Parallel Computing Method of Commonly Used Interpolation Algorithms for Remote Sensing Images

Minghu Fan, Xianyu Zuo, and Bing Zhou

***Abstract*—Parallel computing is a common method to accelerate remote sensing image processing. This article briefly describes six commonly used interpolation functions and studies three commonly used parallel computing methods of the corresponding nine interpolation algorithms in remote sensing image processing. First, two kinds of general parallel interpolation algorithms (for CPU and GPU, respectively) are designed. Then, in two typical application scenarios (data-intensive and computing-intensive), four computing methods (one serial method and three parallel methods) of these interpolation algorithms are tested. Finally, the acceleration effects of all parallel algorithms are compared and analyzed. On the whole, the acceleration effect of the parallel interpolation algorithm is better in computer-intensive scenario. In CPU-oriented methods, the speedup of all parallel interpolation algorithms mainly depends on the number of physical cores of CPU, whereas in GPU-oriented methods, a speedup is greatly affected by the computation complexity of an algorithm and the application scenario. GPU has a better acceleration effect on the interpolation algorithms with bigger computation complexity and has more advantages in the computing-intensive scenarios. In most cases, GPU-based interpolation is ideal for efficient interpolation.**

***Index Terms*—Compute-intensive, data-intensive, interpolation, parallel computing, remote sensing image.**

I. Introduction

IN REMOTE sensing image processing, interpolation is an important step in geometric processing and the basis of image reconstruction. In most cases, the speed of interpolation methods is also often an issue [1]. Early research works focus on the simplification of the interpolation model in order to make a tradeoff between the processing time and the quality of interpolation [2]. With the rapid development of computing power, subsequent research works focus on the application of parallel computing methods [3], [4], [5].

Early interpolation algorithms were relatively simple and the interpolation quality was average. After Shannon introduced information theory, the sinc function was accepted as an ideal interpolation function. However, interpolators based on this function have an infinite impulse response and are not suitable for local interpolation with only a finite impulse response. The solution to this problem is to approximate the sinc function. The functions suggested successively are Taylor polynomials, Lagrange polynomials, and different kinds of spline functions [2]. Up to now, there have been several mature interpolation methods. Among them, the most commonly used methods are nearest neighbor, linear and cubic interpolation, which are implemented by almost all image processing software. As for other interpolation methods, different software has its own tradeoffs and there are certain differences in implementation, such as spline interpolation.

Some application fields of interpolation may have higher requirements on processing speed, such as image processing and computer vision. Early solutions were to accelerate interpolation calculations by optimizing the interpolation model and using special hardware (FPGA). The former effectively reduces the computation of interpolation, and the latter provides a much faster computing speed than that of the algorithms realized by software. Lee et al. [6] proposed a multilevel B-splines interpolation algorithm, which achieved substantial performance improvement by refining a B-spline function into the sum of multiple functions. Berthaud et al. [7] applied the B-spline interpolation method to a real-time image rotation based on FPGA's board. Nuno-Maganda and Arias-Estrada [8] proposed a real-time FPGA architecture for bicubic interpolation, which is applied to digital image scaling. Serhat and Anıl [9] presented an FPGA implementation of a cubic spline interpolation method for empirical mode decomposition, which is 900 times faster than the software implementation. With the development of parallel computing technology, various parallel paradigms have been introduced into interpolation applications. Mei et al. [10] proposed an improved GPU-based adaptive inverse distance weighting interpolation algorithm for k-nearest neighbors search, with a speedup of 1000+. Mohanty et al. [11] proposed a parallel interpolation component known as skeleton, which can be implemented in several parallel paradigms, such as OpenMP, MPI, and CUDA.

Due to the large computation, parallel computing technology has been widely studied in remote sensing image processing [12]. According to the hardware architecture adopted, parallel computing methods can be roughly divided into three categories: CPU-oriented (such as multicore and multiprocessor) methods, special hardware-oriented (FPGA) methods, and GPU-oriented methods. The first two kinds of methods have been introduced into remote sensing image processing very early [13], whereas

Manuscript received 20 May 2023; revised 21 August 2023 and 14 September 2023; accepted 24 October 2023. Date of publication 31 October 2023; date of current version 23 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62176087 and in part by the Major Project of Science and Technology of Henan Province under Grant 201400210300. *(Corresponding author: Bing Zhou.)*

The authors are with the Henan Key Laboratory of Big Data Analysis and Processing and the School of Computer and Information Engineering, Henan University, Kaifeng 475004, China (e-mail: fmh139@163.com; xianyu_zuo@henu.edu.cn; zhoub1631@163.com).

Digital Object Identifier 10.1109/JSTARS.2023.3329018

the third kind has been introduced in recent 10 years [14], [15]. Interpolation is one of the foundations of geometric processing of remote sensing images, and there are many research works on parallel computing related to it, such as resampling [16], [17], geometric correction [18], [19], image registration [20], [21], image mosaic [22], [23], image fusion [22], [24], and so on. From the perspective of application scenarios, these studies can be divided into two categories: data-intensive and computation-intensive [25].

Generally speaking, the existing research works are all aimed at the application and/or optimization of an interpolation algorithm in a certain running environment. However, in practice, we may need to have more options, considering efficiency and quality at the same time. For example, Akima's interpolation method provides a natural and more suitable method for the smooth fitting of data, so it is very popular and there are a lot of improvement studies, such as Bica [26] proposed an optimization method to reduce oscillations on the end-points intervals. MATLAB also provides an improved implementation of this method [27]. Therefore, we choose several commonly used interpolation algorithms and make an in-depth study of their parallel computing methods in remote sensing image processing. Our work refers to several popular remote sensing image processing software (including GDAL, OpenCV, SNAP, ENVI, and MATLAB), and selects six interpolation algorithms, including nearest neighbor, linear, cubic, B-spline, sinc, and Akima algorithms. This article mainly introduces the design and implementation of the parallel algorithms of these algorithms, and compares and analyzes their acceleration effects in the two application scenarios, and also makes a simple comparison with existing research.

II. Interpolation Functions

The interpolation function introduced here is the basis of the interpolation algorithms implemented in this article. In order to simplify the description of the interpolation functions, we first give a general 2-D interpolation formula, as follows:

$$f\left(x,y\right) = \sum_{l=j-n}^{j+n+a} \sum_{k=i-n}^{i+n+a} f\left(k,l\right) * h\left(x-k, y-l\right) . \quad (1)$$

In the above-mentioned formula, the coordinate to be interpolated is (x, y), (i, j) is the coordinate of the sample point in the upper left corner of (x, y), $f(k, l)$ represents the sample point, $h(x\text{-}k,y\text{-}l)$ represents the interpolation kernel function, the value of n determines the size of the neighborhood, and $a = 0$ ($n = 1, 2, \ldots$) when the neighborhood is odd or $a = 1$ ($n = 0, 1, 2, \ldots$) when the neighborhood is even. In general, a 2-D interpolation kernel function can be expressed as a convolution of two 1-D interpolation kernel functions. Let $u = x–k$ and $v = y–l$, and then $h(x-k, y-l) = h(u) * h(v)$. In general, $h(u)$ and $h(v)$ are completely similar. They can be simply regarded as the weights of sample points.

Obviously, the difference between the interpolation formulas is just the kernel function. Therefore, only the formula $h(u)$ is given below (except for Akima interpolation).

A. Nearest Neighbor Interpolation

The 2-D neighborhood of the nearest interpolation is 2∗2, but only the sample point closest to the interpolation point is taken. The interpolation kernel function is as follows:

$$h\ (u) = \begin{cases} 1,\ 0 \le |u| < 0.5 \\ 0,\ \text{elsewhere} \end{cases} . \quad (2)$$

B. Linear Interpolation

The 2-D neighborhood of linear interpolation is 2∗2, in which the influence of all points is calculated. The interpolation kernel function is as follows:

$$h\ (u) = \begin{cases} 1 - |u|\,,\ 0 \le |u| < 1 \\ 0,\ \text{elsewhere} \end{cases} . \quad (3)$$

C. Cubic Interpolation

The 2-D neighborhood of cubic interpolation is 4∗4. The weight coefficient of each point in it is calculated by a piecewise cubic polynomial [28]. The interpolation kernel function is as follows:

$$h\ (u) = \begin{cases} (a+2)\,|u|^3 - (a+3)\,|u|^2 + 1,\ 0 \le |u| < 1 \\ a|u|^3 - 5a|u|^2 + 8a\,|u| - 4a,\ 1 \le |u| < 2 \\ 0,\ \text{elsewhere} \end{cases} \quad (4)$$

where the commonly used value of a is –0.5.

D. B-Spline Interpolation

B-splines are one of the most commonly used families of spline functions [29]. Its common 2-D neighborhood is 4∗4 or larger, such as 6∗6. It also uses a piecewise cubic polynomial to calculate the weight coefficient of each point in the neighborhood [30]. The interpolation kernel function is as follows:

$$h\ (u) = \begin{cases} \frac{1}{2}|u|^3 - |u|^2 + \frac{2}{3},\ 0 \le |u| < 1 \\ -\frac{1}{6}|u|^3 + |u|^2 - 2\,|u| + \frac{3}{4},\ 1 \le |u| \le 2 \\ 0,\ \text{elsewhere} \end{cases} . \quad (5)$$

Different from the above interpolation, $h(u)$ here is the weight coefficient of the control points of the B-spline curve [31]. We also need to calculate the corresponding control points based on the sample points [32], and the corresponding solution equations are as follows:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{k-2} \\ s_{k-1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 4 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \vdots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 4 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{k-2} \\ p_{k-1} \end{bmatrix} \quad (6)$$

where k is the 1-D dimension of the neighborhood. For example, if the 2-D neighborhood is 4∗4, then $k = 4$. $s_i (i = 0, 1, \ldots, k–1)$ represents a sample point, and $p_i (i = 0, 1, \ldots, k–1)$ represents a control point to be sought.

E. Sinc Interpolation

Sinc function, also known as ideal interpolation function, is infinite in space. In practice, its space is often truncated by some kind of window, such as the Lanczos window [33], Kaiser window, Hanning window, and Hamming window [34]. Its 2-D neighborhood is usually odd, such as 5∗5. The interpolation kernel function and a truncated window function are as follows:

$$h\ (u) = \frac{\sin \pi u}{\pi u} = \text{sinc}\,(u) \tag{7}$$

$$w_h\,(t, T) = \alpha - (1-\alpha)\cos\left(\frac{2\pi t}{T}\right),\ -\frac{2\pi}{T} \le t \le \frac{2\pi}{T} \tag{8}$$

where $w_h(t, T)$ is the Hanning window ($\alpha = 0.5$) or Hamming window ($\alpha = 0.54$) in the time domain.

F. Akima Interpolation

Akima interpolation is a continuous differentiable subspline interpolation [35]. Its 2-D neighborhood is 6∗6. Its 1-D interpolation only needs five adjacent points. They are assumed to be $x_{i\text{-}2}$, $x_{i\text{-}1}$, x_i, x_{i+1}, and x_{i+2}, and then the 1-D interpolation function is as follows:

$$f\ (x) = a_0\ + a_1\,(x-x_i) + a_2(x-x_i)^2 + a_3(x-x_i)^3$$
$$x_i \le x \le x_{i+1}. \tag{9}$$

In the above-mentioned formula, the calculation formula of a_i is as follows:

$$\begin{cases} a_0 = f\,(x_i) \\ a_1 = t_i \\ a_2 = \frac{3m_i - 2t_i - t_{i+1}}{x_{i+1}-x_i} \\ a_3 = \frac{t_{i+1}+t_i-2m_i}{(x_{i+1}-x_i)^2} \end{cases} \tag{10}$$

where t_i is the first derivative of the spline curve at x_i, and m_i is the slope between the point (x_i, $f(x_i)$) and the point (x_{i+1}, $f(x_{i+1})$). Their calculation formula is as follows:

$$\begin{cases} t_i = \frac{|m_{i+1}-m_i|m_{i-1}+|m_{i-1}-m_{i-2}|m_i}{|m_{i+1}-m_i|+|m_{i-1}-m_{i-2}|} \\ m_i = \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i} \end{cases}. \tag{11}$$

Different from the previous interpolation methods, its 2-D interpolation also needs to introduce partial differential conditions into the a_i [36], [37], otherwise the 2-D interpolation results are not always smooth. The 2-D interpolation function is as follows:

$$f\ (x,y) = \sum_{l=0}^{3}\sum_{k=0}^{3} a_{lk}(x-x_i)^k(y-y_j)^l$$
$$x \in [x_i, x_{i+1}]\,,\ y \in [y_j, y_{j+1}]\,. \tag{12}$$

The above formula needs to determine 16 coefficients, which can be found in the literature [37].

III. Methods

First of all, based on a serial interpolation algorithm, we designed two general parallel interpolation algorithms for CPU and GPU, respectively. Then, based on them and the above six interpolation functions, nine interpolation algorithms are implemented, each of which implements one serial method and three parallel computing methods, totaling 36 algorithms. Finally, based on an RPC-based orthorectification algorithm for remote sensing images, parallel computing experiments in two application scenarios are designed to test the computing time of these algorithms.

Algorithm 1: Pseudocode for a Serial Orthorectification.

```
  Input:
    I: a remote sensing image.
    E: a DEM image (geographic extent covers that of I).
  Output:
    X: the corrected image of I (m rows and n columns).
1 Get the metadata of I to create image X to be corrected.
2 for i = 1 → m do
3     for j = 1 → n do
4         Interpolate E to get the elevation of X(i, j).
5         Compute the position I(x, y) of X(i, j).
6         Interpolate I to get the value v of I(x,y).
7         X(i, j) ← v
8     end
9 end
```

A. Design of Parallel Algorithms

1) Serial Interpolation Algorithm for CPU: The design of the parallel algorithm is based on a serial orthorectification algorithm described in Algorithm 1.

In Algorithm 1, the locating of a pixel is done in each cycle, and the elevation is required for the fifth step. $I(x,\ y)$ is the pixel coordinate of I, corresponding to $X(i,\ j)$. The mapping relationship between them is based on a geo-spatial locating model.

Steps 4 and 6 in Algorithm 1 were chosen as our experimental scenarios. Parallel experimental algorithms are designed based on them, as are described in Algorithms 2 and 3.

2) Parallel Interpolation Algorithm for CPU: The basic idea of this algorithm is to divide the image into blocks, and then dispatch these blocks to different threads to parallel process. A general parallel interpolation algorithm for CPU is shown in Algorithm 2.

In Algorithm 2, the tasks in steps 5 and 6 are similar to those in steps 2–9 in Algorithm 1, both of which are processed pixel by pixel. The difference between them is that the former is for the whole image, whereas the latter is only for data blocks.

3) Parallel Interpolation Algorithm for GPU: The basic idea of this algorithm is to transfer the image into GPU to process, in order to make use of a large number of parallel threads of GPU to accelerate the processing. A general parallel interpolation algorithm for GPU is shown in Algorithm 3.

Considering the capacity limitation of GPU memory, Algorithm 3 divides the images to be processed into blocks (divided by row and their sizes are determined according to the GPU

Algorithm 2: Pseudocode for a CPU-Oriented Parallel Interpolation.

```
Input:
  X: the image X to be corrected.
  I: a remote sensing image or a DEM image.
Output:
  T: the interpolated images (intermediate data).
1  Divide X and I into $n$ blocks.
2  for $k = 1 \rightarrow n$ do
3      Dispatch block pair [$X(k)$, $I(k)$] to a CPU thread.
4      for each CPU thread do
5          Compute the position $P(k)$ of $X(k)$.
6          Interpolate I with $P(k)$ to get $T(k)$.
7      end
8  end
9  Gather all interpolated blocks to generate T.
```

Algorithm 3: Pseudocode for a GPU-Oriented Parallel Interpolation.

```
Input:
  X: the image to be corrected.
  I: a remote sensing image or a DEM image.
Output:
  T: the interpolated images (intermediate data).
1   Divide X and I into $n$ blocks.
2   for $k = 1 \rightarrow n$ do
3       Transfer block pair [$X(k)$, $I(k)$] to a GPU.
4       Allocate memory of the GPU for $T(k)$.
5       for each GPU kernel function do
6           Locate a pixel of $X(k)$ in $I(k)$, resulting in $p$.
7           Interpolate I to get the value $v$ of $p$.
8           Save $v$ to $T(k)$.
9       end
10      Get the interpolated block $T(k)$ from the GPU.
11  end
12  Merge all interpolated blocks to generate T.
```

memory capacity). In Algorithm 3, each GPU thread processes only one pixel at a time.

B. Implementation of Parallel Algorithms

We have implemented nine interpolation algorithms, including six basic algorithms and three improved algorithms (using batch mode). The so-called batch mode refers to the centralized processing of some preprocessing computations that can be performed independently before interpolation, in order to reduce the computation during interpolation. These computations include the boundary processing of bicubic algorithm, the solution of control points of B-spline algorithm, and the computation of differential conditions of Akima algorithm. The other three interpolation algorithms do not have such preprocessing computations.

Each algorithm implements the following four computing methods.

1) Serial computing method (implemented Algorithm 1).
2) Parallel computing method based on OpenMP (implemented Algorithm 2).
3) Parallel computing method based on ISO C++ 11 (implemented Algorithm 2).
4) Parallel computing method based on CUDA (implemented Algorithm 3).

The implementation of the serial method refers to the relevant implementations of the open-source software GDAL, OpenCV, and SNAP, and unifies the implementation modes of all algorithms. OpenMP is a set of guiding compilation and processing schemes for multiprocessor programming in shared memory parallel systems. It is easy to use and has become one of the popular parallel computing technologies. Our implementation mainly uses its “parallel for” directive to segment image processing automatically. Starting from the 11th edition, the ISO C++ standard has been added as a support framework for parallel computing. Our implementation leverages its mapping mechanism between data blocks, algorithms, and threads to customize parallel computing schemes. CUDA is a parallel computing platform and programming model invented by NVIDIA Corporation. It is one of the mainstream parallel computing technologies based on GPU.

Referring to some literature and combined with our engineering practice, some key parameters of the interpolation algorithms are set as follows: nearest neighbor, bilinear and bicubic interpolation algorithms use default common parameters; the neighborhood of B-spline interpolation algorithm takes $6*6$; Akima interpolation algorithm adopts the improved scheme of MATLAB; sinc interpolation algorithm uses Hanning window function, and its neighborhood takes $7*7$.

C. Experimental Environment

The development environment of the experiment is Windows 10, and various software and hardware configurations are described in Table I.

D. Experimental Data and Application Scenarios

The experiment is based on a RPC-based orthorectification algorithm for remote sensing images. The input and output images of the algorithm are shown in Fig. 1.

The size of each band of the GF1-WFV image is about 306 MB (TIFF format) and the dimension is $12\,000*13\,400$. The size of the corrected image varies depending on the storage format and the dimension is $17\,266*15\,597$. In the correction, the image will be interpolated for about 269 million times. Because the computation complexity of the interpolation algorithm is not large, the feature of the interpolation of the image tends to be data-intensive.

The size of the DEM image is 18.90 MB and the dimension is $3634*2710$. In the correction, the image will also be interpolated 269 million times. Compared with the GF1-WFV image, the

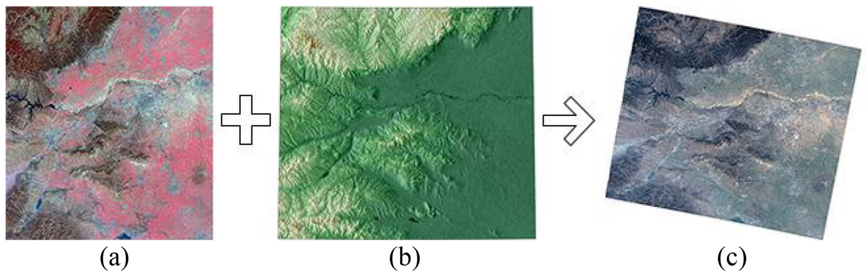

Fig. 1. Original image, DEM image, and corrected image. (a) Official preview of a GF1-WFV image (pseudocolor) with a 16 m resolution, which can be downloaded from the website https://data.cresda.cn. (b) DEM image (3-D false-color) with 90 m resolution, which is derived from the free SRTM data. (c) Corrected image (the true color image synthesized from the first, second, and third bands of the corrected image).

TABLE I
EXPERIMENTAL SYSTEM CONFIGURATIONS

Item	Configurations
CPU	Intel Xeon E5-1620 CPUs running at 3.6 GHz 4 physical cores, 8 hyperthreads 32 GB memory Memory bandwidth of 25.6 GB/s
GPU	nVidia Quadro P620 512 cores 2GB memory Memory bandwidth of 80.13 GB/s
OpenMP	Build-in version of Visual Studio 2019
ISO C++ 11	Build-in version of Visual Studio 2019
CUDA	CUDA Toolkit 11.4

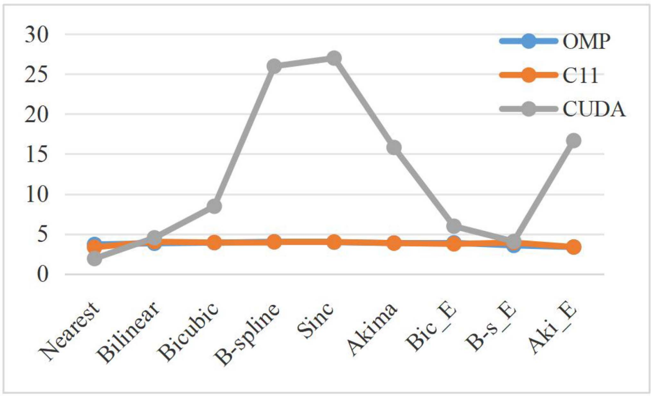

Fig. 2. Speedup of all parallel algorithms.

feature of the interpolation of this image is significantly more computing-intensive.

IV. Result and Discussion

Based on the above algorithms and data, the computing time of 36 algorithms is tested in two scenarios. In order to facilitate the comparison of acceleration effects, the interpolation in both scenarios takes only one band (the GF1-WFV image has four bands). The test results and their analysis are as follows.

A. *Data-Intensive Interpolation*

The interpolation of GF1-WFV images is data-intensive. The computing time of all algorithms and the speedup of parallel algorithms are shown in Table II.

The comparison of the acceleration effects of all parallel algorithms is shown in Fig. 2.

From Table II and Fig. 2, the acceleration effects of the three parallel computing methods have the following characteristics.

1) All three methods have a good acceleration effect, the lowest speedup is close to two times (nearest interpolation algorithm based on CUDA) and the highest speedup is 27 times (sinc interpolation algorithm based on CUDA).
2) The speedup effects of the OMP method and the C11 method are almost identical. The speedup lines of the two methods in Fig. 2 almost completely overlap. Table II also shows that the difference between them is small. This should be related to their similar implementation mechanism, both of which are to process data in blocks.
3) The speedup of the CPU-oriented computing method mainly depends on the number of physical cores of CPU, followed by the ratio of data processing time to computing time, and other factors have little influence. Table II shows that most of the speedups of these methods are around 4, whereas the speedup of algorithms with a large proportion of data processing time is slightly smaller, such as Nearest and Aki_E algorithm. This is consistent with our previous research results on geometric correction [38]. It can be seen that increasing the number of physical cores is the most effective way to improve the performance of this kind of computing method.
4) The acceleration effect of the CUDA method on different algorithms is very different. Compared with the CPU-oriented method, the speedup of most algorithms based on CUDA is higher or much higher, whereas the speedup of Bilinear and B-s_E is not much different from that of the former, and the speedup of the nearest algorithm is significantly lower than that of the former. The main reason for this difference is the unique architecture and data processing mode of GPU. On the one hand, the parallel computing of a large number of physical cores in GPU can greatly reduce the computing time. The longer the computing time of the algorithm, the higher the speedup, such as the sinc interpolation algorithm. On the other hand, data needs to be transferred between CPU memory

TABLE II
COMPUTING TIME (SECONDS) AND SPEEDUP OF ALGORITHMS IN DATA-INTENSIVE INTERPOLATION

		Nearest	Bilinear	Bicubic	B-spline	Sinc	Akima	Bic_E	B-s_E	Aki_E	AVG
Ser	Time	2.87	6.13	16.53	52.85	75.42	50.85	12.88	22.65	75.17	35.04
OMP	Time	0.78	1.60	4.22	13.13	19.00	13.16	3.31	6.37	22.37	9.33
	Speedup	3.69	3.83	3.91	4.03	3.97	3.86	3.89	3.56	3.36	3.79
C11	Time	0.86	1.52	4.21	13.20	18.88	13.20	3.43	5.79	22.35	9.27
	Speedup	3.33	4.03	3.92	4.00	4.00	3.85	3.75	3.91	3.36	3.80
CUDA	Time	1.50	1.36	1.95	2.03	2.79	3.21	2.16	5.58	4.51	2.79
	Speedup	1.92	4.52	8.46	25.98	26.99	15.82	5.97	4.06	16.68	12.27

Bic_E, B-s_E, and Aki_E correspond to the Bicubic, B-spline, and Akima algorithms that use batch mode respectively. AVG represent average value. Ser, OMP, C11 and CUDA represent four computing methods, which correspond to serial method, parallel method based on OpenMP, parallel method based on ISO C++ 11 and parallel method based on CUDA. The following representation is the same as here.

TABLE III
COMPUTING TIME (SECONDS) AND SPEEDUP OF ALGORITHMS IN COMPUTING-INTENSIVE INTERPOLATION

		Nearest	Bilinear	Bicubic	B-spline	Sinc	Akima	Bic_E	B-s_E	Aki_E	AVG
Serial	Time	3.88	8.50	21.37	70.80	99.96	66.98	15.98	26.17	21.68	37.26
OMP	Time	0.93	1.96	5.14	16.64	23.74	16.70	3.95	6.54	5.35	8.99
	Speedup	4.15	4.33	4.15	4.26	4.21	4.01	4.05	4.00	4.05	4.13
C11	Time	0.90	2.00	5.38	16.89	23.53	17.09	4.00	6.51	5.38	9.07
	Speedup	4.29	4.26	3.97	4.19	4.25	3.92	4.00	4.02	4.03	4.10
CUDA	Time	0.68	0.54	1.23	1.34	2.33	2.75	0.72	1.18	0.70	1.27
	Speedup	5.70	15.73	17.44	52.95	42.83	24.35	22.25	22.11	30.96	26.04

In the table, the computing time of the serial algorithms is higher than that in Table II except for that of the Aki_E algorithm. The reason is there is an extra pixel location before DEM interpolation. The great change of the Aki_E algorithm is analyzed below.

and GPU memory during processing, which increases the proportion of data processing time, especially unfavorable for algorithms with short computing time, such as the nearest interpolation algorithm.

5) The improvement of algorithm performance in batch mode is closely related to the data volume to be preprocessed. Algorithms with smaller data can significantly improve computing performance in CPU-oriented methods, such as Bic_E and B-s_E algorithms, whereas algorithms with larger data will reduce computing performance, such as Aki_E algorithm. The reason for this phenomenon is that the storage and transmission of intermediate data generated by preprocessing takes extra time. The more intermediate data, the longer the preprocessing takes.

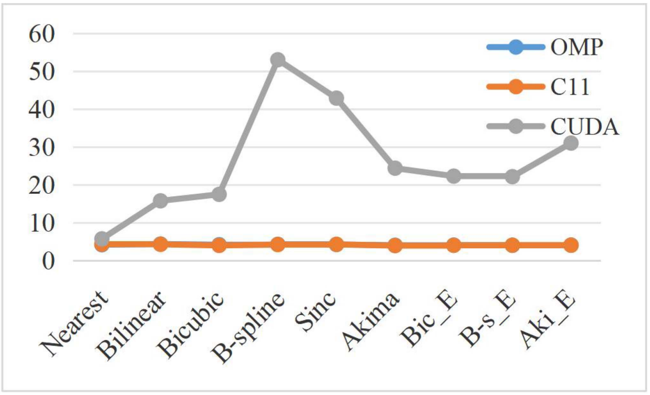

Fig. 3. Speedup of each parallel algorithm.

B. Computing-Intensive Interpolation

The interpolation of the DEM image is data-intensive. The computing time of all algorithms and the speedup of parallel algorithms are shown in Table III.

The comparison of the acceleration effects of each parallel algorithm is shown in Fig. 3.

Compared with Table II and Fig. 2, Table III and Fig. 3 show significant differences, which shows that the performance of parallel computing methods has changed greatly in computing-intensive scenarios, as described in the following.

1) All the speedup increases. This fully shows that the parallel computing method has more advantages in computing-intensive interpolation. For CPU, smaller data are more conducive to using caching to optimize data access speed. For GPU, smaller data gives the algorithm the opportunity to use the data broadcast mechanism to greatly speed up the data reading speed.

2) Table III shows that in the CPU-oriented methods, most of the speedups slightly exceed the number of physical cores of the CPU, and the average reaches 4.1. This is the benefit of the hyper-threading optimization of the CPU, but the improvement is limited. In practice, if hyper-threading is turned OFF or the number of threads is limited to the number of physical cores of the CPU, there will be a small loss in speedup. This also shows that in the CPU-oriented method, only by adding physical cores can the computing performance be greatly increased.

3) Compared with the CPU-oriented method, the speedup increase of the CUDA method is very significant, and the average value is more than double that of Table II. Even for the nearest interpolation algorithm, its speedup exceeds its CPU-oriented method. This mainly benefits from the data broadcasting mechanism of GPU. This mechanism is useless in data-intensive interpolation.
4) The batch mode algorithms have achieved a complete victory among the four computing methods, and the computing time is much lower than that of the original algorithm. Their performance in the CUDA method is even more amazing. The speedup of the Bic_E and B-s_E algorithms increases several times, and the Aki_E algorithm achieves the inversion, from slower than that of the original algorithm in data-intensive interpolation to much faster than that of the original algorithm in computing-intensive. The reason for this great change is that the preprocessed data volume of these algorithms is greatly reduced, which saves a lot of processing time. In addition, these algorithms can also benefit greatly from the data broadcasting mechanism of GPU when reading data.

C. Comparison With Existing Research Works

The performance of the interpolation algorithm is closely related to the implementation of the algorithm and the running environment. This article studies the software implementation of several equidistant interpolation algorithms, only for CPU and GPU. Therefore, we briefly compare them from the following three aspects.

1) Implementation of the algorithm. Existing studies all focus on an interpolation algorithm and often optimize the calculation process for the running environment. For example, Mohanty et al. [11] separated the cubic spline interpolation process into multiple phases to use the pipelining model for the reduction process, in order to adapt to several parallel paradigms. FPGA-based implementation needs to redesign the calculation process according to the hardware architecture. In this article, several interpolation algorithms are studied. In order to conduct a simple horizontal comparison, similar optimization is not used, because some interpolation algorithms cannot be further optimized.
2) Acceleration effect. As far as the software implementation of similar algorithms (such as spline interpolation) is concerned, our speedup is much higher, which may be affected by implementation and hardware performance. But it is also consistent with the conclusion that the larger the size of data, the higher the speedup [11] (the size of the image we use is much larger). However, our speedup is also much lower than irregular interpolation [10] and FPGA-based interpolation. Of course, it is not fair to compare between different types of interpolation methods or applications.
3) Real-time. Previous studies are based on FPGA to achieve real-time interpolation [7], [8]. In our research, the CUDA-based method has been able to interpolate a large image in a few seconds, which can meet the requirements of some near real-time processing. Obviously, if there is more powerful hardware, the software implementation algorithms can also perform real-time computations.

V. Conclusion

Parallel computing is a common acceleration method for remote sensing image processing. All kinds of interpolation algorithms can greatly improve the computing speed through it. However, different parallel computing methods have different acceleration effects on different interpolation algorithms, and different application scenarios have different effects on the methods. On the whole, the speedup of CPU-oriented parallel computing methods is mainly determined by the number of physical cores of CPU, and other factors have little influence. Compared with data-intensive interpolation, computing-intensive interpolation can use the hyper-threading of GPU to obtain a higher speedup. However, the speedup of the GPU method varies with different interpolation algorithms or different application scenarios. On the one hand, an interpolation algorithm with a high computation complexity can obtain a high speedup and vice versa. This is beneficial to the efficient application of interpolation algorithms with large computation complexity. On the other hand, in most cases, the speedup of computing-intensive interpolation is much higher than that of data-intensive interpolation. It shows that the GPU has a greater advantage in such scenarios.

In practical applications, the choice of parallel computing methods needs to consider the computing requirements. In most cases, the GPU-oriented methods are the best choice. If you only use CPU-oriented methods, OpenMP is usually a good choice. If the computing process needs to be handled flexibly, then the ISO C++ 11 (or higher version) parallel framework is a better choice. In addition, for data-intensive interpolation with a smaller computing complexity, such as nearest neighbor, the CPU-based parallel computing method is the best choice.

To sum up, in two typical application scenarios, this article studies three parallel acceleration methods of six commonly used interpolation algorithms in remote sensing image geometric processing, and analyzes and compares the acceleration effects. Further research lines in the future may include the following.

1) Internal depth optimization of complex interpolation algorithms in different scenarios.
2) Real-time computing of complex interpolation algorithms supported by high-performance hardware.
3) FPGA-based implementation of complex interpolation algorithms.
4) Application of various parallel paradigms (such as OpenCL) in interpolation, and so on.

References

[1] N. A. Dodgson, "Quadratic interpolation for image resampling," *IEEE Trans. Image Process.*, vol. 6, no. 9, pp. 1322–1326, Sep. 1997.

[2] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Trans. Med. Imag.*, vol. 18, no. 11, pp. 1049–1075, Nov. 1999.

[3] D. Crookes, "Architectures for high performance image processing: The future," *J. Syst. Archit.*, vol. 45, no. 10, pp. 739–748, 1999, doi: 10.1016/S1383-7621(98)00035-6.

[4] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, “Foreword to the special issue on high performance computing in Earth observation and remote sensing,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 503–507, Sep. 2011.
[5] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya, “pipsCloud: High performance cloud computing for remote sensing big data management and processing,” *Future Gener. Comput. Syst.—Int. J. ESci.*, vol. 78, no. 1, pp. 353–368, 2018, doi: 10.1016/j.future.2016.06.009.
[6] S. Lee, G. Wolberg, and S. Y. Shin, “Scattered data interpolation with multilevel B-splines,” *IEEE Trans. Vis. Comput. Graph.*, vol. 3, no. 3, pp. 228–244, Jul.–Sep. 1997.
[7] C. Berthaud, E. Bourennane, M. Paindavoine, and C. Milan, “Implementation of a real time image rotation using B-spline interpolation on FPGA’s board,” in *Proc. Int. Conf. Image Process.*, Rhodes, Greece, 1998, pp. 995–999.
[8] M. A. Nuno-Maganda and M. O. Arias-Estrada, “Real-time FPGA-based architecture for bicubic interpolation: An application for digital image scaling,” in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Puebla, Mexico, 2005, pp. 1–8.
[9] Ç. Serhat and Ç. Anıl, “FPGA implementation of cubic spline interpolation method for empirical mode decomposition,” in *Proc. 20th Signal Process. Commun. Appl. Conf.*, Mugla, Turkey, 2012, pp. 1–4.
[10] G. Mei, N. Xu, and L. Xu, “Improving GPU-accelerated adaptive IDW interpolation algorithm using fast kNN search,” *SpringerPlus*, vol. 5, no. 1, 2016, Art. no. 1389, doi: 10.1186/s40064-016-3035-2.
[11] P. K. Mohanty, M. Reza, P. Kumar, and P. Kumar, “Implementation of cubic spline interpolation on parallel skeleton using pipeline model on CPU-GPU cluster,” in *Proc . IEEE 6th Int. Conf. Adv. Comput.*, 2016, pp. 747–751.
[12] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, “Recent developments in high performance computing for remote sensing: A review,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.
[13] A. J. Plaza and C. Chang, *High Performance Computing in Remote Sensing* (Chapman & Hall/CRC Computer and Information Science Series). Boca Raton, FL, USA: Chapman & Hall/CRC, 2008, p. 466.
[14] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A unified graphics and computing architecture,” *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar./Apr. 2008.
[15] J. Nickolls and W. J. Dally, “The GPU computing era,” *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010.
[16] P. Liu and J. Gong, “Parallel construction of global pyramid for large remote sensing images,” *Geomatics Inf. Sci. Wuhan Univ.*, vol. 41, no. 1, pp. 117–122, 2016, doi: 10.13203/j.whugis20130718.
[17] Z. Chi, F. Zhang, Z. Du, and R. Liu, “Parallel resampling method of remote sensing data based on pre-partitioning for cloud computing,” *J. Shanghai Jiao Tong Univ.*, vol. 48, no. 11, pp. 1627–1632+1638, 2014.
[18] L. Fang, M. Wang, D. Li, and J. Pan, “CPU/GPU near real-time preprocessing for ZY-3 satellite images: Relative radiometric correction, MTF compensation, and geocorrection,” *ISPRS J. Photogramm. Remote Sens.*, vol. 87, pp. 229–240, 2014, doi: 10.1016/j.isprsjprs.2013.11.010.
[19] Y. Jiang, X. Yang, and H. Yi, “Parallel algorithm of geometrical correction for satellite images,” *Chin. J. Comput.*, vol. 27, no. 7, pp. 944–951, 2004, doi: 10.1016/S0305-0548(02)00154-5.
[20] X. Zhang and X. Zhao, “High-precision registration algorithm and parallel design method for high-resolution optical remote sensing images,” *Int. J. Pattern Recognit. Artif. Intell.*, vol. 35, no. 7, 2021, Art. no. 2154020, doi: 10.1142/S0218001421540203.
[21] Z. Zhu, Z. Shen, and J. Luo, “Parallel remote sensing image registration based on improved SIFT point feature,” *J. Remote Sens.*, vol. 15, no. 5, pp. 1024–1039, 2011, doi: 10.3724/SP.J.1146.2010.00112.
[22] J. Yang and J. Zhang, “Parallel performance of typical algorithms in remote sensing-based mapping on a multi-core computer,” *Photogramm. Eng. Remote Sens.*, vol. 81, no. 5, pp. 373–385, 2015, doi: 10.14358/PERS.81.5.373.
[23] L. Chen, Y. Ma, P. Liu, J. Wei, W. Jie, and J. He, “A review of parallel computing for large-scale remote sensing image mosaicking,” *Cluster Comput.*, vol. 18, no. 2, pp. 517–529, 2015, doi: 10.1007/s10586-015-0422-3.
[24] J. Yang, “The generalized model and parallel computing methods for pixel-level remote sensing image fusion,” Ph.D. disseration, Wuhan Univ., 2014.
[25] M. K. Pektürk and M. Ünal, “High-performance computing for remote sensing big data analytics,” in *Data Mining*, C. Thomas, Ed. London U.K.: IntechOpen Ltd., 2018.
[26] A. M. Bica, “Optimizing at the end-points the Akima’s interpolation method of smooth curve fitting,” *Comput. Aided Geometric Des.*, vol. 31, no. 5, pp. 245–257, 2014, doi: 10.1016/j.cagd.2014.03.001.
[27] C. Moler, “Makima piecewise cubic interpolation,” 2019. Accessed: Oct. 22, 2022. [Online]. Available: https://blogs.mathworks.com/cleve/2019/04/29/makima-piecewise-cubic-interpolation
[28] R. Keys, “Cubic convolution interpolation for digital image processing,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 6, pp. 1153–1160, Dec. 1981.
[29] M. Unser, “Splines: A perfect fit for signal and image processing,” *IEEE Signal Process. Mag.*, vol. 16, no. 6, pp. 22–38, Nov. 1999.
[30] P. Thévenaz, T. Blu, and M. Unser, “Interpolation revisited [medical images application],” *IEEE Trans. Med. Imag.*, vol. 19, no. 7, pp. 739–758, Jul. 2000.
[31] G. M. Phillips and P. J. Taylor, “Algorithms for spline and other approximations to functions and data,” *Comput. Phys. Commun.*, vol. 73, pp. 1–21, 1992, doi: 10.1016/0010-4655(92)90025-T.
[32] H. Caglar, N. Caglar, and K. Elfaituri, “B-spline interpolation compared with finite difference, finite element and finite volume methods which applied to two-point boundary value problems,” *Appl. Math. Comput.*, vol. 175, no. 1, pp. 72–79, 2006, doi: 10.1016/j.amc.2005.07.019.
[33] C. E. Duchon, “Lanczos filtering in one and two dimensions,” *J. Appl. Meteorol.*, vol. 18, pp. 1016–1022, 1979, doi: 10.1175/1520-0450(1979)018<1016:LFIOAT>2.0.CO;2.
[34] I. G. Cumming and F. H. Wong, *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*. Norwood, MA, USA: Artech House, 2005.
[35] H. Akima, “A new method of interpolation and smooth curve fitting based on local procedures,” *J. Assoc. Comput. Machinery*, vol. 17, no. 4, pp. 589–602, 1970, doi: 10.1145/321607.321609.
[36] H. Akima, “A method of bivariate interpolation and smooth surface fitting based on local procedures,” *Commun. ACM*, vol. 17, pp. 18–20, 1974, doi: 10.1145/360767.360779.
[37] H. Akima, “Algorithm 474: Bivariate interpolation and smooth surface fitting based on local procedures [E2],” *Commun. ACM*, vol. 17, no. 1, pp. 26–31, 1974.
[38] M. Fan, X. Zuo, and J. Tian, “Parallel geometric correction for single spaceborne SAR image,” in *Proc. 6th Asia–Pac. Conf. Synthetic Aperture Radar*, Xiamen, China, 2019, pp. 1–5.

Minghu Fan received the dual B.S. degrees in electronic engineering and computing applications from the South-Central Minzu College, Wuhan, China, in 1997, the M.S. degree in computer applied technology from the China University of Geosciences, Wuhan, in 2007, and the Ph.D. degree in cartography and geographic information engineering from the Wuhan University, Wuhan, in 2013.

He is currently a Lecturer with the school of computer and information engineering, Henan University, Kaifeng, China. His current research interests include high-performance computing and remote sensing image processing.

Xianyu Zuo received the B.S. and M.S. degrees in applied mathematics from the Henan Normal University, XinXiang, China, in 2003 and 2006, respectively, and the Ph.D. degree in applied mathematics from the China Academy of Engineering Physics, Beijing, China, in 2012.

He is currently an Associate Professor with the School of Computer and Information Engineering, Henan University, Kaifeng, China. His current research interests include high productivity computing and parallel computing of remote sensing.

Bing Zhou received the B.S. degree in computer science and its application from the Northwest Polytechnic University, Xi’an, China, in 1997, and the M.S. degree in computer software and theory and the Ph.D. degree in computer science and technology from the Xi’an Jiaotong University, Xi’an, in 2000 and 2004, respectively.

He is currently an Associate Professor with the School of Computer and Information Engineering, Henan University, Kaifeng, China. His research interests include parallel computing and data mining.