Research on CUDA-based Kriging Interpolation Algorithm

Meng Li
Vision Computing and Visualization Laboratory, School of Computer Science and Technology,
University of Science and Technology of China,
Hefei, China
lmroot@mail.ustc.edu.cn

Lanfang Dong
Vision Computing and Visualization Laboratory, School of Computer Science and Technology,
University of Science and Technology of China,
Hefei, China
lfdong@ustc.edu.cn

***Abstract*-Three-dimensional geological model can describe the types of geological information efficiently, express a variety of topological relations among geological phenomena intuitively. Kriging interpolation algorithm is an important spatial interpolation method of three-dimensional geological modeling, but every grid point needs to compute augmented matrix and solve equations, so it costs too much time. With the modeling scale and drilling data quantity increasing rapidly, we need to make Kriging algorithm parallel imperatively. However parallel machine has many disadvantages- high cost for example, we propose to use CUDA which is multi-threaded parallelism and low lost GPU-using to implement Kriging interpolation algorithm in this paper, then optimize the memory access and so on according to the executive features of CUDA. We make high speedup and low running time. Compared with the single-threaded CPU implementation, the speedup of multi-threaded CUDA implementation achieves more than 110.**

Keywords-Three-Dimensional Geological Modeling; Kriging; CUDA;Parallel; Optimization

I. INTRODUCTION

For modeling the three-dimensional geological model, we generally first build the stratum surfaces which are formed by the data interpolated from limited discrete space drilling data points, and then use the stratum surfaces to build three-dimensional geological model. Kriging interpolation algorithm [1] is a relatively well-known method, but because every grid point must calculate the coefficients of the augmented matrix and solve equations, therefore the computation requires very long time when the grid count of stratum surfaces is large. Most of the existing acceleration algorithms [2, 3] use the parallel machine, but the hardware cost of these methods is relatively very high. The computing capability of multi-threaded parallel GPU becomes more and more powerful, ratio of performance to price becomes higher and higher in recent years, so we propose to use CUDA to implement Kriging interpolation algorithm.

CUDA (Compute Unified Device Architecture) is a general purpose parallel computing architecture proposed by NVIDIA Corporation. It uses the GPU(s) to process data which can be expressed as the problems of parallel computing. It is able to run tens of thousands of threads parallel, increasing the interpolation speed as a result. However, the running performance of CUDA programs are affected by the number of threads executing simultaneously in each SM (Stream Multiprocessors), memory access latency and many other factors, so needing to do careful optimization.

II. BASIC PRINCIPLES OF KRIGING INTERPOLATION ALGORITHM

The Kriging interpolation algorithm used in this paper is Ordinary Kriging [4, 5], shown in Fig.1. Ordinary Kriging computes estimate value $Z'(x^*)$ of non-sampling point x^* from the value $Z(x_1),\ldots,Z(x_n)$ of sampling point $x_1,\ldots,x_n$.The calculation formula is

$$Z'(x^*)=\sum_{i=1}^{n}\lambda_i Z(x_i)\,. \tag{1}$$

Here the weights λ solved by the following equations (μ is a lagrange multiplier):

$$\begin{cases}\sum_{i=1,j=1}^{n} C(x_i-x_j)\,\lambda_i = C(x^*-x_j)\\ \sum_{i=1}^{n}\lambda_i = 1\end{cases}. \tag{2}$$

Matrix form is:

$$\begin{pmatrix} C(x_1-x_1) & \cdots & C(x_1-x_n) & 1\\ \vdots & \ddots & \vdots & \vdots\\ C(x_n-x_1) & \cdots & C(x_n-x_n) & 1\\ 1 & \cdots & 1 & 0\end{pmatrix}\begin{pmatrix}\lambda_1\\ \vdots\\ \lambda_n\\ -\mu\end{pmatrix}= \begin{pmatrix} C(x^*-x_1)\\ \vdots\\ C(x^*-x_n)\\ 1\end{pmatrix}. \tag{3}$$

Denoted as:

$$\boldsymbol{A\lambda} = \boldsymbol{B}. \quad (4)$$

Here $C(h)$ is the stationary covariance function (h is lag distance):

$$C(h) = C(0) - \gamma(h). \quad (5)$$

Here $\gamma(h)$ is the variogram. Theoretical model of variogram used in this paper is exponential model (a is range, c is sill.):

$$\gamma(h) = c.[1 - \exp(-\frac{3h}{a})]. \quad (6)$$

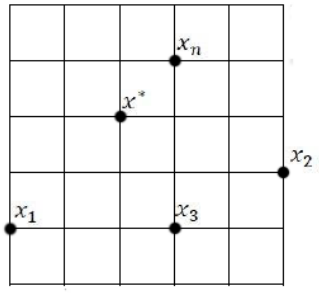


Figure 1. Kriging interpolation

III. BASIC PRINCIPLES OF CUDA PROGRAMMING

CUDA extends the C language by allowing the programmer to define the C function using __global__ declaratory called kernel (running in the GPU(s)) which will be called by N-CUDA threads simultaneously. The CUDA program is divided into two parts: the implementation of the CPU on the host side and the implementation (known as kernel) of the GPU in the device (graphics card) side. GPU has multiple SMs, each of them creates, manages, schedules, and executes threads in groups of 32 parallel threads (called Wrap). Multiple threads are organized into blocks (Block), multiple threads blocks are organized into grids (Grid). Kernel called method in the host code is "Kernel <<<Grid, Block>>> (parameters)". When the number of threads in the block is not a multiple of 32, the rest of the threads will be divided into a separate wrap.

Memory space [6] of CUDA includes global, local, shared, texture, and registers. In terms of access speed, Registers&Shared Memory > Contant&Texture Memory > Global&Local Memory. When writing CUDA programs, you should make full use of registers and shared memory. Automatic variables in kernel function normally are stored in the registers, but they will be stored in local memory if you exceed the number of available registers per-thread. Global memory accesses of half-wrap or wrap thread when certain conditions [7] are met can be combined into one transaction. But data transfer between the host memory and video memory goes through the PCI-Express X16 bus, has high latency, so we must minimize data transfer between the host and the device.

IV. CUDA IMPLEMENTATION AND OPTIMIZATION OF KRIGING INTERPOLATION ALGORITHM

With the dimensions of geological modeling increasing rapidly, it is imperative to parallel the Kriging interpolation algorithm. Although multi-core processors are already quite popular in PC, only in this way can they handle 12 threads concurrently at most (Intel Core i7 990X (Extreme Edition)). Feasible method is to use multi-CPU parallel machine(s) or many core multi-threads GPU(s) to parallel Kriging interpolation algorithm. Although the previous approach is feasible, it requires high initial investment. NVIDIA GPU is able to process lots of threads concurrently, most users can afford, so paralleling Kriging interpolation algorithm using GPU is more desirable. OpenGL and CUDA can also be combined [8] in order to fully enhance the efficiency of the geological model visualization.

The implementation of Kriging algorithm is basically consistent between CUDA program and CPU program. First we calculate the coefficients of matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, second we solve the equations, and finally we calculate the estimated value of grid points in stratum surfaces using the linear combination method after got the weights. Since the calculation process of every grid point has nothing to do with the other grid points, is only related with the sampling values of every stratum surface and coordinates of drilling points, so you can take advantage of multi-threaded feature of CUDA to calculate many grids at the same time.

Mesh size of stratum surface is denoted as M * N, the number of stratum surfaces is denoted as H in the experiments. A series of experiments was performed to monitor the performance gain of implementation and improvement of our CPU/CUDA program (both using C language).

A. Experiment 1

CPU program: Every grid point is required to calculate matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, solve equations using the gauss method, and calculate the estimate values of non-sampling grid points from the sampling values of the known drilling points after got the weights. We use single-thread to calculate.

CUDA program: We use M * N threads (thread Block size is (m, n, 1), thread Grid size is (M/m, N/n, 1)) to calculate all non-sampling grid points surface by surface of individual stratum surface. Every thread calculates one grid point in a stratum surface mesh. The calculation process is the same as the CPU program. One stratum surface mesh is calculated every time called the kernel, the number of stratum surfaces is H, and so we need to call the kernel H times. CUDA Kernel 1 is:

```
1   __global__ void Kriging(...)
2   {
3    Prepare_A
4    Prepare_B
5    GaussSolve
6    Linear_Combine
7   }
```

B. Experiment 2

After careful analysis, we find that matrix $\boldsymbol{A}$ is only related with the coordinates of drilling points, does not care which stratum surface it is, and does not change in the whole Kriging interpolation calculation process. $\boldsymbol{A}$ is double counting in Experiment 1, improved as follows:

CPU/CUDA program: First the matrix $\boldsymbol{A}$ is calculated, consideration is saving the matrix $\boldsymbol{A}$, then we assign it to every grid point in the calculation process, the remaining step is the same as experiment 1.

As a result, the performance gain of this improvement is very high compared with experiment 1. CUDA Kernel 2 is:

```
1    Prepare_A
2    __global__ void Kriging(...)
3    {
4     memcpy
5     Prepare_B
6     GaussSolve
7     Linear_Combine
8    }
```

C. Experiment 3

CPU program: Same as the CPU part of Experiment 2.

CUDA program: The CUDA program is based on two-dimensional thread Block in the previous two experiments, that is to say we call the kernel H times, requiring multiple time-consuming data transfer operations between host and device. We can use the three-dimensional thread Block (Block size (m, n, H), Grid size (M/m, N/n, 1)), that is M * N * H threads calculating all geological model grid points simultaneously. This method only need to call the kernel once, but the inadequacy of this method will increase the scheduling overhead in the GPU, so the performance gain of using three-dimensional Block is not very high compared with CUDA program of experiment 2.

D. Experiment 4

We also find that the weights are related with the coordinates of the drilling points, independent of the corresponding stratum surface sampling value by getting a closer look, improved as follows:

CPU/CUDA program: Based on experiment 2, after we get the corresponding weights of one non-sampling grid point, the estimate value of same grid position can be obtained in all H stratum surfaces, no need of solving the weights of the other stratum surfaces. Because we are able to get the value of all H stratum surfaces, the CUDA kernel only needs to be called once. In other words this improvement can reduce computing matrix $\boldsymbol{B}$ and solving equation using gauss method H-1 times.

Therefore, the performance gain of this improvement is obvious compared with experiment 2. CUDA Kernel 3 is:

```
1    Prepare_A
2    __global__ void Kriging(...)
3    {
4     memcpy
5     Prepare_B
6     GaussSolve
7     for (k=0;k<H;k++)
8     {
9      Linear_Combine
10    }
11   }
```

E. Experiment 5

CPU program: Same as the CPU part of experiment 4.

GPU program: Because the coordinates of drilling points, the known sampling values of every stratum surface and left matrix $\boldsymbol{A}$ is constant, we can put them in the constant memory with cache in the GPU. The access speed of cache is as fast as register. But the cache hit rate is an important factor influencing the speed of calculation for the time being. Every constant cache entry is designed to store multiple consecutive words, so we can store the location coordinates of drilling points in an array whose elements as a struct [9].This will increase the cache hit rate by continuous access, but reduce the utilization rate of the cache entries as a result.

Moreover, we are able to read the elements used several times in a loop which are located in global memory into automatic variables before accessing, thus we can increase the ratio of access to calculating, but the register usage per-thread now increases certainly. Fortunately, the register usage is not a limiting factor to parallelism for our kernel.

The experiment results show that the performance of optimized code has small increase compared with CUDA program of experiment 4, mainly because most of the time every thread is still spent on computing matrix $\boldsymbol{B}$ and solve equations.

F. Experiment 6

Based on experiment 5, we find that the using of double precision in the calculation will increase the computing time greatly, therefore we change the accuracy of the data from double to float. We also use fast math functions-"__expf" for example which are a slight decrease in accuracy using hardware implementation by GPU [10] in CUDA program. After export the experimental data, we compare them with the data from experiment 5, find no significant drop in accuracy, however the performance of the algorithm has been greatly improved for the CUDA implementation.

V. EXPERIMENTAL RESULTS

Geological data used in this paper has grid size 50 * 90, 10 stratums, and 10 wells in every stratum surface. PC configuration is:

Graphics Card: NVIDIA GeForce GTX 460 768M
CPU: Intel E5500@2.8GHz
OS: Windows 7 Ultimate
Platform: Visual Studio 2008 SP1+CUDA 3.2 RC

Size of thread Block and thread Grid is decided by the size of stratum surface mesh, the number of registers used by the kernel and many other factors. We choose the size of two-dimensional thread Block is (25, 5, 1), the size of thread Grid is (50/25, 90/5, 1) in this paper. Every Block has 125 threads, each SM schedules execution threads by a wrap, and as a result the last wrap has only three threads empty. The size of three-dimensional thread Block is (10, 9, 11), the size of thread Grid is (50/10, 90/9, 1) in experiment 3.All the results reported here are the mean values of five or more records of the same experiments.

Fig.2 shows the computation time decline gradually by improving and optimizing the CPU/CUDA program, under the same time experiment, computing time of CUDA program is much less than the computing time of CPU program. After having been optimized, the time of CUDA program required decline further (from experiment 4 to 6), indicating that the optimization is very important to CUDA program.

To demonstrate the performance gain of our CUDA implementation, we test speedup of different grid count of stratum surfaces. By zooming length and width of the mesh 2x, 4x, 6x, 8x, 10x at the same time, grid count increases to 4.95 million from 49500.Because the location coordinates of drilling points are also zoomed, their constraints on the grid points has not been changed.

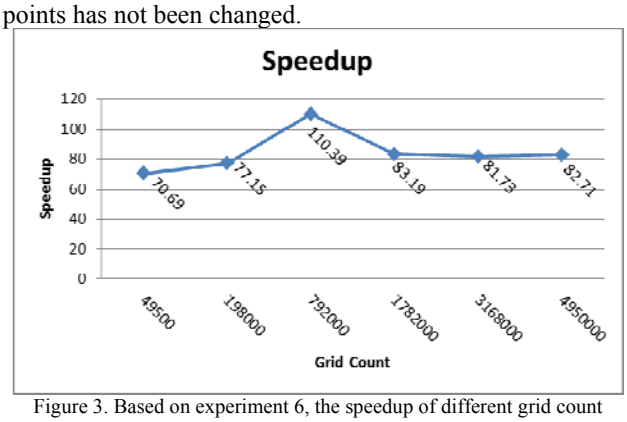


Figure 3. Based on experiment 6, the speedup of different grid count

As shown in Fig.3, with the growth of stratum surface grid count, speedup increases quickly. However, due to hardware limitations operational capability, speedup becomes stable after reached the peak, stable speedup is about 80, but the maximum speedup is more than 110.

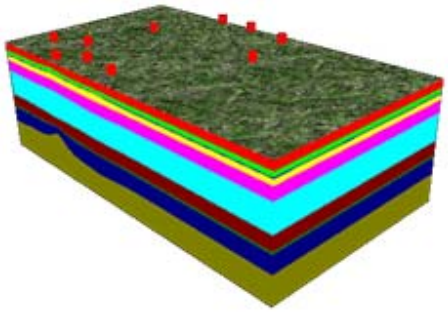


Figure 4. Visualization of geological model

VI. CONCLUSION

In this paper, we have taken significant steps towards implementing the Kriging interpolation algorithm using CUDA, and we also optimize the CUDA program in order to obtaining higher speedup and lower running time than the CPU implementation. The visualization (shown in Fig.4) of interpolated data verifies that the calculation of CUDA program using GPU is fast, true and reliable. As for future work, we are interested in paralleling the other types of Kriging using CUDA.

ACKNOWLEDGMENTS

This work is supported by the Knowledge Innovation Project of Chinese Academy of Sciences under Grant No. KJCX1-YW-21. The authors wish to thank professor Detang Lu from University of Science and Technology of China for the useful suggestions and providing help.

REFERENCES

[1] R.Franke, “Scattered Data Interpolation: Tests of Some Methods,” Mathematics of Computation,vol.38,1982, pp.181-200.

[2] K.E.Kerry and K.A.Hawick, “Kriging Interpolation on High-Performance Computers,” Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, 1998,pp.429-438, doi:10.1007/BFb0037170.

[3] J. A. Pedelty, J. T. Morisette, T. J. Stohlgren, M. A. Kahlkan, J. A. Smith and J. L. Schnase, “ High performance geostatistical modeling of biospherice resources in the Cerro Grande Wildfire Site, Los Alamos, New Mexico, and Rocky Mountain National Park, Colorado,” In Proceedings of the 2004 NASA Earth Science Technology Conference (ESTO’03, College Park, June), 2003.

[4] J.L.Mallet,“Geomodeling,” New York: Oxford University Press,2002.

[5] X.Li,M.Hu,Z.Wang, “Constructing Digital Elevation Model based on Kriging method,” Application and Research of Computers, supplement,2008.

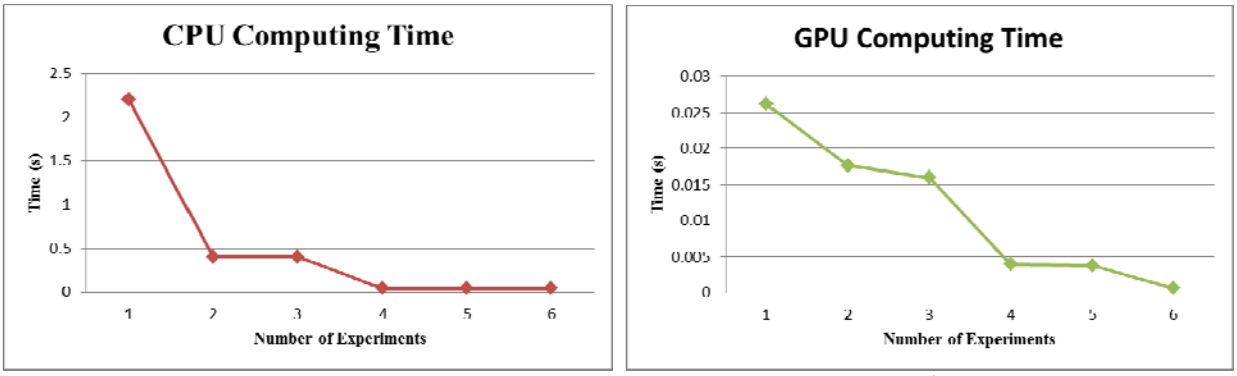
[6] S.Zhang,Y.Chu,“CUDA GPU Computing for High Performance,”Beijing: China Water Power Press,2009.

[7] NVIDIA, “CUDA C Programming Guide,”Version 3.2,2010.

[8] NVIDIA, “CUDA/OpenGL Fluid Simulation,” 2008.

[9] D.B.Kirk and W.M.Hu, “Programming Massively Parallel Procelssors:A Hands-on Approach,” Burlington:Morgan Kaufmann,2010.

[10] NVIDIA,“CUDA C Best Practices Guide,” Version 3.2,2010.



a b

Figure 2. Computing time of the experiment:(a) CPU Computing Time (b) GPU Computing Time