Title: # A Parallel Implementation of Kriging with a Trend

Author(s): Allyson Gajraj
Wayne Joubert
Jack Jones

Submitted to: For External Distribution

Los Alamos
National Laboratory

A Parallel Implementation of Kriging with a Trend

Allyson Gajraj and Wayne Joubert, Los Alamos National Laboratory, Jack Jones, Amoco

A. Summary

This paper describes the parallelization of the GSLIB[1] **ktb3dm** code. The code is parallelized using the message passing paradigm, Parallel Virtual Machine (PVM), under a Multiple Instructions, Multiple Data (MIMD) architecture. The code performance is analyzed using different grid sizes of 5x5x1, 50x50x1, 100x100x1 and 500x500x1 with 1, 2, 4, 8 and in some cases 16 processors on the Cray T3D supercomputer. The parallelization effort focused on the main kriging do loop. The results confirm that there is a substantial benefit to be derived in terms of CPU time savings (or execution speed) by using the parallel version of the code, especially when considering larger grids. Additionally, speed-up and scalability analyses show that actual speed-up is close to theoretical, while the code scales appropriately within the 1 to 16 processor range tested. The kriging of a quarter-million grid cell system fell from over 9 CPU minutes on one Cray T3D processor to about 1.25 CPU minutes on 16 processors on the same machine.

B. Introduction

B.1. Kriging

B.1a. Overview

Kriging is an interpolation algorithm which may be used as a means of obtaining the "best" (in a least squares error sense) estimate of a reservoir model. This is accomplished by minimizing the estimation variance from a prior covariance model through generalized linear regression techniques.[1] The prior covariance model is generally specified using variogram models.

Kriging uses the Minimum Variance, Unbiased Estimate (MVUE) principle. The estimation is *unbiased* in that the expected value of the error is set to zero, i.e.

$$E\{Z_0 - Z_0^*\} = 0 \tag{1}$$

where

Z_0 is the true value at location 0, and

Z_0^* is the associated estimated value.

The error variance may be specified as:

$$Var\{Z_0 - Z_0^*\} = \sum_i \sum_j a_i a_j C_{ij} \tag{2}$$

where

a_i, a_j are functions of the kriging weights, and

C_{ij} is the covariance between the values at locations i and j.

Minimizing the error variance is equivalent to setting:

$$\frac{\partial Var}{\partial a_i} = 0 \tag{3}$$

B.1b. Stationarity

Property estimation at unsampled locations using kriging (in fact *all* geostatistical analysis) is based on the assumption of stationarity. Simply stated, the region of stationarity defines that part of the reservoir, geologic structure, etc. over which the (geo)statistical model is valid. For 1st order stationarity, this implies that the 1st statistical moment (the expected value) is constant over the region of stationarity. Second order stationarity requires that covariance is a function of the lag distance only. If this is valid, we may relate the variogram and the covariance by the relationship:

$$C(\bar{h}) = C(0) - \gamma(\bar{h}) \tag{4}$$

where

$C(\bar{h})$ is the covariance for lag distance, $\bar{h}$

$C(0)$ is the zero-distance covariance (variance), and

$\gamma(\bar{h})$ is the variogram for lag distance, $\bar{h}$.

It should be pointed out that stationarity is a property of the random function (RF) model and not of the underlying spatial distribution.[1] Further, the assumption of a stationary RF model does not necessarily have to be applied to the entire data set but only to the search neighborhood -- since the kriged value is based on the values from that sub-set only. Thus the *local stationarity* assumed by kriging is often a viable assumption even in datasets for which global stationarity is clearly inappropriate.[2]

B.1c. Estimation vs Simulation

It should be noted that the resulting distribution represents an *estimate* of the truth and, while the conditioning information is honored, is a smooth image of the underlying reality.[3] Thus kriging is not an appropriate algorithm to generate input numerical models for flow simulation since the extreme permeability values, i.e. the flow paths and barriers are critical to the fluid flow response.[4] It may however be used as a pre-cursor or pre-processor to stochastic simulation.

B.2. Parallelization Considerations[5]

B.2a. What is Parallelism?

Parallelism may be defined as a strategy for performing large, complex tasks faster. This is done by:

- Breaking up the task into smaller tasks

- Assigning the smaller tasks to multiple workers to work on simultaneously
- Coordinating the workers
- Not breaking up the task so small that it takes longer to tell the worker what to do than it takes to do it.

B.2b. Steps for Creating a Parallel Program

- If starting with an existing serial code, debug the serial code completely.
- Identify the parts of the program which may be executed concurrently:
  - This requires a thorough understanding of the algorithm.
  - Exploit any inherent parallelism which may exist.
  - Restructuring of the program and/or algorithm may be required or an entirely new algorithm may be warranted.
- Decompose the program:
  - Functional parallelism - consisting of allocating different tasks to different processors
  - Data parallelism - allocation of sub-sets of the domain to different processors
  - Both
- Code development
  - Code may be influenced/determined by machine architecture
  - Choose a programming paradigm (e.g. message passing or data parallel)
  - Determine communication - how information is transferred among processors
  - Add code to accomplish task control and communications
- Compile, test, debug
- Optimization
  - Measure performance
  - Locate problem areas
  - Improve them

C. ktb3dm

C.1. Theory

This treatise, for the most part, follows that of reference 1.

C.1a. Kriging with a Trend Model

For estimating reservoir models for which the expected values (means) are stationary within the search neighborhoods, simple and (more usually) ordinary kriging algorithms are generally used. However, if there is a discernible trend in the data -- for example a thickening of the pay in a particular direction, a modification of the algorithm is used. The underlying RF model for a trend model is the sum of a trend component plus a residual:

$$Z(\bar{u}) = m(\bar{u}) + R(\bar{u}) \tag{5}$$

The trend component, $m(\bar{u}) = E\{Z(\bar{u})\}$, is usually modeled as smoothly-varying, deterministic function of the coordinates vector, $\bar{u}$, with the unknown parameters fitted from the data:

$$m(\bar{u}) = \sum_{l=0}^{L} a_l f_l(\bar{u}) \tag{6}$$

The $f_l(\bar{u})$'s are known functions of $\bar{u}$ while the $a_l(\bar{u})$'s are unknown parameters (hence the trend value is also unknown). The residual component is usually modeled as a stationary random function with a mean of zero, $E\{R(\bar{u})\} = 0$, and covariance, $C_R(\bar{h})$, where $\bar{h}$ is the lag separation vector.

The estimator is:

$$Z^*(\bar{u}) = \sum_{\alpha=1}^{n} \xi_\alpha(\bar{u}) Z(\overline{u_\alpha}) \tag{7}$$

and the system of equations for solving for the kriging weights, $\xi_\beta(\bar{u})$'s, is given by:

$$\begin{cases} \sum_{\beta=1}^{n} \xi_\beta(\bar{u}) C_R(\overline{u_\beta} - \overline{u_\alpha}) + \sum_{l=0}^{L} \mu_l(\bar{u}) f_l(\overline{u_\alpha}) = C_R(\bar{u} - \overline{u_\alpha}), \ \alpha = 1,\dots,n \\ \sum_{\beta=1}^{n} \xi_\beta(\bar{u}) f_l(\overline{u_\beta}) = f_l(\bar{u}), \ l = 0,\dots,L \end{cases} \tag{8}$$

where the $\mu_l(\bar{u})$'s are the $(L+1)$ Lagrange parameters associated with the $(L+1)$ constraints on the weights.[1] Thus stationarity is restored through the splitting of the variable into a trend (non-stationary) component and a residual (stationary) component.[6] As shown in Equation (8) above, the modeling is performed utilizing the covariance function of the residual or stationary component in the kriging equations to which the trend is then added back via the calculation of its local means. By convention, $f_0(\bar{u}) = 1, \forall \bar{u}$, thus the case L=0 corresponds to ordinary kriging with a constant (but unknown) mean, $m(\bar{u}) = a_0$.

C.1b. Kriging with an External Drift

This approach is used if the trend in the primary variable (i.e. the variable being kriged) can be related to that in a secondary variable. As an example, a relationship between the spatial trends in the permeability field (the primary variable) and those in the porosity field (the secondary variable) may have been observed or inferred. The porosity trend thus represents the "external drift" and this may be used in kriging the permeability values if:

- the external variable varies smoothly in space (otherwise the resulting kriging system may be unstable), and
- the external variable is known at all locations at which there are conditioning data for the primary variable as well as at the locations at which the primary variable is to be estimated.

The model is limited to two terms: $m(\bar{u}) = a_0 + a_1 f_1(\bar{u})$, where $f_1(\bar{u})$ is equal to the secondary (external) variable. Thus

$$E\{Z(\bar{u})\} = m(\bar{u}) = a_0 + a_1 y(\bar{u}) \tag{9}$$

where $y(\bar{u})$ is the secondary variable. Thus a linear rescaling of the units from the secondary variable is implied by Equation (9). Hence the estimate and the corresponding system of equations are given by:

$$Z_{KT}^{*} = \sum_{\alpha=1}^{n} \xi_\alpha(\bar{u}) Z(\overline{u_\alpha}) \tag{10}$$

$$\begin{cases} \sum_{\beta=1}^{n} \xi_\beta(\bar{u}) C_R(\overline{u_\beta} - \overline{u_\alpha}) + \mu_0 + \mu_1(\bar{u}) y(\overline{u_\alpha}) = C_R(\bar{u} - \overline{u_\alpha}),\ \alpha = 1,\dots,n \\ \sum_{\beta=1}^{n} \xi_\beta(\bar{u}) = 1 \\ \sum_{\beta=1}^{n} \xi_\beta(\bar{u}) y(\overline{u_\beta}) = y(\bar{u}) \end{cases} \tag{11}$$

Note that the cross covariance does not figure in this system (unlike in cokriging). In practice, the external drift is included as an additional trend term in the execution of the **ktb3dm** algorithm.

C.2. Overview

The **ktb3dm** suite of programs from the GSLIB library of geostatistical software is the most flexible of the GSLIB kriging program suites. It allows advanced 3-D point or block, simple or ordinary kriging, kriging with a polynomial trend with up to nine monomial terms, as well as kriging with an external drift.

The original suite of files required is:

- **ktb3dm.f**: an example driver for **ktb3dm**. This code is comprised of the main driver and the subroutine **readparm** which reads the input data. The readparm routine is called first, then ktb3d is called and the program terminates after that.
- **kt3d.inc**: an include file with maximum array dimensions.
- **kt3d.f**: some subroutines required by **ktb3dm**. These routines are:
  - <u>Subroutines</u>:

    - **super**: sets up the 3-D "super block" model and orders the data by super block number.
    - **picksup**: establishes which super blocks must be searched given that the point being estimated falls within the super block centered at (0,0,0).
    - **search**: searches through all the data which were tagged in the super block routine.
    - **matbld**: computes the terms of the kriging matrix which are common to ordinary and 'universal' kriging.
    - **setrot**: sets up the matrix to transform Cartesian coordinates to coordinates which account for the angles and anisotropies.
    - **ktsol**: solves the system of linear equations by Gaussian elimination with partial pivoting.
    - **sortem**: sorts a real array in ascending order; 1-7 associated arrays are also sorted based on the sorting order of the primary array.
  - <u>Functions</u>:
    - **cova3**: returns the covariance associated with a variogram model which is specified by a nugget effect and up to four different nested variogram structures.
    - **sqdist**: calculates the anisotropic distance between two points.
- **ktb3d.f**: the main kriging routine. This code contains the ktb3d subroutine which performs the actual kriging. There are three main parts to this subroutine:
  - Initializations: Routines called are:
    - setrot
    - super
    - picksup
    - cova3
  - Main Do Loop: The values are kriged here. The looping is over all (x-y-z) locations to be estimated. Routines called are:
    - search
    - matbld
    - ktsol
    - cova3
  - Output of results and debugging information.
- ktb3d.par: an example parameter file for **ktb3dm**

D. Parallelization Strategy

D.1. Overview

After considering the recommended parallelization steps outlined above, it was decided to use the node-only model of "crowd" computing -- multiple instances of a single program execute with one process taking over the non-computational responsibilities in addition to contributing to the computation itself.[7] In addition, that same processor also performs all initializations which is then shared with the other processors. So the approach is a Single-Program, Multiple-Data (SPMD) model. PVM under a MIMD architecture is the message passing paradigm used. Also, switching among alternative message passing paradigms is facilitated by the use of C preprocessor directives (#ifdef statements) built into the message passing routine.

D.2. Modifications to Serial Code

The serial code was developed using Fortran 77. Upgrading to Fortran 90 was undertaken to facilitate the use of dynamic storage allocation. The improved functionality of Fortran 90 also prompted the upgrade.[8] In general, the parallelization approach was modeled on the FALCON code.[9] Apart from the language upgrade, the major modifications of the serial code were in the areas of:

- Dynamic storage allocation for arrays (although not entirely). Dynamic storage allocation for some of the arrays was deemed unnecessary since these arrays were relatively small -- and would continue to be so regardless of the grid size -- and to "force" dynamic allocation would have entailed substantial coding modifications which would not have justified the additional effort in doing so.
- Parallelization of the main do loop within the ktb3d subroutine via a message passing interface. This part of the algorithm was the only part which was naturally parallelizable using the paradigm selected. It was also the only part of the code which was strongly a function of the grid size (see below).
- input/output (I/O). Broadly speaking, the input parameter file was unchanged, except that a switch to set the output format was added. In addition, the new include file explicitly defines the number of input conditioning data values, whereas this number was not defined in the serial version of the code. The serial code generated a geoEAS-type output for the kriged values. Our modified code allowed output in either the geoEAS-type format or in the FALCON format. The default output is the FALCON format since we found that to output a very large grid in the geoEAS format was much more computationally expensive.

E. Results

The parallelized code was tested on a variety of grid sizes, ranging from 25 to 250,000 grid block systems. In one case, using a 2,500 gridblock system, an external drift was included. The CPU times for different segments of the code were noted, using from 1 to 8 processors on the Cray T3D supercomputer (and up to 16 processors in the case of the 250,000 gridblock case). Figures 1 to 8 summarize the timing results obtained. As shown, for the smaller grids, there is little or no improvement in execution time by using more processors -- in fact for the 25-gridblock case we suffer an increase in CPU time. This is because the CPU time taken for the parallelized part of the code is relatively much smaller than the other parts of the code for small grids. With increase in grid size however, we begin to obtain some gains in execution time with additional processors, as the parallelized code requires increasingly more CPU time.

It should be pointed out that the results obtained substantially agreed with serial code runs on a Sun Sparcstation (using equivalent input datasets) in terms of the kriged block values as well as the variances of those estimates. There were slight differences which were determined to be due to the slight differences in sorting on the Sun and the T3D. There is no reason to favor one sorting set over the other because the differences pertain to alternate sorting of values which were equidistant from the value being kriged.

F. Optimization

Speedup. To further analyze the CPU timing, the breakdown of the time required for various parts of the main kriging do loop was determined for various grid sizes. Specifically, the times spent on calls to the search, matbld and ktsol subroutines were measured. It was found that about 45% of the main loop's CPU time was spent in the search subroutine, while about 30% was spent in the matbld routine with about 14% in ktsol. Figures 9 and 10 show these timing results for the case of a 250,000-block system, with 1 to 16 processors on the Cray T3D.

Amdahl's Law[5] defines the expected speed-up based on the parallelized fraction of the code. This equation is given by:

$$speedup = \frac{1}{1-f} \tag{12}$$

where f is the fraction of the code which can be parallelized. If we consider the number of processors performing the parallelization, the equation becomes:

$$speedup = \frac{1}{\frac{f}{n}+(1-f)} \tag{13}$$

where n is the number of processors used. Thus, it can be seen that Equation (12) is a limiting case of Equation (13), i.e. for an infinite number of processors the speedup is defined by Equation (12).

To determine the fraction of code which was parallelized, we ran the cases from 25 to 250,000 grid blocks on one processor and determined the fraction of the total CPU time required for the main do loop. This was an attempt at estimating the fraction to use in Amdahl's equation. As Figure 11 shows, this fraction increased monotonically and appeared to be leveling off at the high end. Since we were more interested in the larger grid sizes, we focused on the largest used and determined the fraction of CPU time spent on 1 processor for the main do loop. This was found to be 93.52%. This was then taken as the parallelized fraction of the code, f. To compare the speedup obtained to that expected from Equation (13), we additionally ran the 250,000-block case with 2, 4, 8 and 16 processors and determined the CPU times for the *total run* in each case. Taking the 16-processor run as an example, the CPU time for the total run was 75.020 CPU seconds, while for a single processor run that time was 551.017 CPU seconds. This means that we obtained a speedup of 7.34. Using 93.52% in Equation (13), we see that the expected speedup with 16 processors is 8.11. Thus our observed performance, while not optimum, is within the expected "ballpark". Further work is being done in improving this value.

Scalability. Should the parallel implementation be scaling appropriately, then there would be a inverse relationship between the number of processors used and the CPU time for taken for the parallelized part of the code -- the Main Do Loop; i.e.

$$Number\ of\ processors \propto \frac{1}{CPU\ Time} \tag{14}$$

Thus a plot of the inverse of CPU Time vs Number of Processors should be a straight line; as Figure 12 shows, this is approximately so.

G. Conclusions and Further Work

G.1. Conclusions

The conclusions from this work are:

- The parallelization of the **ktb3dm** code was successful, both in terms of replicating the results of the serial code -- which are assumed to be the correct values -- and also in terms of the objective of speeding up the algorithm, especially for the larger grids for which it will be used.
- The observed speed-up is within the range of the expected or theoretical value.
- The scalability analysis verifies that the code scales appropriately within the 1 to 16 processor range tested.

G.2. Further Work

In summary, the further work is:

- Based on the breakdown of the main do loop CPU time, almost one-half of the time is required for the search routine, hence this is a prime study target for improvement. This routine calls a sorting routine, in which a modified quicksort is currently used. We intend to try to improve this sorting algorithm.
- The next most expensive part of the main do loop is the matbld code. There seems to be little room for improvement there, but we will study this code as well.
- The ktsol code uses Gaussian elimination with partial pivoting. The matrix size is relatively small, with a rank of about 30. This code will also be studied to see if there may be room for improvement.

References

1. Deutsch, C.V. and Journel, A.G.: *GSLIB Geostatistical Software Library and User's Guide*, Oxford University Press, New York (1992).
2. Isaaks, E.H. and Srivastava, R.M.: *An Introduction to Applied Geostatistics*, Oxford University Press Inc., New York (1989).
3. Journel, A.G.: *Fundamentals of Geostatistics in Five Lessons. Volume 8 Short Course in Geology*, American Geophysical Union, Washington, D.C. (1989).

4. Deutsch, C.V.: "Annealing Techniques Applied to Reservoir Modeling and the Integration of Geological and Engineering (Well Test) Data", Ph.D. dissertation, Stanford University, Stanford, CA (1992).

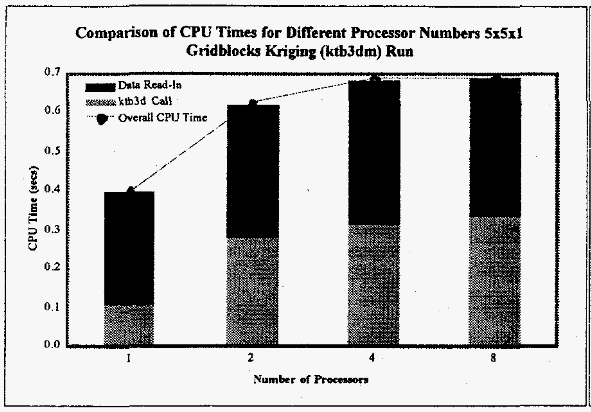
5. Maui High Performance Computing Center: "SP Parallel Programming Workshop Introduction to Parallel Computing", 02 July 1996 Revision by Blaise Barney, Copyright 1995 Maui High Performance Computing Center. All rights reserved. http://www.mhpcc.edu/training/workshop/html/parallel-intro/ParallelIntro.html.

6. Kelkar, B.G.: *Applications of Statistics to Reservoir Characterization*, self-published, Tulsa, OK (1989).

7. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing,* The MIT Press, Cambridge, Massachusetts (1994).

8. Adams, J.C., Brainerd, W.S., Martin, J.T., Smith, B.T. and Wagener, J.L.: *Fortran 90 Handbook Complete ANSI/ISO Reference*, Intertext Publications, McGraw-Hill Book Company, New York, New York (1992).

9. Joubert, W., Koch, K., Lubeck, O., van Bloemen Waanders, B., Stephenson, R. and Shiralkar, G.: "Next Generation Oil Reservoir Simulations", presented at the Spring 1996 Cray Users Group Meeting, Barcelona, Spain, March 11-15, 1996.



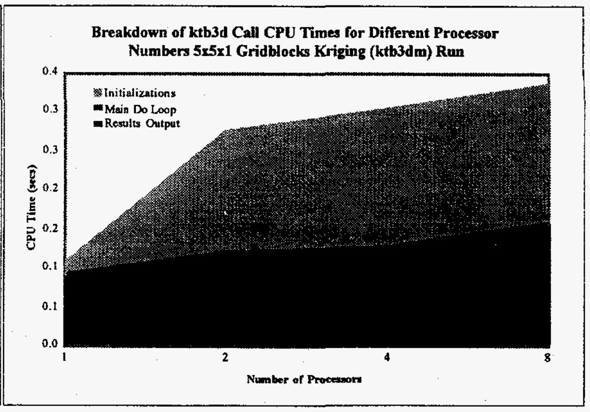
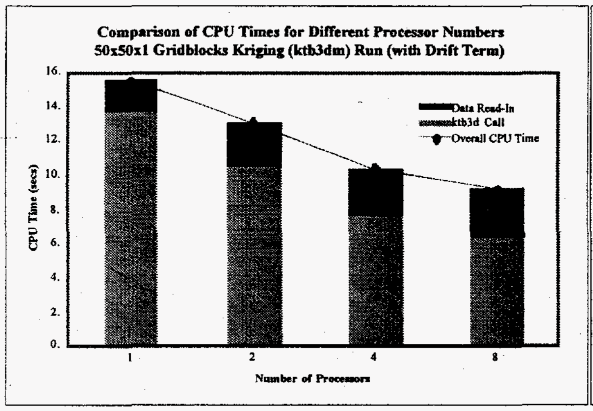


Figure 1: Comparisons of CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 5x5x1 Kriged Grid

Figure 2: Breakdown of ktb3d CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 5x5x1 Kriged Grid



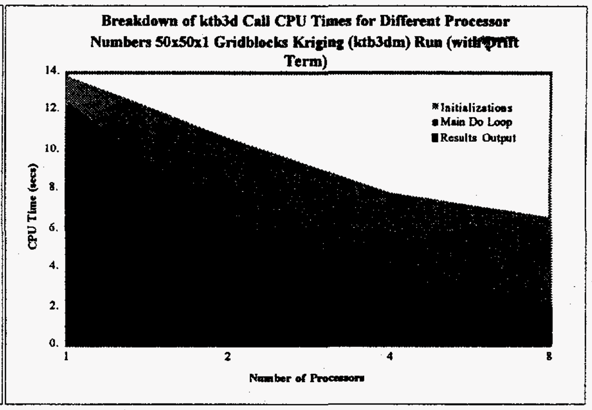


Figure 3: Comparisons of CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 50x50x1 Kriged Grid

Figure 4: Breakdown of ktb3d CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 50x50x1 Kriged Grid

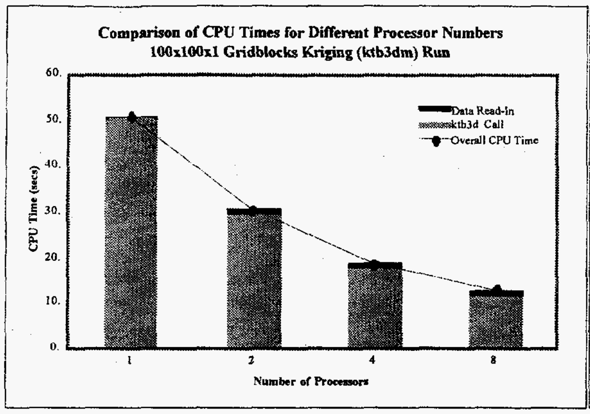


Figure 5: Comparisons of CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 100x100x1 Kriged Grid

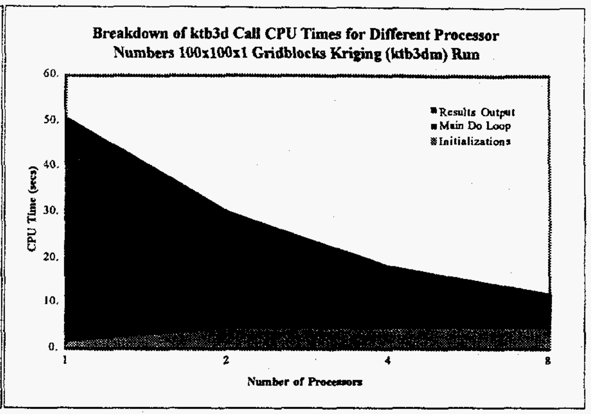


Figure 6: Breakdown of ktb3d CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 100x100x1 Kriged Grid

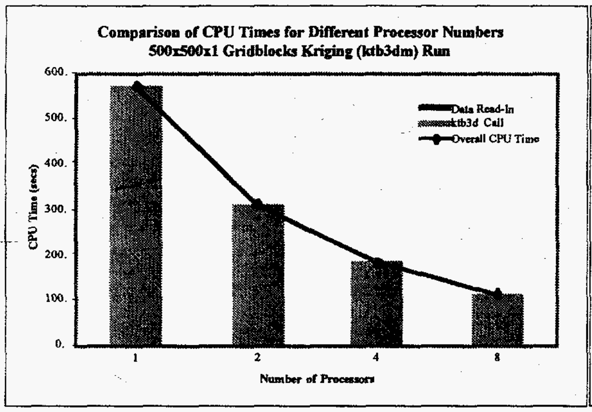


Figure 7: Comparisons of CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 500x500x1 Kriged Grid

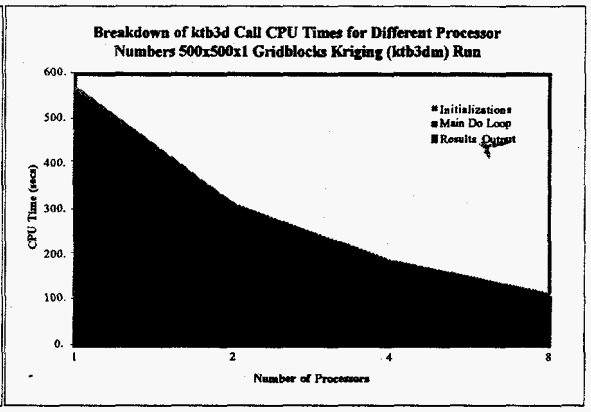


Figure 8: Breakdown of ktb3d CPU Time as a Function of Number of Processors Used in a Cray-T3D Run for a 500x500x1 Kriged Grid

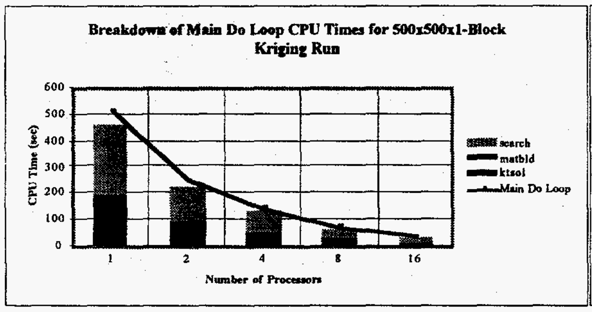


Figure 9: Breakdown of Main Do Loop CPU Times for 500x500x1-Block Kriging Run on a Cray T3D

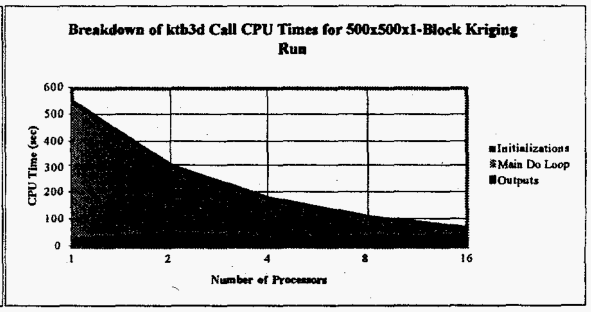


Figure 10: Breakdown of ktb3d Call CPU Times for a 500x500x1 Kriged Grid on a Cray T3D

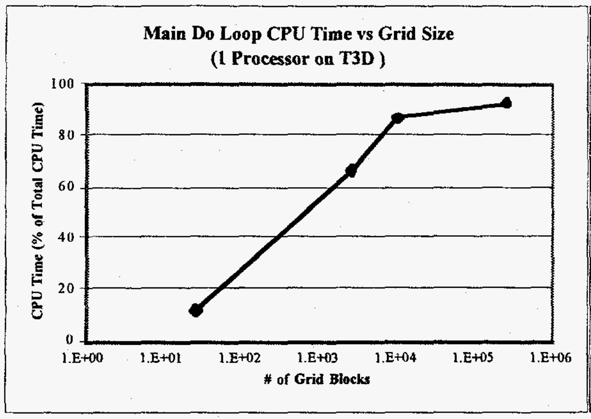


Figure11: Comparison of CPU Times for Main Do Loop for Different Grid Sizes Using 1 T3D Processor

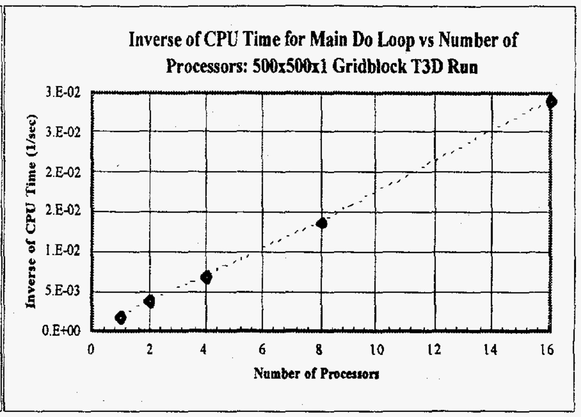


Figure 12: Plot of Inverse of the Main Do Loop CPU Time as a Function of the Number of Processors for the 250,000-Block Kriging Run on the T3D