieved by running multiple server instances on several high-performance computers. The parallel Kriging algorithm is discussed in the following Section.

4 Parallel Kriging

Different parallel strategies of Kriging interpolation can be identified in the literature. In general, these strategies can be divided into two categories. One is using a parallel programming language or parallel library calls on tightly coupled machines [10][11][12], the other is applying domain decomposition to the Kriging surface [1][12]. This work implements the second parallel strategy.

The output of the Kriging interpolation is a raster map in which Kriging results are stored as pixel values. To generate a raster map, values for all the pixels in the raster map need to be estimated. In Section 2.2, we introduced the steps to estimate the value of a single pixel in the output raster. The Kriging procedure at one pixel is independent from all other pixels. That is, the same procedure repeats for each pixel in the raster map. Therefore, the Kriging problem naturally fits a SPMD parallel approach such as domain decomposition.

4.1 Domain decomposition

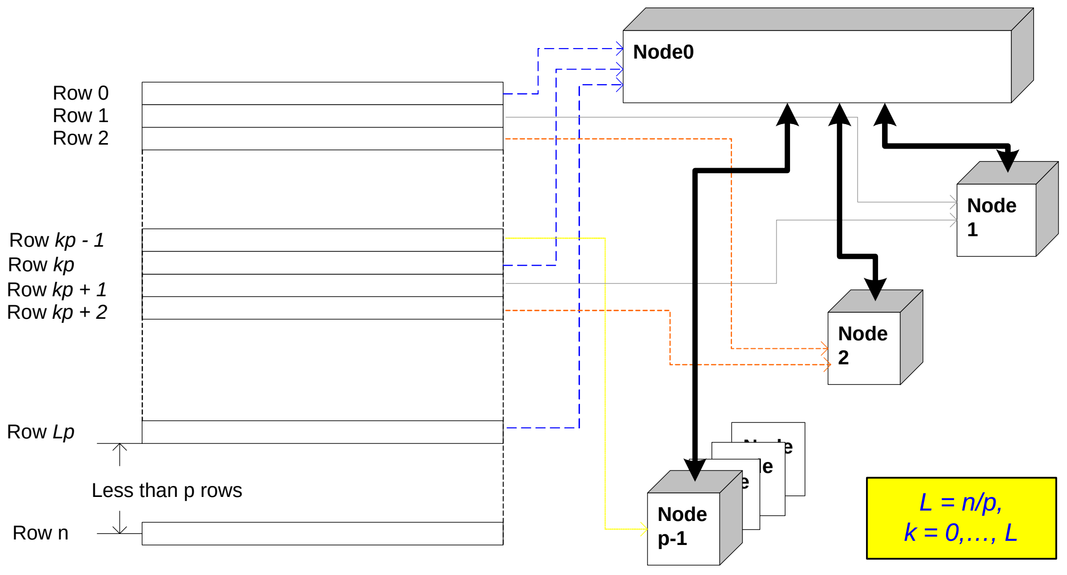

Figure 4: An illustration of domain decomposition: *n*-row raster file with *p* processors.

Figure 4 presents the implementation of domain decomposition. To generate a Kriging raster map with *n* rows in parallel with *p* processors. We can choose to decompose the domain along either rows or columns. Since the algorithm is implemented in C, which is a row major order programming language, we decompose the domain along rows. Kriging for one row of the raster map is defined as a primitive task. An inner loop over columns is the same for all primitive tasks. Rows in the raster map are assigned in round-robin fashion to each processor as shown in Figure 4. Node *0* will be assigned rows *0, p, 2p, …, Lp*, node 1 will be assigned rows *1, p+1, …, Lp + 1*, and for *0 < i < p*, node *i* will be assigned rows *i*, *p* + *i*, *2p* + *i*, …, and so on. Each node will be assigned at least $L = \lfloor n/p \rfloor$ rows. The remaining *n* – *L* rows will be assigned evenly to the first *n* – *L* processors. In this way, the first *n* – *L* processors perform *L* + *1* primitive tasks whereas other processors perform *L* primitive tasks, and hence a balanced load among the processors is achieved.

4.2 Implementation

The MPI code employs node 0 as a master node and other nodes as workers. Besides the Kriging operations that every node must perform, node *0* is also responsible for all the I/O operations during the Kriging interpolation. Prior to the actual interpolation steps, node 0 takes care of the initialization job including acquiring the GRASS environment, processing input parameters and retrieving site data from the file system. (This is true for a shared memory machine in which the GRASS environment variables set by the Kriging server are available for all nodes. For a computer cluster, each worker node also has to do the GRASS initialization as well.) After that, node *0* broadcasts the Kriging parameters and site data to all the worker nodes using MPI_Bcast. The Kriging parameters contain information about the dimension and resolution of the Kriging surface. Worker nodes can use this information to locate the rows and columns on the Kriging surface. Each node then determines its portion of the Kriging surface according to the decomposition algorithm and allocates a local buffer that can hold all Kriging results for the node. Pixels on a Kriging surface are interpolated row by row, and for each pixel, the program will go through the Kriging steps discussed in Section 2.2. After a pixel is krigged, the estimated value is stored in the local buffer, and then the next pixel is processed. No communication happens until all the nodes finish their part of the job. A barrier (MPI_Barrier) is placed at the beginning and at the end of the Kriging computation so that we can separate the communication from computation. This helps in the performance prediction, as discussed later. After all nodes have accomplished their Kriging computation, worker nodes send their results to the master node by issuing a non-blocking send call (MPI_ISend). Correspondingly, the master node posts a series of non-blocking receive calls (MPI_IRecv) for each worker node, followed by a series of waits (MPI_Wait). The non-blocking receive calls allow the program to proceed without waiting for the sending data to be received [13]. Therefore multiple receive calls can be issued in a batch. After the wait calls are finished, all the results are copied to the proper place of the cell matrix buffer that stores all the estimated value for the Kriging surface.

5 Results and Discussion

We tested the parallel Kriging component on rcf.unl.edu, a 32 processor SGI Origin 300 (shared-memory) machine. The client component is running on yeti.unl.edu, a Sun workstation. A Kriging interpolation of a Palmer Drought Severity Index (PDSI) [14] on a US-wide region was selected as the test problem. Various combinations of output raster size, input sample number and working processors are employed in order to fit polynomial functions that can be used in predicting the Kriging response time.

5.1 Complexity analysis

We mentioned in Section 3 that the Kriging adapter was capable of choosing an optimal computing resource to process Kriging requests depending on the problem size, or in other words, the expected response time for a Kriging request. The computing time for the Kriging problem can be described by a polynomial of the total number of input sample points *n* and the number of pixels in the output raster map *s*, given a fixed number of the nearest neighboring sample points.

$$T_K = a_K + s(b_K + c_K n + d_K n^2) \qquad (5.1)$$

In the above equation, a_K, b_K, c_K, and d_K are platform-dependent coefficients that can be determined in experiments. The quadratic term accounts for the computing time spent in sorting the sample distances to the Kriging pixel. Equation (5.1) can be used in estimating the response time for a request handled by a local GRASS library call.

For a request to be handled remotely by the parallel Kriging component, the complexity analysis is a little more complicated. Essentially, the response time is composed of three parts: Kriging computation time T'_K, message passing communication overhead T_C, and component latency T_L. Equation (5.2) describes the response time for processing a Kriging request using a remote parallel component.

$$T = T_K^{'} + T_C + T_L \qquad (5.2)$$

In our MPI Kriging implementation, the computation is separated from communication among working processors. The computation time for Kriging with a single processor is estimated using Equation (5.1). Assuming the parallelism is fully scalable within a number of processors, the computation time for parallel Kriging with p processors can be obtained by multiply the T_K with a factor *1/p*.

$$T_K^{'} = {T_K}/{p} \qquad (5.3)$$

The communication in the MPI Kriging mainly occurs at the time when the master node collects results from worker nodes. In parallel computing, intensive intercommunication may significantly degrade the performance. The implementation alleviates the intercommunication in a way that each worker node only sends its results once. For a raster map of size *s* krigged using *p* processors, each worker processor sends a part of the raster result, roughly a size of *s/p*. Therefore, T_C can be approximated by the following equation:

$$T_C = (p-1)(C_1 + C_2 s / p) = -C_1 + C_1 p + C_2 s - C_2 s / p$$

As the number of processors increases, the fourth term tends to be insignificant. Thus, the communication overhead is bounded by the following polynomial:

$$T_C < a_C + b_C p + c_C s \qquad (5.4)$$

where a_C, b_C, and c_C are constants. The component latency T_L in Equation (5.2) represents the time incurred due to the use of a remote component instead of a local handler. It includes the CORBA communication time, file I/O overhead, and time for spatial data operations other than Kriging. Assuming the network transfer rate and I/O bus speed are constant, the component latency is linearly related to raster size or the number of pixels in output raster map. Equation (5.5) describes the component latency:

$$T_L = a_L + b_L s \qquad (5.5)$$

where a_L, b_L are constant coefficients.

In Section 5.2 we present our experimental results and analyze the timing coefficients

5.2 Results

5.2.1 Scalability of the parallel Kriging algorithm

Kriging problems with three different sizes are tested to evaluate the scalability of the parallel algorithm. The input sample site number is fixed at 1930 (the number of sample stations for an Oct. 2003 PDSI map), while the output raster size varies from 512^2, 1024^2 to 2048^2. (The size is changed through map resolutions. The interpolation region is unaltered throughout the experiments.) Figure 5 shows how the Kriging time changes with the number of processors. The dotted line describes the total time (including computation, I/O and communication overhead) spent for a MPI run. Each dot represents where a measurement was taken. The solid line describes the time for the Kriging computation, i.e. the T'_K. Starting from the time for one processor, the Kriging time firstly decreases linearly as the processor number increases. Then after some number of processors is reached (for size of 512^2, and 2048^2, this number is about 19, for size of 1024^2, it is around 25), the time curve becomes flat and irregular. This is because the parallel computer (RCF) we used is not available exclusively for this test. Like many real world situations, the load on RCF is variable. Most of the time, one third of the 32 processors in RCF are busying executing other jobs, sometimes even more. When many of these running jobs attempt to access the file system, the I/O bus becomes a bottleneck for the performance. This explains why sometimes we can achieve better scalability (for example, the test for the raster with a size of 1024^2) than other times (the other two runs in Figure 5.).

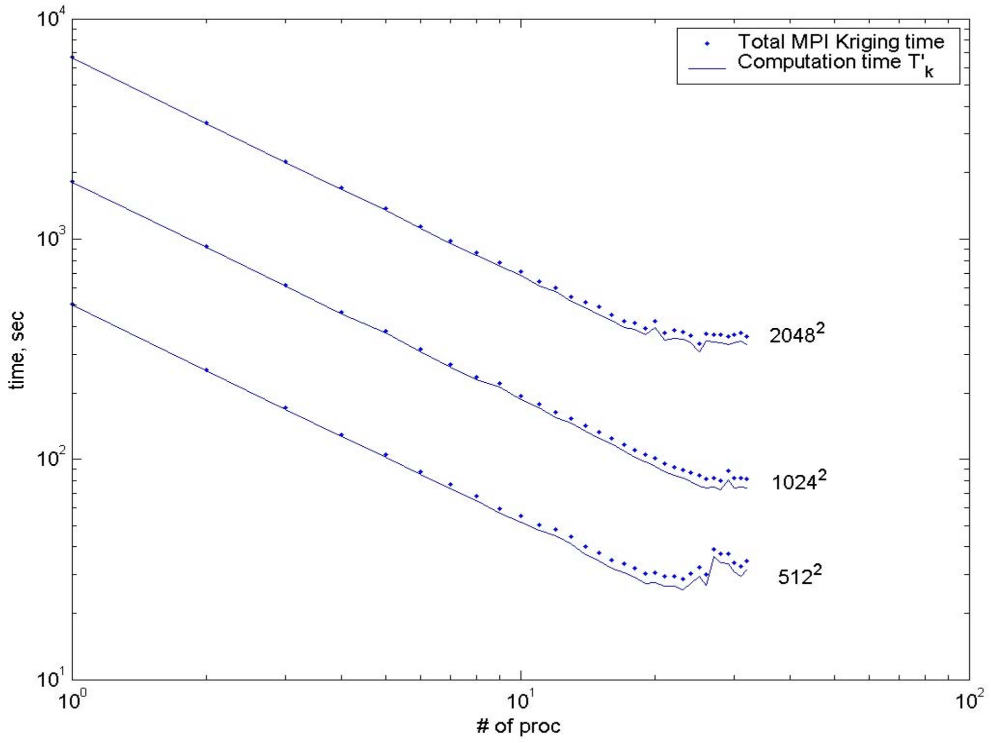

Figure 5: Parallel Kriging time changes with number of processors.

Figure 6 shows the speedup of computation times measured for the 3 tests. The speedup is defined as the ratio between sequential Kriging time and parallel Kriging time. The time used to compute speedup here is computation time T'_K, which does not take communication overhead into account. Within a range of processors (here it is 19), the MPI Kriging exhibits linear speedup for raster maps of a size from 512^2 to 2048^2. We defined the scaling efficiency for *p* processors as ratio of speedup to *p*. Table 1 shows that pretty good scaling efficiencies (>*95%*) are achieved for the three tests.

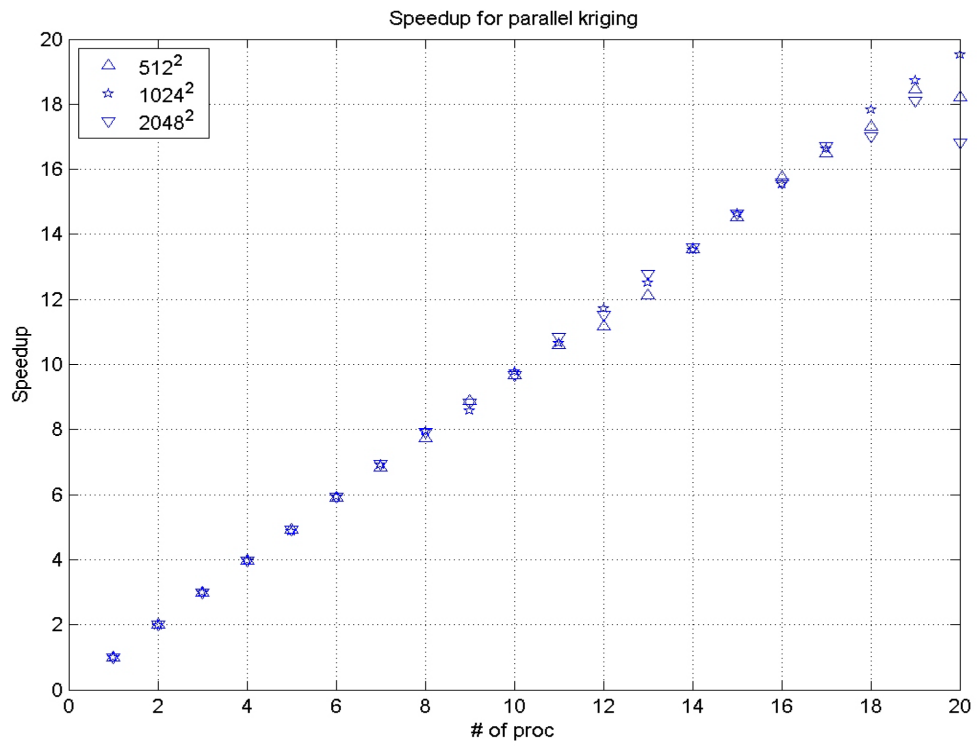

Figure 1: Speedups measured in the test exhibit linear relation to the number of processors.

# proc	Size of Raster map		
	512^2	1024^2	2048^2
1	100%	100%	100%
2	99.76%	99.41%	99.70%
4	99.39%	99.58%	99.08%
8	98.54%	99.45%	98.95%
16	98.46%	97.09%	97.49%
19	97.20%	98.61%	95.25%

Table 1: Efficiency of the parallel Kriging algorithm.

5.2.2 Execution time prediction

Thousands of runs have been made to fit the polynomials discussed in Section 5.1 in order to predict the Kriging response time. For the sequential Kriging test, we vary the input site number from 50 to 3200, and vary the raster size from 50^2 to 1600^2. Thus, each test suit contains all possible combinations of 9 different raster sizes and 7 different input site numbers. Six identical test suits have been launched at different times of the day on different days and the results were averaged to ease of effect of variable load. The sequential experiments were conducted on the client host. Combining the computed coefficients with polynomial function (5.1) results in polynomial function (5.6):

$$T_K = 4.5882 + s(8.1e-5 + 6.861e-7n + 6.54e-11n^2) \qquad (5.6)$$

The fitted polynomial produces the plots shown in Figure 7. The stars in the figure represent measured test results, while the solid line is generated by the polynomial function. The plots show the polynomial gives a pretty good prediction on sequential

Kriging response time. For parallel Kriging, polynomial function (5.1) can be fit in a similar manner with experimental results collected on the server host (RCF).

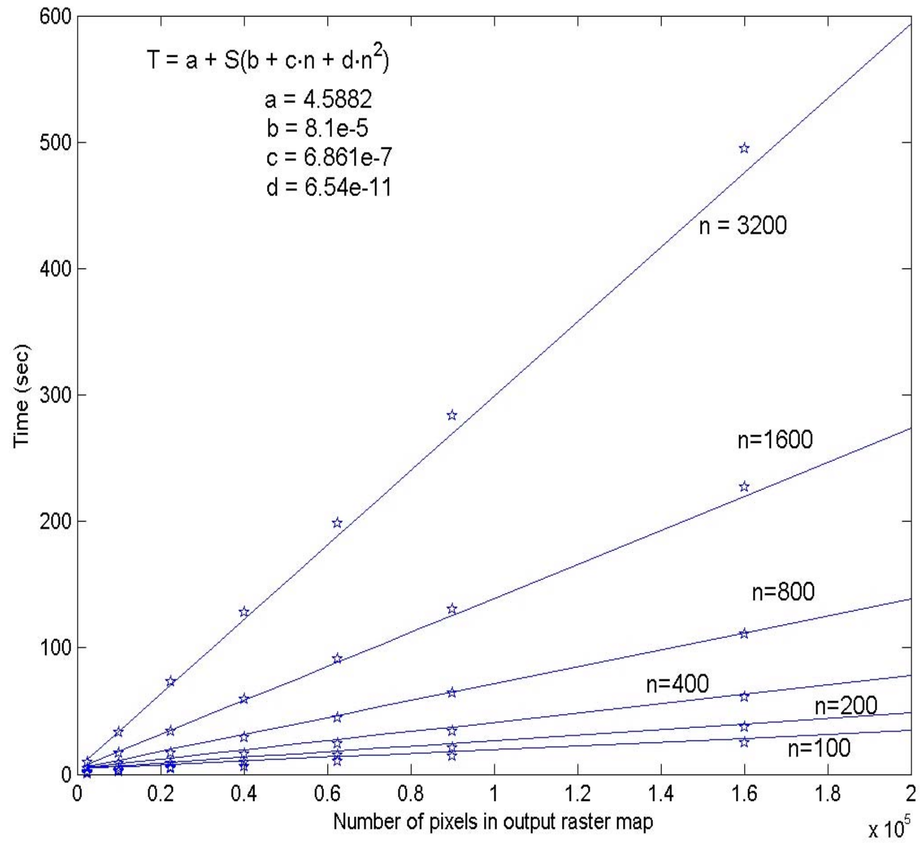

Figure 2: Response time for sequential Kriging

Each test suit used in predicting the parallel Kriging time contains runs in which 6 different sample site numbers vary from 50 to 1600 combined with 11 different raster sizes ranging from 50^2 to 2000^2. The number of processors used in these runs varies from 2 to 16. Identical test suits were run multiple times and the results were averaged. In each run, timings for COBRA latency, file I/O overhead on RCF, MPI communication overhead and Kriging computation time are included in the result. Since RCF is a shared memory computer, the communication overhead T_C is expected to be small. This is proved by the observation of test results. Throughout the test range, the communication overhead never reached 1 second, while the file I/O overhead ranges from about 1 second to ten's seconds, sometimes hundred's of seconds in the worst case. Hence, we exclude the communication overhead when fitting Function (5.2). Equation (5.7) gives the result.

$$\begin{aligned} T &= 5.3949 + 3.4344e - 5 \cdot s + \\ &\quad (8.926 + 1.01e - 4 \cdot s + 2.514e - 7 \cdot sn + 4.09e - 11 \cdot sn^2)/p \end{aligned} \tag{5.7}$$

The prediction and measured response times are plotted in Figures 8-1 and 8-2 for raster maps of size 400^2 and 1600^2, respectively. The figures show the response times for a SDI request handled by the remote parallel Kriging component with the number of processors varying from 2 to 16.

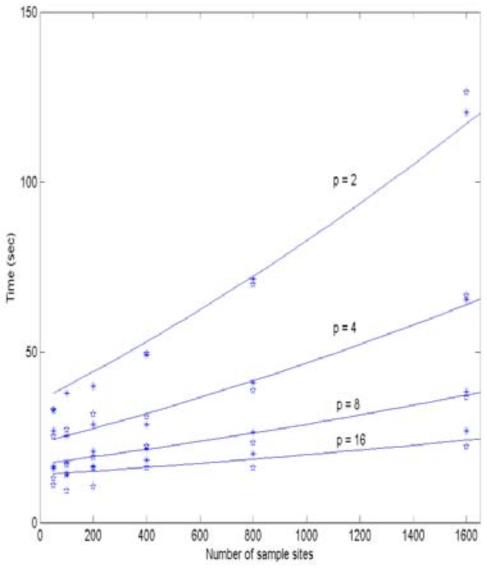

Figure 8-1: raster size = 400^2

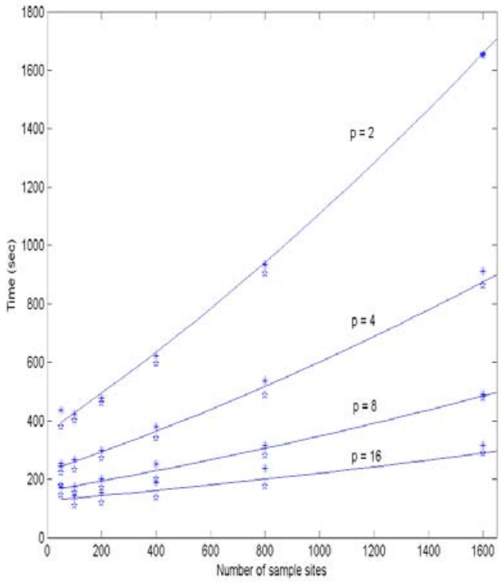

Figure 8-2: raster size = 1600^2

The solid line is produced by the polynomial function (5.7). The asterisk and star marks are the observations from two different runs. For both runs, the polynomial function gives better predictions of response time for a 1600^2-raster map than those for a raster map with a size 16 times smaller. For the same output size, better predictions are obtained with a larger input sample set. This is due to the variable file I/O overhead on the host computer. For the same output raster size and number of processors, a small input sample set tends to result in a short response time, which is more vulnerable to the system load variability than a longer response time. The fitted polynomial is used in the Kriging adapter in which both response times for sequential Kriging and parallel Kriging are estimated using these polynomial functions. Currently, the Kriging adapter simply chooses a Kriging handler that results in a shorter response to process the request. This simple mechanism assumes the requested computing resource is always available. A more practical mechanism should consider other factors such as resource management and request priority. This, however, is beyond the scope of this paper.

6 Conclusions

This work provides a high performance SDI framework in which GRASS commands are integrated with component technology and parallel computing technology to provide high performance SDI service in a distributed computing environment. Taking Kriging interpolation as an example, we implemented a high-performance Kriging component for NADSS to study agricultural-climatic problems. A predictive model is also presented to estimate the response time of the high-performance Kriging component. With the prediction results, the Kriging processing time can be controlled within an acceptable range by adjusting the number of working processors. Experiments were conducted on a 32 processors shared-memory computer. Results exhibit excellent scalability in our parallel algorithm within a range of processors. The predicted response times fairly agree with the experimental results for a range of Kriging inputs. The SDI framework is shown to be an effective solution to overcome the two drawbacks in GRASS SDI support identified in the beginning of this paper.

However, due the limits of experimental computing resources available, the resulting timing model is not ideal for accurately predicating response time. The model only works when a cluster is used exclusively for the SDI framework.

References

[1] J.A. Pedelty, J.T.Morisette, J.L. Schnase, J.A.Smith, High Performance Geostatistical Modeling of Biospheric resources in the Cerro Grande Wildfire Site, Los Alamos, New Mexico and Rocky Mountain National Park, Colorado, Proceedings of the *"Earth Science Technology Conference (ESTC) "*, 2003.

[2] S. Goddard, S. K. Harms, S. E. Reichenbach, T. Tadesse and W.J. Waltman, Geospatial Decision Support for Drought Risk Management, *"Communications of the ACM"*, Vol. 46, No. 1, pp. 35-37, 2003.

[3] S. Wang, D.A. Bennett, M.P. Armstrong, R. Rajagopal, and E. Brands, Using Grid-Enabled Teleimmersive Spatial Decision Support Systems (TIDSS) to Visualize Uncertainty for Water Quality Protection in Agroecosystems, Proceedings of the *"International Conference of Geoinformatics'2002", Nanjing, P.R. China, June, 1-3, 2002.*

[4] S. Anderson, An Evaluation of Spatial Interpolation Methods on Air Temperature in Phoenix, AZ, Department of Geography, Arizona State University[online], available: *http://www.cobblestoneconcepts.com/ucgis2summer/anderson/anderson.htmT*.

[5] GRASS Geographic Information System Information, [Online], available: *http://www.cecer.army.mil/grass/GRASS.main.html*.

[6] Ying Ma, Interpolation of Precipitation and the Standardized Precipitation Index (SPI), Master Project Report. 2003.

[7] A.D. Hartkamp, Interpolation Techniques for Climate Variables, *"Geographic Information Systems Series 99-01", Natural Resources Group*, 1999.

[8] Chao-yi Lang, Kriging Interpolation, Computer Science, Cornell University [online], available *http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-94to95/clang/kriging.html*.

[9] Introduction to GRASSLib, [Online], available: *http://nadss.unl.edu/grasslib*.

[10] K.Kerry and K.Hawick, Spatial Interpolation on Distributd, High-Performance Computers, "*DHPC Technical Report DHPC-015*", Department of Computer Science, University of Adelaide, 1997.

[11] K.Kerry and K.Hawick, Kriging Interpolation on High Performance Computers, Proceedings of the *"High Performance Computing and Networks Europe"*. LNCS 1401, Springer-Verlag 1998.

[12] Jason Morrison, Kriging in a Parallel Environment [online], available: *http://citeseer.nj.nec.com/498126.html*.

[13] M. J. Quinn, *"Parallel Programming in C with MPI and OpenMP"*, pp. 93, 2004.

[14] Palmer, W.C. (1965). Meteorological Drought. *"Research Paper No. 45"*, US. Department of Commerce Weather Bureau, Washington D.C.