On Parallelizing Universal Kriging Interpolation based on OpenMP

Tangpei Cheng, Dandan Li, Qun Wang*
School of Information Engineering
China University of Geosciences (Beijing)
Beijing, P.R.China
qunw@cugb.edu.cn

***Abstract*—Kriging is one of the important interpolation methods in geostatistics, which has been widely applied in engineering project. In this paper, we present an efficient method for the parallelization of universal Kriging interpolation on shared memory multiprocessors. By using OpenMP directives, we implement a portable parallel algorithm, which enables an incremental approach to add parallelism, without modifying the rest part of sequential code. To achieve optimal performance, the parallel grain size has been considered and analyzed. Numerical experiments have been carried out on two multicore windows workstations, the results of which demonstrate this method could enhance the overall performance significantly.**

Keywords- Kriging; spatial interpolation; parallel algorithm; OpenMP

I. INTRODUCTION

Kriging interpolation method is a group of geostatistical techniques to interpolate the value of a random field at an unobserved location from observations of its value at nearby locations. Kriging interpolation method has been widely applied in mining [1], hydrogeology [2], environmental science [3], black box modeling in computer experiments [4] and remote sensing [5] etc., which is also a computational bottleneck of these applications, preventing them from obtaining desirable performance. For this reason, research on parallel computing for Kriging interpolation has received considerable attention in recent years to improve the overall performance [6-10]. We note that most of these works are implemented on high-performance computer or distributed memory clusters by using MPI. Due to the emerging trends of multicore CPU recently, the shared memory multiprocessors, which support an incremental parallelization from serial program, are readily available. Therefore, the main objective of this work is to present a parallel version of universal Kriging interpolation method based on OpenMP, which could meet the intense demands on performance.

The outline of this paper is as follows. Section 2 gives a brief description of OpenMP programming paradigm. Section 3 gives an overview of the universal Kriging method and the OpenMP parallel implementation details on it. Experimental results as well as performance analysis are presented in Section 4 and Section 5 summarizes the work.

II. OPENMP PROGRAMMING PARADIGM

OpenMP is a shared-memory application programming interface (API), whose features are based on prior efforts to facilitate shared-memory parallel programming [11]. As shown in Fig. 1, OpenMP provides a fork-and-join execution model which supports an incremental approach to design parallel programs. Parallel work can be explicitly coded through the use of parallel regions, or implicitly obtained by work-sharing constructs, such as parallel loops. Compared to MPI, OpenMP applications are relatively easy to implement from the standard sequential code only by placing parallel directives around time consuming loops which do not contain data dependences, leaving the most part of the program unchanged. Another salient advantage of OpenMP lies in that it could achieve low latency and high bandwidth. Also, it adds fine granularity and enables increased and dynamic load balancing, which may lead to performance enhancement. More detail information about OpenMP can be found at the web site: http://www.openmp.org.

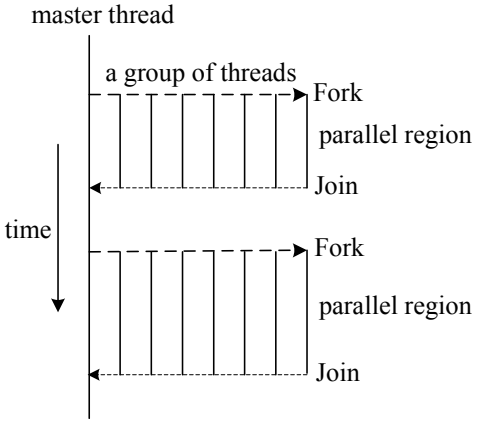

Figure 1. The fork-join model of OpenMP. Program begins execution as a single thread until a parallelization directive for a parallel region is found. Then the master thread creates a group of threads and the intensive computational work can be distributed among threads, without explicitly distributing the data.

978-0-7695-4110-5/10 $26.00 © 2010 IEEE
DOI 10.1109/DCABES.2010.14

III. PARALLELIZING OF UNIVERSAL KRIGING INTERPOLATION

A. Universal Kriging Interpolation

The basic premise of Kriging interpolation is that every unknown point can be estimated by the weighted sum of the known points:

$$Z_0^* = \sum_{i=1}^{n} \lambda_i^\circ Z_i \tag{1}$$

where Z_0^* represents the unknown point, Z_i refers to each known point and λ_i° is the weight given to it. The body of the Kriging algorithm is involved in the selection of the appropriate weights. For details about the theory of Kriging interpolation, readers may refer to [12] [13].

Universal Kriging assumes a general linear trend model. It includes the drift functions to calculate $m(x)$, which is the expectation of $Z(x)$. Considering

$$m(x) = a_0 + a_1 u + a_2 v + a_3 u^2 + a_4 uv + a_5 v^2 \tag{2}$$

where u, v are the coordinates of point x. Then we can get

$$\begin{aligned} &\sum_i \lambda_i^\circ (a_0 + a_1 x_i + a_2 y_i + a_3 x_i^2 + a_4 x_i y_i + a_5 y_i^2) \\ &= a_0 + a_1 x_0 + a_2 y_0 + a_3 x_0^2 + a_4 x_0 y_0 + a_5 y_0^2 \end{aligned} \tag{3}$$

In order to set up Eq. (3), the following equations can be gotten

$$\begin{cases} \sum_i \lambda_i^\circ = 1; & \sum_i \lambda_i^\circ x_i = x_0; \\ \sum_i \lambda_i^\circ y_i = y_0; & \sum_i \lambda_i^\circ x_i^2 = x_0^2; \\ \sum_i \lambda_i^\circ x_i y_i = x_0 y_0; & \sum_i \lambda_i^\circ y_i^2 = y_0^2 \end{cases} \tag{4}$$

Set

$$\sum_i \lambda_i^\circ P_l(x_i) = P_l(x_0), \quad (l = 0,1,2,3,4,5) \tag{5}$$

in which $P_l = \{1, x, y, x^2, xy, y^2\}$.

As

$$\begin{aligned} &E[(Z_0^* - Z_0)^2] = Var(Z_0) + \\ &\sum_i \sum_j \lambda_i^\circ \lambda_j^\circ c(x_i, x_j) - 2\sum_i \lambda_i^\circ c(x_i, x_0) \end{aligned} \tag{6}$$

where $c(x_i, x_j) = COV(Z_i, Z_j)$ and $c(x_i, x_0) = COV(Z_i, Z_0)$, based on Lagrange multiplier rule, we have

$$\begin{cases} \sum_j \lambda_i^\circ c(x_i, x_j) - \sum_{l=0}^{5} \mu_l P_l(x_i) = c(x_i, x_0) \quad (i = 1,2,\cdots,n) \\ \sum_i \lambda_i^\circ P_l(x_i) = P_l(x_0) \qquad\qquad (l = 0,1,\cdots,5) \end{cases} \tag{7}$$

which could be rewritten in the matrix form such as $Ax = b$ to calculate the value of λ_i° $(i = 1,2,\cdots,n)$. From Eq. (1), finally we could get the estimation of unknown points.

B. Parallel Algorithm on Shared-memory System

As stated in section 1, the computational steps of universal Kriging method which is based on covariance function could be schematically summarized as follow:

Step 1 calculating the distance between each known point;

Step 2 sorting the distances according to their values;

Step 3 grouping the sorted distances;

Step 4 constructing a variogram and the covariance function $c(x, y)$;

Step 5 computing covariance between each known point and then the coefficient matrix A;

Step 6 computing the inverse matrix of A;

Step 7 calculating the weights $[\lambda_1, \lambda_2, \cdots, \lambda_n]^T$ and then the estimate for each unknown point.

The first task in a parallel implementation is to identify the portions of the code where there is parallelism to exploit [14]. In scientific codes, the most common form of parallelism is data parallelism; and for shared-memory systems, it typically comes from the iterative loops.

In our work, an incremental approach based on OpenMP to parallelize the universal Kriging interpolation algorithm was carried out. By placing directives around time consuming loops which do not contain data dependences, the parallelization can be applied separately to individual parts, leaving the rest of source code unchanged. By profiling the execution of the sequential code of universal Kriging, it is noted that step 7, which involved a three-level nested loop, took up the most part of computational time. The program structure of step 7 can be briefly outlined as follow:

```
for i := 0 to NP − 1 do
  for j := 0 to NV − 1 do
    calculating the RHS
  endfor;
  …
  for j := 0 to NV + 5 do
    for k := 0 to NV + 5 do
      calculating the weights
    endfor;
  endfor;
  …
  for j := 0 to NV − 1 do
    calculating the estimates
  endfor;
endfor;
```

The variables i, j, and k are the loop counter of each for-loop. The variables NP and NV refer to the number of unknown and known points respectively. RHS denotes the right hand side of the linear equation.

The program block is largely a three-level nested for-loop, which mainly consists of three different computational steps. An important consideration on parallelization the code is to

decide the parallel grain size. Theoretically speaking, by analyzing the data dependency, each of the three loops could be parallelized by OpenMP directives. Especially, enlarge the grain size of a parallel program appears to bring better performance as it avoids frequent fork-join operation at the beginning of each iteration. However, the sequential code uses the same storage space to store the RHS for each unknown point, which means there is a loop-carried dependence in the outer loop. Parallelizing the outer loop, each RHS has to be made private explicitly and additional storage spaces are required, which may prevent the program from getting optimal performance. Therefore, making the inner loop parallel, which results in small grain size parallelism, is the best option for our case.

It is estimated that the second computational step of calculating the weights which is a two-level nested for-loop consumes the most part of execution time of the block. The OpenMP parallelization could be written as follow:

```
# pragma omp parallel for private(k, j)
shared(C_UK, A_UK, B_UK, NV)
  for j := 0 to NV + 5 do
    C_UK[j] := 0;
    for k := 0 to NV + 5 do
      C_UK[j] := A_UK[j*(NV + 6) + k]* B_UK[k];
    endfor;
  endfor;
```

The variables C_UK , A_UK and B_UK are the estimate, the inverse matrix and right hand side respectively.

IV. Performance Analysis

The numerical experiments were carried out on two win server 2003 (x64) workstations. Workstation 1: Intel Xeon E5310 1.6GHz (2CPU, 4 cores per each) and 2.0 GB memory; Workstation 2: Intel Xeon 5110 (2CPU, 2 cores per each) and 2.0 GB memory. The size of measured points and unknown points are 947 and 15719 respectively. The exponential variogram models and quadratic drift function were applied to the universal Kriging algorithm.

The first experiment was designed to find the hotspots in sequential program. The test was carried out only on Workstation 1. From Fig. 2, we observed that step 7 which is responsible for calculating the weights and estimates takes up the overwhelming majority of computational time. Another time consuming step is the computation of the inverse matrix. However, it is comparatively small when it is compared to step 7. Therefore, this experiment confirmed that the parallelization of step 7 is the primary concern of our work.

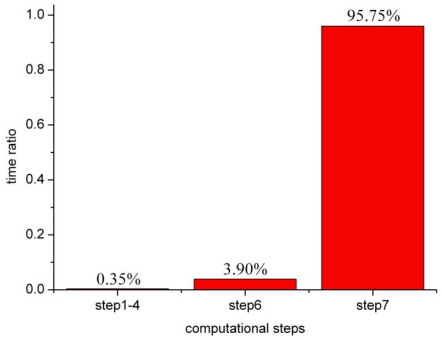

Figure 2. Time ratio for different computational steps

In the second experiment, the speedup versus different number of threads on two workstations is tested. From Fig. 3, it can be found that the speedup scales well from 1 to 4 threads for Workstation 1 and 1 to 2 threads for Workstation2, which demonstrates significant progress in reducing computational time and desirable parallel performance. However, we observed an obvious deviation from 4 to 8 threads for Workstation 1 and 2 to 4 threads for Workstation 2. From our analysis, this could be attributed to that the problem of cache coherence between two CPU in one workstation may degrade the parallel performance.

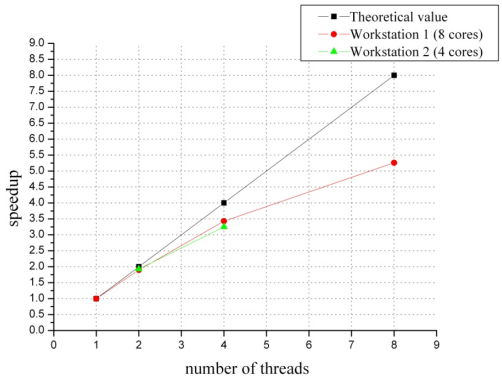

Figure 3. Speedup vs. number of threads

V. Conclusions

This paper describes an efficient fine-grain parallel scheme on shared-memory system, along with its implementation for universal Kriging interpolation method. As multiple-processors computers are currently much more affordable and available, and OpenMP is becoming the de facto standard for parallelizing applications, this ensures portability over a wide range of computers. In summary, we present a portable parallel implementation by using OpenMP directives, which enables an incremental approach to add parallelism, without modifying the rest part of sequential code. The experiment results demonstrate that the parallel scheme has achieved desirable performance. Further research will involve the parallel implementation of universal Kriging method on distributed shared memory architecture.

ACKNOWLEDGMENT

We would like to thank Prof. Nengxiong Xu for his help with the research. This research was partially supported by the Fundamental Research Funds for the Central Universities of China.

REFERENCES

[1] L. Wang, P. M. Wong, M. Kanevski, T. D. Gedeon, "Combining neural networks with kriging for stochastic reservoir modeling", IN SITU, 1999, vol. 23, pp. 151–169.

[2] P. Goovaerts, "Geostatistical approaches for incorporating elevation into the spatial interpolation of rainfa", JOURNAL OF HYDROLOGY, 2000, vol. 228, pp. 113–129.

[3] S. J. Jeffrey, J. O. Carter, K. B. Moodie, A. R. Beswick, "Using spatial interpolation to construct a comprehensive archive of Australian climate data", ENVIRONMENTAL MODELLING & SOFTWARE, 2001, vol. 16, pp. 309–330.

[4] D. R. Jones, M. Schonlau, W. J. Welch, "Efficient global optimization of expensive black-box functions", JOURNAL OF GLOBAL OPTIMIZATION, 1998, vol. 13, pp. 455–492.

[5] E. R. Richard, L. D. Jennifer, R. B. Louisa. "Kriging in the shadows: Geostatistical interpolation for remote sensing", Remote Sensing of Environment, 1994, vol. 49, pp. 32–40.

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

[7] K. E. Kerry, K. A. Hawick, "Kriging Interpolation on High-Performance Computers", Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, LNCS, Springer Berlin / Heidelberg, 1998, vol. 1401, pp. 429–438.

[8] A. Gebhardt, "PVM kriging with R", In Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, 2003.

[9] Nadja Samonig, Parallel computing in spatial statistics, Master's thesis, University Klagenfurt, 2001.

[10] J. A. Pedelty, J. T. Morisette, J. A. Smith, J. L. Schnase, C. S. Crosier, T. J. Stohlgren, "High Performance Geostatistical Modeling of Biospheric Resources", Eos Trans. AGU, 2004, vol. 85.

[11] C. Barbara, J. Gabriele, V. P. Ruud, "Using OpenMP: Portable Shared Memory Parallel Programming", The MIT Press, 2007.

[12] C. Noel, "The origins of kriging", Mathematical Geology, 1990, vol. 22, pp. 239–252.

[13] L. S. Michael, "Interpolation of spatial data: some theory for kriging", Springer Series in Statistics, 1999.

[14] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White, Sourcebook of Parallel Computing, San Francisco, CA: Elsevier Science, 2003.