Fast computation of bare soil surface roughness on a Fermi GPU

Article *in* Computers & Geosciences · May 2015

DOI: 10.1016/j.cageo.2015.05.013

Xiaojie li
Chinese Academy of Sciences
27 PUBLICATIONS **78** CITATIONS
SEE PROFILE

Changhe Song
Xi'an Electronic Science and Technology University
6 PUBLICATIONS **72** CITATIONS
SEE PROFILE

Sebastian Lopez
Universidad de Las Palmas de Gran Canaria
133 PUBLICATIONS **933** CITATIONS
SEE PROFILE

Jose F. Lopez
Universidad de Las Palmas de Gran Canaria
33 PUBLICATIONS **133** CITATIONS
SEE PROFILE

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Fast computation of bare soil surface roughness on a Fermi GPU

Xiaojie Li[a,b,*], Changhe Song[c], Sebastián López[d], Yunsong Li[c], José F. López[d]

[a] Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences, Changchun 130102, China
[b] Space Science and Engineering Center, University of Wisconsin, Madison, WI 53706, USA
[c] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China
[d] University of Las Palmas de Gran Canaria, Institute for Applied Microelectronics, Las Palmas de Gran, Canaria, Spain

ARTICLE INFO

Article history:
Received 27 August 2014
Received in revised form
2 April 2015
Accepted 20 May 2015
Available online 27 May 2015

Keywords:
Bare soil surface roughness
Correlation length
Graphics Processing Unit (GPU)

ABSTRACT

Surface roughness is an important factor in bare soil microwave radiation for the observation of the Earth. Correlation length and standard deviation of surface height are the two statistical parameters that describe surface roughness. However, when the number of data points is large, the calculation of surface roughness parameters becomes time-consuming. Therefore, it is desired to have a high-performance computing facility to execute this task. A Graphics Processing Unit (GPU) provides hundreds of computing cores along with a high memory bandwidth. To carry out a parallel implementation of the algorithms, Compute Unified Device Architecture (CUDA) provides researchers with an easy way to execute multiple threads in parallel on GPUs. In this paper, we propose a GPU-based parallel computing method for 2D surface roughness estimation. We use an NVIDIA GeForce GTX 590 graphics card to run the CUDA implementation. The experimental input data is collected by our in-house surface roughness tester which is designed based on the laser triangulation principle, giving sample data points of up to 52,040. According to the experimental results, the serial CPU version of the implementation takes 5422 s whereas our GPU implementation takes only 47 s, resulting a significant 115 × speedup.

1. Introduction

Surface roughness is a description of the randomness or irregularity of the microtopography of a terrain. The standard deviation of surface height (σ) and the surface correlation length (l) describe the statistical variation of a surface height relative to a reference surface for a random component. Measurement of surface roughness is one of the key topics in soil erosion research in the sense that is an important parameter in order to determine soil hydrological characteristics and soil properties. While observing the earth using a microwave radiometer, surface roughness represents a key factor to analyze (Zheng and Zhao, 2010; Mittal and Singh 2010; Hong, 2010). Several research instances can be found in literature on determining surface roughness parameters. Seppke et al. (2010) and Wang et al. (2011) utilized satellite images for inversion of soil roughness parameters, got σ=0.3–3 cm, cl=3–35 cm. Oh et al. (2007) and Thomas (2003) developed a retrieval method of soil moisture and surface roughness from backscatter measurements of vegetation canopy. Moreover, Tang et al. (2009) applied a digital image processing method to obtain roughness parameters of triangular prisms, the measurement results showed that roughness less than 0.35 mm over an area of 60 cm × 60 cm could be recognized. The contact method was also employed by Rosario et al. (2008) and Šařec et al. (2007) to measure the surface roughness parameters for different soils. Moreover, some authors of the current papers presented some surface roughness testing apparatus and the corresponding testing methods (Li et al., 2012) that features rapid testing speed and high testing precision, requires no manual work and obtains three-dimensional parameters for the surface. By a single scan, a total number of data points in 500 mm × 600 mm range of 40,000–100,000 can be obtained. However, calculations of the correlation length and the standard deviation of a surface height based on these data points present a huge workload and require a large computing time. Therefore, a high performance computing environment becomes imperative to overcome this issue.

With the advent of Graphics Processing Units (GPUs), inherently parallel problems can be effectively accelerated. Modern GPUs possess hundreds of parallel processor cores and are capable of executing tens of thousands of threads in parallel. Unlike traditional CPUs, GPUs are not optimized for single thread performance. Instead, they are optimized for executing a large number of threads simultaneously. Mateo-Lázaro et al. (2014) presented an application that visualized three-dimensional geological structures with digital terrain models by a GPU. Yang et al. (2014)

* Corresponding author at: Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences, Changchun 130102, China.
E-mail address: lixiaojie@iga.ac.cn (X. Li).

http://dx.doi.org/10.1016/j.cageo.2015.05.013
0098-3004/© 2015 Elsevier Ltd. All rights reserved.

implemented reverse time migration using GPU programming, combining staggered finite difference scheme with convolutional perfectly matched layer (CPML) boundary condition. Huang et al. (2013) focused on an innovative implementation of the Direct Sampling method which exploits the benefits of Graphics Processing Units (GPUs) to improve computational performance.

In this paper, we develop a GPU-based method to calculate the soil surface roughness parameters with a total of 52,040 data points. We reach a speedup factor of 115 ×, including GPU I/O interface, as compared to a single threaded CPU version. This paper is organized as follows. Section 2 outlines the methodology followed in our study. The GPU implementation is described in detail in Section 3. The experiment at the field test site and the GPU based data calculation of the surface roughness parameters are described in Section 4. Finally, Section 5 concludes the findings of the paper.

2. Methodology

When an electromagnetic wave is incident on the boundary of a random surface present between two semi-infinite media, a portion of the incident energy is scattered backwards and the rest is transmitted into the lower medium. The scattered amplitude along the specular direction decreases while those along the other directions increase as the surface gets rougher. When surface roughness reaches a certain value, the surface emissivity becomes irrelevant to the observation angle. Surface roughness influences the surface scattering process and thus affects the radiation brightness temperature received by a radiometer antenna, which is an important factor in the analysis of soil microwave radiation.

Single point laser triangulation principle generally uses direct-type and oblique-type structures. Direct type's principle is shown in Fig. 1.

In Fig. 1, the emitted laser beam is focused by the convergent lens and strikes the surface of the measured object. The scattered light is incident on the receiving lens and, after that, it is detected by the photoelectric detector. According to the similarity of the triangles AOB and A′OB′

$$\frac{a}{b-h'\mathrm{ctag}\theta}=\frac{z}{h'/\sin\ \theta} \tag{1}$$

where h' is the spot on an imaging surface displacement for the measured plane displacement z; a is the distance along the optical axis of the lens, from the incidence point on the surface to the intersection point on the receiving lens in mm; b is the distance in millimeters from the receiving lens behind main surface to the center of the imaging surface; and θ is the angle in degrees between the laser beam optical axis and the receiving lens optical axis.

Eq. (1) leads to

$$z=\frac{ah'}{b\ \sin\ \theta-h'\cos\theta} \tag{2}$$

Fig. 2 shows the surface roughness tester developed by microwave remote sensing lab, which belongs to the Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences. Based on the laser triangulation principle (Xu et al., 2002), a linear laser is emitted on the tested object and the image of the reflected light is received by camera array. A measurement of the 3D area can be obtained through an image processing algorithm and a coordinate transformation. For a single scan, the scanned regions are 500 mm, 600 mm and 400 mm with corresponding directional resolutions of 0.78 mm, 1.00 mm, and 0.83 mm in x direction, y direction and z direction respectively. For each directional measurement, the scanning time is 20 s.

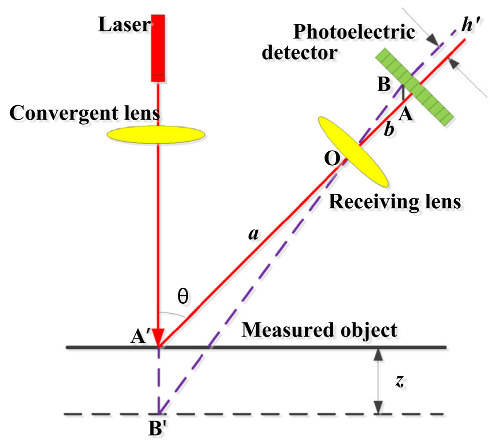

Fig. 1. Single point laser triangulation measurement principle.

This completes the apparatus based measurements. The next step is to perform measurements on the surfaces in a 3D local coordinate system. Based on this data sample, and once these measurements have been completed, the calculation of the soil surface roughness parameters can be carried out.

Soil surface can be characterized mainly by three parameters; the standard deviation of surface height (rms height, σ), the surface correlation length (l), and the auto correlation function. The σ describes random surface characteristics, while the correlation length and the correlation function describe periodicity of the surface (Ulaby et al., 1987). Fig. 3(a) and (b) shows random height variations superimposed on either a periodic or a flat surface.

The standard deviation of a surface height (σ) and the surface correlation length (l) are defined as follows (Ulaby et al., 1987):

a. Standard deviation of a surface height:
Consider a surface in the x–y plane whose height at a point (x, y) is $z(x, y)$ above the x–y plane. For a statistically representative segment of the surface, with dimensions L_x and L_y, centered at the origin, the mean height of the surface is

$$\bar{z}=\frac{1}{L_xL_y}\int_{-L_x/2}^{L_x/2}\int_{-L_y/2}^{L_y/2}z(x,y)dx\,dy \tag{3}$$

and the second moment is

$$\bar{z^2}=\frac{1}{L_xL_y}\int_{-L_x/2}^{L_x/2}\int_{-L_y/2}^{L_y/2}z^2(x,y)dx\,dy \tag{4}$$

The standard deviation of the surface height σ, is then expressed as

$$\sigma=\left(\bar{z^2}-\bar{z}^2\right)^{\frac{1}{2}}=[\frac{1}{N-1}(\sum_{i=1}^{N}(z_i)^2-N\left(\bar{z}\right)^2]^{\frac{1}{2}} \tag{5}$$

where $\bar{z}=\frac{1}{N}\sum_{i=1}^{N}z_i$ and N is the number of samples.

b. Surface correlation length

For the discrete case, the normalized autocorrelation function is given by

$$\rho(j)=\frac{\sum_{i=1}^{N-j+1}(z_i-\bar{z}_i)(z_{i+j-1}-\bar{z}_{i+j-1})}{\sum_{i=1}^{N}z_i^2} \tag{6}$$

In the result produced by the microwave radiometer and the radar detector, the scattering values are spread over a 2D area. Hence, it makes sense to calculate the 2D correlation coefficient. The equation was given by

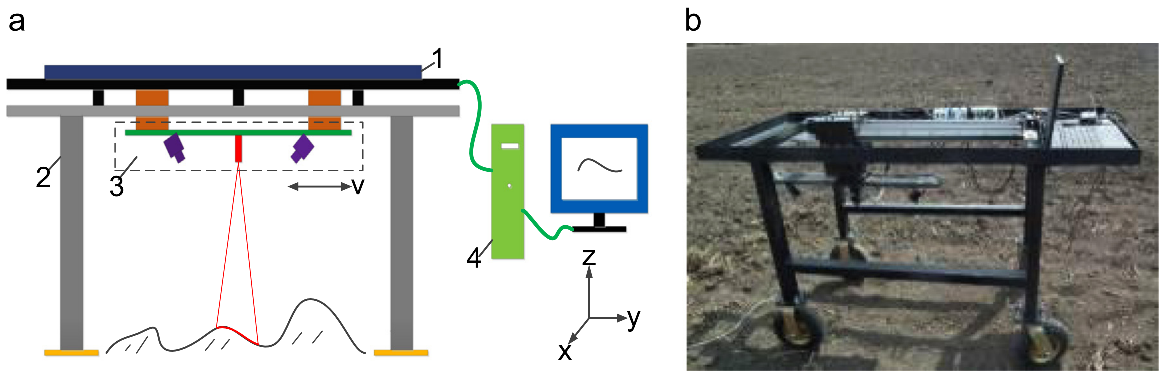

Fig. 2. (a) Structure of surface roughness tester (1. Power Mobile Unit, 2. Adjust the support unit, 3. Imaging unit, 4. Computer). (b) Picture of surface roughness tester.

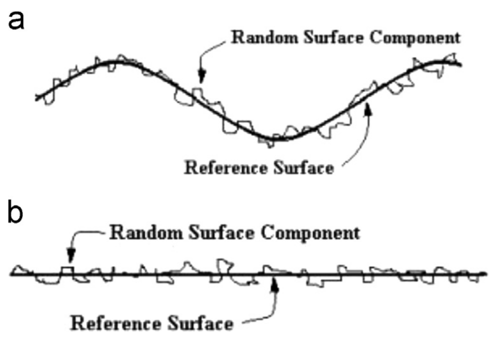

Fig. 3. Random height variations superimposed on a (a) periodic surface and (b) flat surface.

$$\rho(j) = \frac{\sum_{i=1,k=1}^{M} (z_i - \bar{z}_i)(z_k - \bar{z}_k)}{\sum_{i=1}^{M} (z_i - \bar{z}_i)^2} \Big|_{\sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} \in (j-0.5,\ j+0.5)} \tag{7}$$

Where j is the value that we give to calculate the different correlation coefficients.

The surface correlation length l usually is defined as the displacement x' for which $\rho(x')$ is equal to $1/e$, i.e. $\rho(l)=1/e$. The unit of l is mm.

The correlation length of a surface provides a reference for estimating the statistical independence of two points on that surface. If the two points are separated by a horizontal distance greater than l, then their heights may be considered to be (approximately) statistically independent one to each other.

3. GPU implementation

3.1. C code implementation

The C code implementation computes the correlation coefficients in four independent steps, which are described as follows:

i. Calculate the distances between any two points (x_i, y_i) and (x_k, y_k) in the X–Y plane, where $i, k=1,2,\ldots N$.
ii. Consider any distance values within the range of $(j-0.5, j+0.5)$ to be integer j, record the total number of point pairs that have a distance j and denote the number as M, where $j=1,2,\ldots 300$ in our study case. When recording the point pairs, the z value of each point is also accumulated as it will be used in the next step.
iii. Compute the mean height value of the point pairs that have a distance of j. This step has to wait for M to be ready in step (ii).
iv. Calculate the correlation coefficients based on Eq. (7).

As can be seen, the distances between any (x_i, y_i) and (x_k, y_k) can be computed in parallel. The determination of whether the distance is j can also be executed in parallel. However, the C code failed to explore the parallel feature. In order to improve the processing speed of the calculation of bare soil surface roughness, we propose to utilize the GPU to explore the parallelism and accelerate the C code.

3.2. CUDA implementation

The CUDA is selected to implement the GPU-based surface roughness parameter computation scheme because of its easy deployment and high computing performance (Sharma et al., 2009). Moreover, it is best suited to NVIDIA GPU implementations (NVIDIA, 2011; Mielikainen et al., 2012; Satria et al., 2012) as compared to other computing techniques such as OpenCL and Direct Compute.

A generic CUDA program consists of three parts. The first part refers to the data transfer from the host CPU memory to the dedicated GPU memory which is also called device or global memory. The second part involves code execution on GPU, performed inside a kernel. The third part refers to the copying of output data from GPU back to the host CPU memory. The launch of these three parts is sequential. Execution inside a GPU kernel occurs in parallel.

The parallel execution in GPUs is accomplished by executing thousands of threads that is the basic atomic unit of parallelism. Threads are organized into a three-level hierarchy as shown in Fig. 4. The grid is the highest-level element, which presents the resources of the kernel. Blocks are subparts of the grid and constitute the multiprocessor-level elements. Each block has its private shared memory which cannot be accessed by other blocks but can be shared internally among all the threads which reside within that block. The threads are stream-processor level elements and they possess their own local memory and registers.

The Intel i7 970 processor is selected as our CPU platform and the NVIDIA GTX 590 board as the GPU platform. Specifications of both are listed in Table 1.

In the process of converting C codes to CUDA codes, it is worthwhile to pay attention to the handling of the grid point (i, j). In CUDA codes, the loops for spatial grid points (i, j) are replaced by index computations using thread and block indices:

$$i == threadIdx.x + blockIdx.x{*}blockDim.x \tag{8}$$

$$j == threadIdx.y + blockIdx.y{*}blockDim.y \tag{9}$$

where *threadIdx* and *blockIdx* are the thread index and the block index, respectively, *blockDim* is the dimensional size of a block. Thus each grid point (i, j) will be handled by a thread in the proposed CUDA code.

We used gcc -lstdc++ -O3 for the original C codes, with GNU C

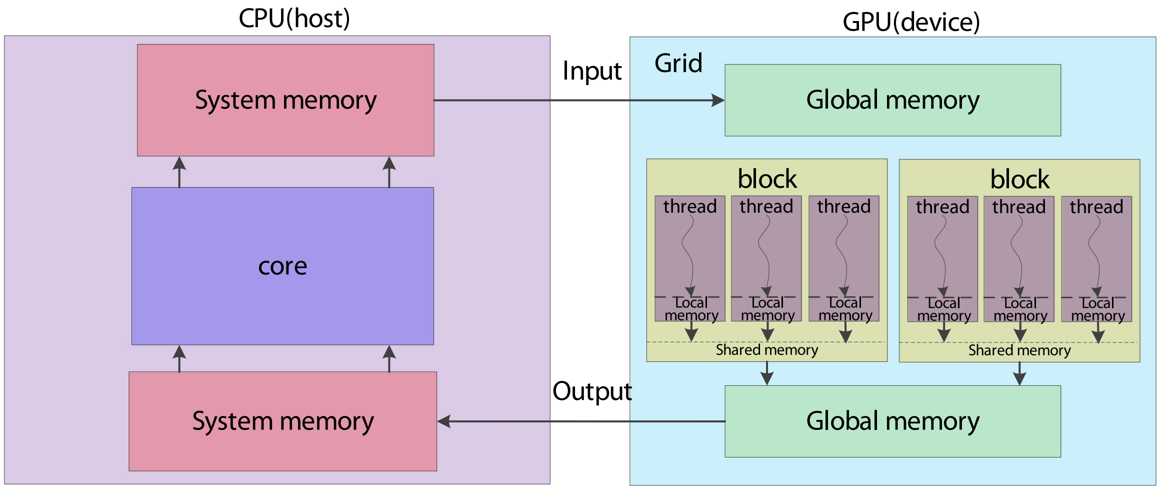

Fig. 4. The three-level hierarchy.

Table 1
Device parameters of the GPU and CPU.

	CPU	GPU
Model	Intel i7 970	NVIDIA GTX 590
Number of cores	6	16 multiprocessors (MP) 32 stream processors (SP) per MP
Frequency (GHz)	3.2	1.215
L1 cache (KBytes)	6 * 64	48 or 16
L2 cache (KBytes)	6 * 256	768
Shared memory (KBytes)	n/a	16 or 48

library version 2.12-1.107. The CUDA codes were compiled using nvcc (NVIDIA CUDA compiler) version of 5.5 and executed on a NVIDIA GTX590 GPU board with compute capability of 2.0. The compiler options thus are −O3 −arch sm_20 -fmad=false -m64 −maxrregcount 63 −restrict. The maxregcount value 63 indicates the number of registers allowed per thread, which was randomly picked at this moment. Besides, the thread block size was chosen as 64 at this stage.

In our experimental study, the original C implementation took 5422 s, while the first unoptimized CUDA version consumed 1132 s, which means, a simple parallel acceleration of GPU has brought a speedup of 5, as is listed in Table 2.

3.2.1. Further improvement by allowing more L1 cache space

As has been mentioned above, the GPU scheme was evaluated by running the CUDA codes on one GPU of the GTX590 board. As is listed in Table 1, the data cache could be configured as 16 KB software-managed cache called shared memory and 48 KB hardware cache (L1) or the other way around. By default, the L1 cache is set to be 16 KB. Since the shared memory is not used in the current version, the L1 cache is thus configured to be up to 48 KB for a better cache of the data by setting the command "cudaFuncCachePreferL1" in the CUDA codes. It turned out that the larger L1 cache space helps to speed up the CUDA codes as shown in Table 3.

Table 2
The time usage of different computation platforms.

	Time usage (s)	Speedup
CPU	5422	1
GPU	1132	5 ×

Table 3
The time usage when more L1 cache is enabled.

	Time usage (s)	Speedup
CPU	5422	1
GPU	818	7 ×

As can be seen, the time usage of the GPU-based implementation was reduced to 818 s from 1132 s. The larger L1 cache helps to improve the hit rate, therefore, the final speed is improved. It should be noted that the cost of enabling larger L1 cache is smaller than shared memory space. However, there is no shared memory usage in the current version and even when the shared memory is used in our next version, the 16 KB shared memory is large enough for the method we used based on our evaluation.

3.2.2. Further improvement by using the shared memory

To calculate the mean values of z_i and z_k in a more efficient way on a GPU, we further developed a method that adopts the faster shared memory of a GPU, which is discussed as follows.

If the distance is r, the corresponding thread maintains two local variables to store the local sum of z_i and z_k. When all these threads are finished, the local sums are buffered to the shared memory. Then the local sums in the shared memory are accessed and summed up again by each of these threads to calculate D_1 and D_2. As a result, each of these threads have a local copy of D_1 and D_2. This process is also illustrated in Fig. 5. The value of N is calculated in the same way as with D_1 and D_2. Thus the mean values are calculated by dividing D_1 and D_2 by N.

Subsequently the distances are computed and checked again to determine which threads should continue in the accumulation processes in Eq. (9). The accumulation process also employs the method as we developed to achieve D_1 and D_2. The computations

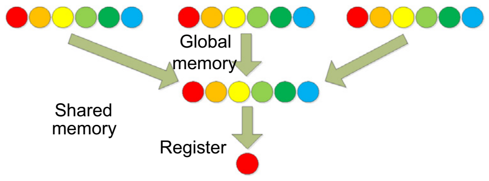

Fig. 5. The summation process of heights of different points; different colors indicate that these points are processed by different threads. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
The time usage when the shared memory is used.

	Time usage (s)	Speedup
CPU	5422	1
GPU	124	44 ×

Table 5
The time usage when memory coalescing is used.

	Time usage (s)	Speedup
CPU	5422	1
GPU	83	65x

for different distances are assigned to be performed in different blocks.

The distance between (x_i, y_i) and (x_k, y_k) is calculated and checked twice in our GPU-based implementation. As a matter of fact, we have also considered saving the determined result into an array which can be used directly in the latter steps. However, saving the result into an array occupies additional memory space. An array with a size of $M(M-1)/2$ is needed for M points. When M is large, the method will suffer from lack of memory space. In addition, the read/write time of the array will also be involved. We therefore prefer more computation instead of more memory accesses.

After using the shared memory of the GPU, the speedup is shown in Table 4. As can be seen, the speedup of the GPU-based implementation has been boosted into 44 × with the adoption of the shared memory. This is achieved in part by the buffering scheme we designed above and the high accessing speed of the shared memory as it is located on chip. The authors thus argue that the use of shared memory is very important to achieve high speedups in the development of a GPU program.

3.2.3. *Further improvement with memory coalescing*

The input data is a three-dimensional data array which was originally stored in a text file. The location of a point (x, y) is stored in a line along with its height information $z(x, y)$. Different points are stored in different lines of the file. It is easy to see that the original layout of the data does not satisfy the memory access pattern required by the coalesced memory access, as the access of the x values for different points are interlaced. The uncoalesced memory access causes a waste of the bandwidth of the global memory and thus affects the final performance of our GPU-based implementation.

To solve this problem and enable coalesced memory accesses, the array is reorganized into a column wise fashion in the global memory when reading from the text file, which means the x values of all points are stored continuously in the memory, followed by all the y values and then all the z values for every point. A pictorial layout of the data structure is shown in Fig. 6 and the result for this version is listed in Table 5.

As can be seen in Table 5, the GPU time usage was reduced to 83 s and the speedup reached 65 × after we reorganized the data layout. This is because when several threads of the GPU try to access the data values of several points that are next to each other, the coalesced memory access technique is able to converge these accesses into a big one so as to reduce the global memory access overhead. It should also be noted that it is also important to ensure the memory addresses are aligned when allocating the memory spaces to enable fully coalesced memory access.

3.2.4. *Further improvement with best block size per grid*

As the CUDA architecture automatically schedules the execution of different blocks, we simply set the number of blocks to be the number of required distance calculations. In our experiment, the number of blocks is 300, which is the number of distances with different starting points. However, the decision on the number of threads in one block, i.e. the block size, is a more complicated task.

Firstly, the threads inside a block are scheduled based on warps, which consists of 32 threads for the GPU we used in this study. Therefore, the block size should be a multiple of 32 so that all warps are fully loaded. Secondly, an optimized block size should enable a high occupancy of the multiprocessor, which is mainly decided by the number of active warps. Theoretically, the multiprocessor is able to serve up to 6 active blocks and each block is able to serve up to 8 active warps. When the actual number is larger than these values, some blocks or warps will be inactive until the multiprocessor is free. Besides, the size of shared memory used in each block and the number of registers used in each thread also affect the number of active warps in a multiprocessor. As a result, we set the block size experimentally.

We carried out a test with varied block size for 52,040 data points, and the results are presented in Fig. 7. It is observed that the best performance is achieved when the block size is 512. Note that the GPU speed increases greatly with larger block sizes when the block size is below 128. This is because small block sizes could not keep the multiprocessor busy. Therefore, when the block size is larger, the multiprocessor become busier and the program runs faster. It is also observed that the GPU speed become similar for the block size between 128 and 1024. This is due to the multiprocessor has already become fully busy when the block size is 128. When the block size is further increased, the number of active warps does not change any more. However, some memory access overhead can be concealed in the switching between active blocks and inactive blocks. Therefore, the best performance appears when the block size is 512.

With this configuration, the final time usage and speedup are shown in Table 6. As can be seen, the GPU time usage is only 47 s compared with the 5422 s of CPU time usage. By optimizing the block size, the speedup increased from 65 × to 115 × .

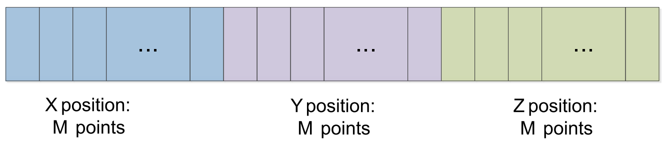

Fig. 6. A pictorial layout of the data structure.

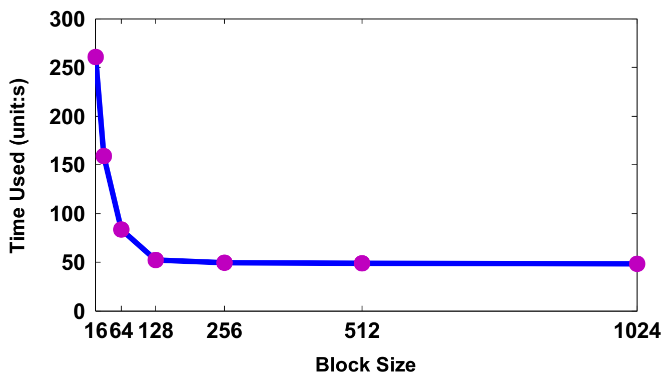

Fig. 7. Runtime for 52,040 data points with different block size.

Table 6
the time usage when the block size is set to 512.

	Time usage (s)	Speedup
CPU	5422	1
GPU	47	115 ×

(a) show the bare soil surface roughness for locations *A* and *B* with location *B* rougher than location *A*. Figs. 8(b) and 9(b) show the 3D surface data of locations *A* and *B* that means the 3D digitization results can be obtained by using our instrument. Different colors of points in the figures indicate different heights of the surface.

The 2D soil surface roughness can be easily obtained from the

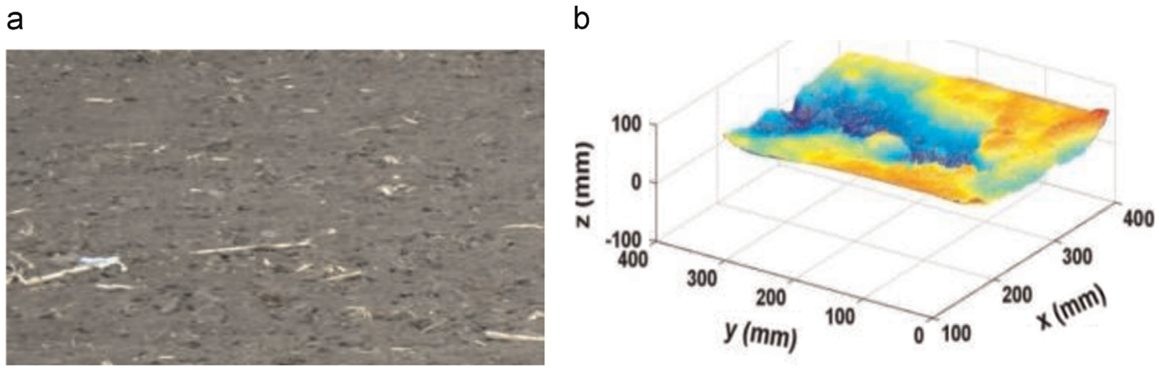

Fig. 8. (a) Bare soil location *A*. (b) The original data of location *A*. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

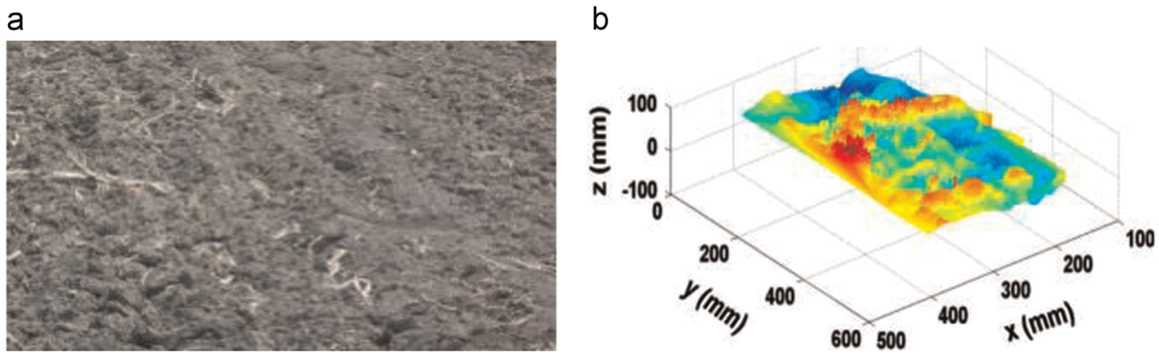

Fig. 9. (a) Bare soil location *B*. (b) The original data of location *B*. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

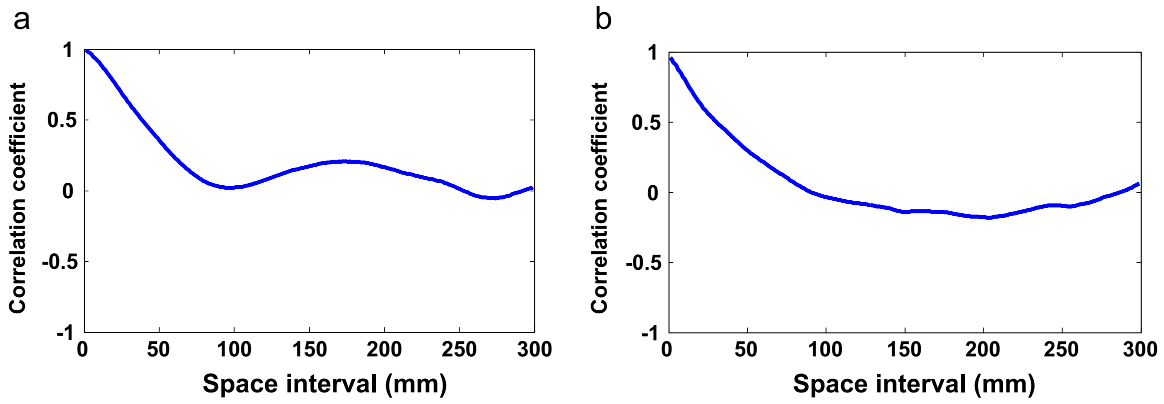

Fig. 10. (a). 2D correlation coefficients of location *A*. (b). 2D correlation coefficients of location *B*.

4. Experiment and results

The field test was carried out at the Jilin Agricultural University, China in April 19, 2012 using a surface roughness tester developed by the microwave remote sensing lab of the Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences.

This was used to measure the surface state of the two farmlands. The pictures of location and the original data by the surface roughness tester are presented in Figs. 8 and 9. Figs. 8(a) and 9 3D coordinates by using our GPU program. The correlation coefficients for location *A*, with a total number of 52,040 points, are displayed in Fig. 10(a). A correlation length of 49.09 mm is obtained in this case. The standard deviation of the surface height is 12.70 mm. The GPU correlation coefficients for the location *B*, with a total number of 53,869 points are displayed in Fig. 10(b). A correlation length of 43.21 mm is obtained in this case and the standard deviation of the surface height is 16.85 mm. The standard deviation of location *B* is bigger than location *A*, and the

Table 7
The CPU- and GPU-based results for the calculation of roughness parameters.

Data points	CPU time (s)	GPU time (s)	Speedup
47,958	4847	40	121
52,040	5422	47	115
53,869	5772	51	113
66,864	8486	80	106

correlation length of location B is smaller than location A, which means location B is rougher than location A.

In order to test the applicability of our GPU program, we have also used additional datasets. The timing results calculated at different data points using CPU and GPU are shown in Table 7 along with the speedup for the GPU based implementation.

As observed in Table 7, the data points are from 47,958 to 66,864, which is a typical number of points for our instrument. The results show, the computing time on GPU are reduced to less than 80 s from more than 2 h on a CPU, it means the GPU implementation is suitable for on-site test. And great improvements in the speedup which are more than 100 × are achieved using the GPU architecture.

This implementation takes into account the data transfer overhead while calculating the speedup. Unlike many other implementations, the data transfer time only takes a small portion of the overall time since the input is composed by $3N$ floating point numbers, where N is the number of points, and the output is composed by j floating point numbers, where j is the number of height values used for calculation. Compared with the computation that has a complexity of N^2, the data transfer only takes a very short time. Therefore, the performance lost owing to data transfer overheads is very small.

5. Conclusion

In this paper, we have developed a GPU-based parallel computing method for 2D surface roughness estimation. It was implemented in CUDA and executed on an NVIDIA GeForce GTX 590 card. Calculating the distance between any two points can be carried out by threads. In our implementation, we designed the computations for different appointed distances such that they are performed in different blocks. This method was applied to the data collected by our surface roughness tester on April 2012 based on the laser triangulation principle. For a total of 52,040 data points, the serial CPU version of the implementation took 5422 s whereas the GPU implementation took 47 s, leading to a significant 115 × speedup.

This result can be used in Advanced Integrated Equation Model (AIEM) to calculate the soil backward scattering coefficient. Therefore, it has an important role in the study of microwave detecting ground radiation.

Acknowledgments

This work was supported by the National Natural Science Foundation of China, No. 41201335 and was contributed in Space Science and Engineering Center, University of Wisconsin–Madison USA. The authors would like to thank Dr. Bormin Huang for his excellent suggestions in our research.

References

Hong, S., 2010. Surface roughness and polarization ratio in microwave remote sensing. Int. J. Remote Sens. 31 (10), . 2709–2716.
Huang, T., Li, X., Zhang, T., Lu, D., 2013. GPU-accelerated direct sampling method for multiple-point statistical simulation. Comput. Geosci. 57, 13–23.
Li, X., Zhao, K., Zheng, X., 2012. Development of surface roughness tester based on laser triangulation method. Trans. Chin. Soc. Agric. Eng. 28 (8), 116–121.
Mateo-Lázaro, J., Sánchez-Navarro, J.Á., García-Gil, A., Edo-Romero, V., 2014. 3d-geological structures with digital elevation models using GPU programming. Comput. Geosci. 70, 138–146.
Mielikainen, J., Huang, B., Huang, H., Goldberg, M., 2012. Improved GPU/CUDA based parallel Weather and Research Forecast (WRF) Single Moment 5-Class (WSM5) cloud microphysics. Appl. Earth Obs. Remote Sens. vol. 5 (4), 1256–1265.
Mittal, G., Singh, D., 2010. Critical analysis of microwave specular scattering response on roughness parameter and mois-true content for periodic rough surfaces and its retrieval. Prog. Electromagn. Res. 100, 129–152.
NVIDIA, 2011. NVIDIA CUDA C Programming Guide.
Oh, Y., Hong, J.-Y., Jung, S.-G., 2007. Retrieval of soil moisture and surface roughness from backscatter measurements of vegetation canopy. In: Asia-Pacific Microwave Conference, pp. 1–3.
Rosario, G.-M., Maria Cruz, D.-A., Requejo, S., Tarquis Alfonso, A., Maria, A., 2008. Multifractal analysis of soil surface roughness. vol. 7. Soil Science Society of America, USA, pp. 512–520.
Šařec, P., Šařec, O., Prošek, V., Čížková, K., 2007. Laser profilometer testing by laboratory measurements. Res. Agric. Eng. 53 (1), 1–7.
Satria, M., Huang, B., Hsieh, T., Chang, Y., Liang, W., 2012. GPU acceleration of tsunami propagation model. Appl. Earth Obs. Remote Sens. 5 (3), 1014–1023.
Seppke, B., Dreschler-Fischer, L., Heiming, J.-A., Wengenroth, F., 2010. Fast derivation of soil surface roughness parameters using multi-band SAR imagery and the integral equation model. In: Proceedings of the 2010 International Conference on Pattern Recognition. IEEE Computer Society, pp. 3931–3934.
Sharma, B., Thota, R., Vydyanathan, N., Amit, K., 2009. Towards a robust, real-time face processing system using CUDA-enabled GPUs. Presented at the 2009 International Conference on High Performance Computing HPC, pp. 368–377.
Tang, X., Xiao, H., Ding, H., Liu, J., 2009. Surface roughness measurement based on image processing and image recognition. Comput. Simul. Mod. Sci., 91–96.
Thomas, H., 2003. Measuring surface soil parameters using passive microwave remote sensing. The ELBARA Field Experiment.
Ulaby, F.-T., Moore, R.-K., Fung, A.-K., 1987. Microwave remote sensing active and passive. Radar Remote Sens. Surf. Scatt. Emiss. Theory 2, 819–833.
Wang, S.-G., Li, X., Han, X.-J., Jin, R., 2011. Estimation of surface soil moisture and roughness from multi-angular ASAR imagery in the Watershed Allied Telemetry Experimental Research (WATER). Hydrol. Earth Syst. Sci. 15, 1415–1426.
Xu, Z.-Q., Sun, C.-K., Ye, S.-H., 2002. Reverse Engineering Technology. Beijing, pp. 13–16.
Yang, P., Gao, J., Wang, B., 2014. RTM using effective boundary saving: a staggered grid GPU implementation. Comput. Geosci. 68, 64–72.
Zheng, X., Zhao, K., 2010. A method for surface roughness parameter estimation in passive microwave remote sensing. Chin. Geogr. Sci. 20 (4), 345–352.