sensors

MDPI

Article

High Resolution and Fast Processing of Spectral Reconstruction in Fourier Transform Imaging Spectroscopy

Weikang Zhang [1,2,*], Desheng Wen [1], Zongxi Song [1], Xin Wei [1,2], Gang Liu [1,2] and Zhixin Li [1,2]

1 Xi'an Institute of Optics and Precision Mechanics, Chinese Academy of Sciences, Xi'an 710119, China; ven@opt.ac.cn (D.W.); songxi@opt.ac.cn (Z.S.); weixin@opt.cn (X.W.); liugang@opt.cn (G.L.); lizhixin2015@opt.cn (Z.L.)

2 University of Chinese Academy of Sciences, Beijing 100049, China

* Correspondence: zhangweikang@opt.cn

Received: 20 August 2018; Accepted: 22 November 2018; Published: 27 November 2018

Abstract: High-resolution spectrum estimation has continually attracted great attention in spectrum reconstruction based on Fourier transform imaging spectroscopy (FTIS). In this paper, a parallel solution for interference data processing using high-resolution spectrum estimation is proposed to reconstruct the spectrum in a fast high-resolution way. In batch processing, we use high-performance parallel-computing on the graphics processing unit (GPU) for higher efficiency and lower operation time. In addition, a parallel processing mechanism is designed for our parallel algorithm to obtain higher performance. At the same time, other solving algorithms for the modern spectrum estimation model are introduced for discussion and comparison. We compare traditional high-resolution solving algorithms running on the central processing unit (CPU) and the parallel algorithm on the GPU for processing the interferogram. The experimental results illustrate that runtime is reduced by about 70% using our parallel solution, and the GPU has a great advantage in processing large data and accelerating applications.

Keywords: spectrum reconstruction; Fourier transform imaging spectrometer; parallel computing; high performance; GPU

1. Introduction

In recent years, with the rapid development of imaging spectroscopy, the Fourier transform spectrometer has become one of the most important payloads in space exploration and component analysis [1–6]. This type of spectrometer, the Fourier transform imaging spectrometer or interference imaging spectrometer, has wide application in remote sensing, environmental monitoring, mineral exploration, and agriculture because of its high throughput, multiple channels, and high resolution.

Fourier transform imaging spectroscopy (FTIS) [7,8] acquires the data cube containing space and spectral information of the same target. Spectrum reconstruction [9–12] is the process of transformation from the interferogram to spectrogram. The remote sensing image, whose initial data are collected by the interference imaging spectrometer, is a typical application of spectrum reconstruction.

The equipment at the heart of spectroscopy is the interferometer. In theory and practice, usually, measures are taken to divide light from the target into two coherent light beams, which will interfere on the sensor and change the optical path difference (OPD) of the two beams in order to obtain a series of interference patterns. Figure 1 shows the principle of large aperture static imaging spectrometry (LASIS) [13].

The current Fourier transform spectrometers are able to capture large areas in increased spatial and spectral resolution, and the amount of interferograms from the sensors both in orbit and on the

ground is increasing rapidly. To process the large-scale interferograms as soon as possible, we usually omit some steps so that the recovery spectrum has many side lobes and false peaks, which is not reliable. These problems should be solved in a fast processing pipeline.

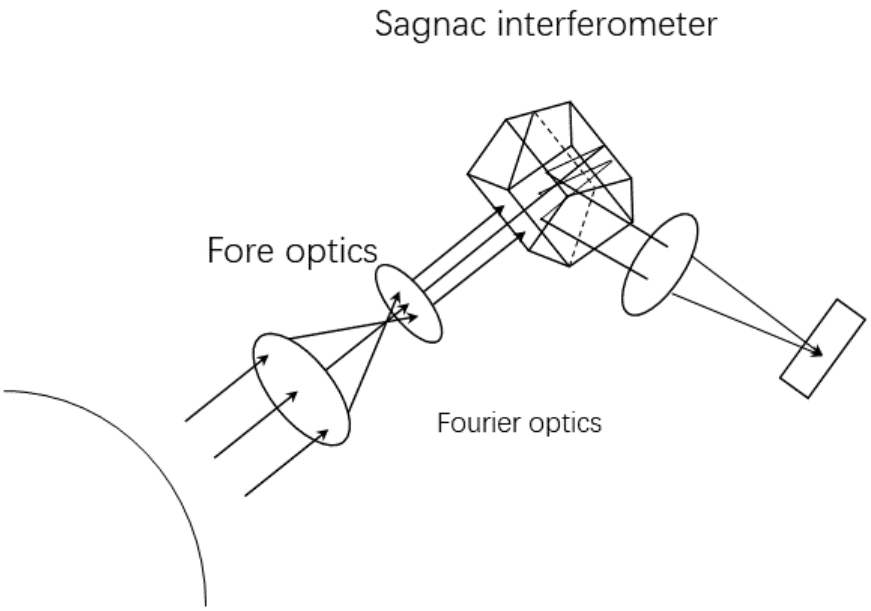

Figure 1. Schematic diagram of the large aperture static imaging spectrometry (LASIS) based on the Sagnac interferometer.

In the traditional spectrum reconstruction mechanism, fast Fourier transform (FFT) is the key part. However, the spectrum reconstructed by FFT from the actual obtained interferogram extends badly and has many side lobes, which would decrease the recovery accuracy and the resolution power. In addition, the interferogram cannot be scanned to infinity as the theoretical calculation. The interferogram value, out of the scanning range, is regarded as zero, which can result in spectrum leakage. If we use apodization technology, the spectrum leakage could be solved, but the main lobe width of the power spectrum would increase. With its full width at half maximum (FWHM) becoming large, there may be spectral peaks that are hard to identify in the spectrum. Meanwhile, the spectral resolution of FFT depends on the length of the signal. In practice, especially on a satellite, it is not permitted to transmit long interference data, and there are not enough sampling points to improve the spectral resolution. When the signal is short, the spectrum is difficult to identify carefully.

To solve these problems, Jian et al. [14,15] brought modern spectrum estimation into the field of spectral reconstruction. They introduced the multiple signal classification (MUSIC) [16] algorithm and the autoregressive (AR) [17,18] model for better performance in spectral recovery. These algorithms are good at spectrum reconstruction in resolution, but are very time consuming. The MUSIC algorithm is suitable for monochromatic light spectrum reconstruction, whereas AR spectral estimation is widely applicable. Although modern spectrum estimation has many advantages in resolution and suppressing the heavy-tailed spectrum, even for a short signal, the algorithm complexity is so high that these models are restricted in practical spectrum reconstruction. Quick solutions are needed.

In recent years, programmable graphics hardware has grown significantly in terms of performance and functionality. In comparison with the traditional data processing pipeline, performing general-purpose computations on GPUs is a new means of data handling, and it is possible to achieve a significant reduction in processing time when parallelized and executed on a GPU in accelerating algorithms and batch processing. It is particularly suitable for solving problems that can be expressed as data parallel computation, that is the same program is executed on many data elements in parallel [19]. Furthermore, compared with multi-thread computation on the CPU and multi-core CPUs, the parallel

computing on GPUs has thousands of threads, which are simple to create and apply. In addition, multi-thread computation on the CPU and multi-core CPUs would occupy many system resources and decrease the total performance of the system. However, on GPUs, it just needs to transfer data from the CPU to the GPU and the GPU to the CPU and would not occupy resources on the CPU. The complex computing tasks are assigned to the GPU, and this does not have an impact on the overall performance of the system. Another important advantage on the GPU is that we can process many batches of data at the same time. The more the amount of data, the higher the performance.

Compute Unified Device Architecture (CUDA) was introduced by NVIDIA, a general purpose parallel computing platform and programming model using the parallel-compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. Another advantage of CUDA is that it provides standard programming languages such as C/C++, which is familiar for programmers [20,21]. This is an effective way to deal with many interferograms to reach the goal of high resolution and fast processing in spectrum reconstruction.

This paper will focus on quick solutions of the autoregressive model for high spectral resolution with better performance in real time and the parallel processing scheme of spectrum reconstruction based on high performance parallel computing on the GPU. The main contributions of our research are as follows:

- We propose a parallel solution for high-resolution spectrum estimation. We use the parallel Burg method to solve the AR model and to obtain the model solution in a faster way. We validate the performance of our parallel algorithm by comparing it to measures on a CPU that is also used in the experiments.
- We use our parallel Burg solution for batch processing. The GPU is suitable for data parallel computation. We accelerate the application in data parallelism. Thousands of threads have been used for large-scale data processing.
- We design an asynchronous parallel processing mechanism for high-resolution spectrum reconstruction by making full use of the GPU with overlap. Within the limited threads and memory, the asynchronous parallel mechanism simultaneously executes kernels and transfers data using multiple streams. The number of streams can be set to two, three, or other values.
- Many solutions for the AR model are discussed for comparison. We use the Yule–Walker and least-squares methods for the AR model, and two recursive algorithms of the Yule–Walker equation have been employed for fast parameter estimation.

The rest of this paper is organized as follows: Section 2 discusses the related work, and Section 3 depicts a parallel algorithm and designs a parallel processing mechanism for fast parameter estimation. The experiments are arranged in Section 4. In Section 5, we give an analysis and draw conclusions.

2. Related Work

In this section, we will discuss the related work on the general data processing pipeline and a high-resolution model for the interferogram in the Fourier transform spectrometer. Two theoretical solutions for the model are also discussed.

2.1. General Data Processing

Based on the theory of Fourier transform spectroscopy [22], the spectrum could be obtained through Fourier transformation for the interferogram, listed as the following equations:

$$I(\Delta) = \int_{-\infty}^{+\infty} B(\sigma) e^{j2\pi\sigma\Delta} \, d\sigma, \tag{1}$$

$$B(\sigma) = \int_{-\infty}^{+\infty} I(\Delta) e^{-j2\pi\sigma\Delta} \, d\Delta, \tag{2}$$

where I is the interferogram, B is the spectrum, and Δ and σ mean the path difference and the wave number, respectively. The processing of the interferogram mainly includes preprocessing, apodization [23–25], phase correction [26–28], and FFT, where apodization is to eliminate the effect of the instrument linear function for power spectrum and phase correction can correct phase error. This is the most direct and simple way for spectrum reconstruction based on Equations (1) and (2) between the interferogram and spectrogram.

The traditional algorithm mainly including FFT and apodization is substantially equivalent to the BT (Blackman and Tukey) method in classical spectrum estimation. However, the classical method suffers from large estimated variance and bad resolution, which could be overcome by modern spectrum estimation, the autoregressive model for example.

2.2. Autoregressive Model

The autoregressive (AR) model is a linear prediction model that could be employed to predict unobserved values when given a set of known data. In theory, any random signal can be considered as a sequence produced by white noise through the model, whose parameters could be calculated by observed values so that the unobserved data are not thought to be zero. Generally, for generating random sequences, a linear difference equation is used as the system model, which is known as the p-order auto-regression model, as follows:

$$x(n) = -\sum_{k=1}^{p} a(k)x(n-k) + \omega(n) \tag{3}$$

where $x(n)$ is the random sequences, $\omega(n)$ means white noise, and $a(k)$ is the factor. Its corresponding power spectral estimation $P(\omega)$ is:

$$P(\omega) = \frac{\sigma_\omega^2}{|1 + \sum_{k=1}^{p} a(k)e^{-j\omega k}|^2} \tag{4}$$

where σ_ω^2 is the variance of prediction error. We can get the whole model if all $a(k)$ and σ_ω are calculated.

For the solution of parameters, there are two theoretical solutions, the Yule–Walker equation and least-squares method, which will be discussed in the following part. The model is an excellent approximation to the stochastic process, as long as we get a good estimation of the parameters and choose an appropriate order [29–32].

2.2.1. Yule–Walker Equation

From Equation (3), we can obtain the signal correlation:

$$\begin{aligned} r_{xx}(m) &= E[x(n)x(n+m)] \\ &= E\{x(n)[-\sum_{k=1}^{p} a(k)x(n-k+m) + w(n+m)]\} \\ &= -\sum_{k=1}^{p} a(k)r_{xx}(m-k) + E[x(n)w(n+m)]. \end{aligned} \tag{5}$$

Through Equation (5), it can be inferred that $x(n)$ is only related to $w(n)$ and not related to $w(n+m), m \geq 1$. We can obtain the matrix representation as follows.

$$\begin{bmatrix} r_{xx}(0) & r_{xx}(-1) & \cdots & r_{xx}(-p) \\ r_{xx}(1) & r_{xx}(0) & \cdots & r_{xx}(-p+1) \\ \vdots & \vdots & \ddots & \vdots \\ r_{xx}(p) & r_{xx}(p-1) & \cdots & r_{xx}(0) \end{bmatrix} \begin{bmatrix} 1 \\ a(1) \\ \vdots \\ a(p) \end{bmatrix} = \begin{bmatrix} \sigma_w^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

This is the famous Yule–Walker equation, which is the basis of other estimation methods. The coefficients of the AR model and power spectral density can be estimated based on Equation (6). If $\{r_{xx}(k)\}_{k=0}^{P}$ can be computed, we can obtain:

$$r_p + R_p \beta_{YW} = 0, \tag{7}$$

where

$$\beta_{YW} = [a(1), a(2), \cdots, a(P)]^T, \tag{8}$$

$$r_p = [r_{xx}(1), r_{xx}(2), \cdots, r_{xx}(p)], \tag{9}$$

$$R_p = \begin{bmatrix} r_{xx}(0) & \cdots & r_{xx}(-p+1) \\ \vdots & \ddots & \vdots \\ r_{xx}(p-1) & \cdots & r_{xx}(0) \end{bmatrix} \tag{10}$$

β_{YW} can be estimated by:

$$\beta_{YW} = -R_p^{-1} r_p. \tag{11}$$

2.2.2. Least-Squares Method

The least-squares (LS) method, widely used in data processing in engineering technology and scientific research, searches for the best matching function by minimizing the square errors. It is suitable for random and stationary signals.

Suppose there are n points (x_i, y_i) in the dataset, where x_i is the independent variable and y_i is the observation value, $i = 1, 2, \cdots, n$. Let $f(x, \beta)$ be a fitting function for the dataset, where β has m parameters to be estimated. Define the difference r_i between y_i and $f(x_i, \beta)$ as follows,

$$r_i = y_i - f(x_i, \beta). \tag{12}$$

The least-squares method is to minimize the sum of r_i, that is,

$$\min S = \min \frac{1}{2} \sum_{i=1}^{n} r_i^2. \tag{13}$$

Here, we give the estimation result for β in a matrix form directly (see [33] for more details).

$$\hat{\beta} = (X^T X)^{-1} X^T y. \tag{14}$$

In the AR model, Equation (3) could be written in the following formula:

$$y = X\beta + w, \tag{15}$$

where $y = [x_{p+1}, x_{p+2}, \cdots, x_N]^T$, $\beta = [a(1), a(2), \cdots, a(p)]$, $w = [w(p+1), w(p+2), \cdots, w(N)]$ and:

$$X = \begin{bmatrix} x_p & x_{p-1} & \cdots & x_1 \\ x_{p+1} & x_p & \cdots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-1} & x_{N-2} & \cdots & x_{N-p} \end{bmatrix}. \tag{16}$$

In fact, we would obtain the parameters' estimation of the AR model directly through Equation (14). In some cases, the matrix decomposition of X is also a nice solution for β. When X is a full-rank matrix with $X \in R^{m \times n}$, that is, if $m \geq n$, $rank(X) = n$. X can be decomposed by QR factorization,

$$X = QR, \tag{17}$$

where Q is an orthogonal matrix, meaning $Q^T Q = I$, and R is an upper triangular matrix [33]. It can be rewritten as:

$$\hat{\beta} = R^{-1} Q^T y, \tag{18}$$

which means the least-squares method is converted into QR decomposition and matrix inversion. The parallel QR decomposition and matrix inversion [12] should be taken into account in parallel application, which we will not discuss in detail.

3. Review of the AR Model

In this section, we mainly discuss the parallel recursive algorithm and mechanism of high resolution and fast processing of spectral reconstruction based on high-performance parallel computing. Before that, the recursive algorithms for the Yule–Walker equation are discussed first.

3.1. The Recursive Algorithms for the Yule–Walker Equation

It is difficult to estimate all coefficients of the Yule–Walker equation and least-squares method because they need matrix inversion, so that it would add a heavy computational burden when the order P is large. Two famous recursive algorithms, Levinson–Durbin and Burg estimation, have been employed in solving the Yule–Walker equation.

Levinson et al. [34,35] proposed a high-performance recursive algorithm by successively estimating $\{a(1,1), \sigma_1^2\}$, $\{a(2,1), a(2,2), \sigma_2^2\}$, $\cdots$, $\{a(p,1), a(p,2), \cdots, \sigma_P^2\}$. The result with p order is the required solution of Equation (6). The recursive relation is listed as follows.

$$\begin{cases} a(p,p) = -\dfrac{1}{\sigma_{p-1}^2}[r_{xx}(p) + \sum\limits_{k=1}^{p-1} a(p-1,k) r_{xx}(p-k)] \\ a(p,k) = a(p-1,k) + a(p,p) a(p-1,p-k), k = 1,2,\cdots,p-1 \\ \sigma_p^2 = [1 - a(p,p)^2]\sigma_{p-1}^2, \sigma_0^2 = r_{xx}(0) \end{cases} \tag{19}$$

The Levinson–Durbin (L-D) recursive algorithm is used to solve the coefficients of the AR model. Although it can simplify computation, we still need to know the autocorrelation sequence $r_{xx}(k)$, which could only be calculated from finite time series. When the data length N is small, the computation error will become large.

Under the constraint of the L-D recursive relation, Burg et al. proposed a recursive algorithm to minimize the sum of a priori and a posteriori prediction error energy to evaluate all coefficients.

According to the linear prediction theory, $\hat{x}$ can be indicated by the weighted sum of the previous value, that is,

$$\hat{x}(n) = -\sum_{k=1}^{p} a(p,k) x(n-k). \tag{20}$$

The a priori prediction error can be written as:

$$e(p,n) = x(n) - \hat{x}(n) = x(n) + \sum_{k=1}^{p} a(p,k) x(n-k). \tag{21}$$

In Equation (19), let the reflection coefficient $k(p) = a(p,p)$, and define the a posteriori prediction error:

$$b(p,n) = x(n-p) + \sum_{k=1}^{p} a(p,k)x(n-p+k), \tag{22}$$

the recursive formula is

$$e(p,n) = e(p-1,n) + k(p)b(p-1,n-1), \tag{23}$$

similarly,

$$b(p,n) = b(p-1,n-1) + k(p)e(p-1,n). \tag{24}$$

For better estimation of $k(p)$, the Burg algorithm is based on the principle of minimum mean squared error, i.e., $\min[e^2 + b^2]$. $k(p)$ is evaluated by:

$$k(p) = -\frac{2E[e(p-1,n)b(p-1,n-1)]}{E[e^2(p-1,n) + b^2(p-1,n-1)]}. \tag{25}$$

For a stationary random process,

$$\hat{k}(p) = -\frac{2\sum_{n=p}^{N-1} e(p-1,n)b(p-1,n-1)}{\sum_{n=p}^{N-1}[e^2(p-1,n) + b^2(p-1,n-1)]} \tag{26}$$

and:

$$e(0,n) = b(0,n) = x(n), \tag{27}$$

$$\hat{\sigma}_0^2 = \hat{r}_{xx}(0) = \frac{1}{N}\sum_{n=0}^{N-1} x^2(n). \tag{28}$$

All parameters could be calculated from the following iterative relation.

$$\begin{cases} a(p,k) = a(p-1,k) + \hat{k}(p)a(p-1,p-k) \\ \sigma_p^2 = [1-\hat{k}^2(p)]\sigma_{p-1}^2 \end{cases} \tag{29}$$

3.2. Parallel Burg Recursive Algorithm

The AR model for spectral estimation would provide a higher resolution; meanwhile, it needs more time to compute the parameters due to its high algorithm complexity, especially for large data and batch processing. In order to process huge data using the high-resolution method for spectrum reconstruction in a more effective way, we parallelized the Burg method for the extension of its applicability. The parallel Burg (P-Burg) algorithm mainly contains two parts: initialization and update.

Both initialization and update involve the dot product on the GPU. Given two vectors G and H with the same length N, we calculate the dot product through:

$$V = G \cdot H = \sum_{i=0}^{N-1} g_i h_i. \tag{30}$$

For threads on a GPU, some of them are used for summation reduction besides completing the computing task of multiplying the corresponding elements. In summation reduction, since each thread combines two entries into one, we complete this step with half as many entries as we started. In the next step, we do the same thing on the remaining half. Thus, it can be seen that there would be $\log N$ steps with N (N is a power of two). After these steps of summation reduction, the final result would be recorded in v_0, illustrated by Figure 2. Specifically, shared memory could be used for faster addressing and better performance. When the summation reduction begins, we declare a buffer of shared memory in a block, which will be used to store each thread's running sum. When all the threads finish the

computation task, the buffer is filled up. Then, we can sum the values in the buffer to accomplish the reduction, as Figure 2 shows. For example, if we use 256 threads per block, 256 values from the shared memory would be used to form 128 values through summing every two values into one. In the second iteration, 128 values would be summed into 64 values. Until the last iteration, the reduction is completed. It will take eight iterations to reduce the 256 values into a single sum.

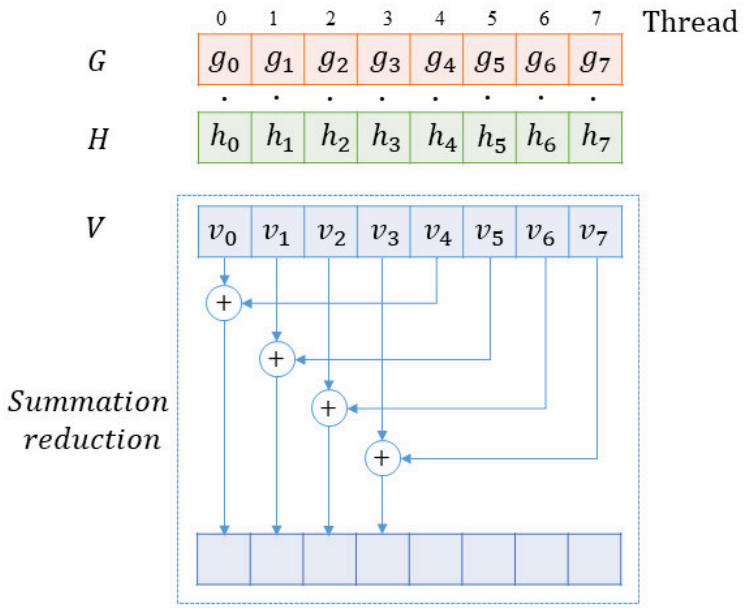

Figure 2. Parallel dot product for two vectors and a step of summation reduction on the GPU.

Specifically, slightly different from the representation in Section 2, in this section, e, b, a, k are considered as row vectors, and $a_m(j)$ means the j^{th} element of the vector a in the m^{th} iteration. Furthermore, the index of a vector starts from zero.

3.2.1. Initialization

We usually initialize the parameters by:

$$e_0(tid) = b_0(tid) = x(tid), \tag{31}$$

$$\sigma_0^2 = \frac{1}{N} x \cdot x, \tag{32}$$

where tid means the index of the current thread and $tid = 0, 1, \cdots, N-1$ means that we arrange N threads to compute the dot product. The first element of a, that is $a(0)$, is set to one.

3.2.2. Update

After initialization, the parameters will be updated iteratively. In one iteration, the reflection coefficient k is firstly updated by:

$$\hat{k}(m) = -\frac{2e_{m-1} \cdot b_{m-1}}{e_{m-1} \cdot e_{m-1} + b_{m-1} \cdot b_{m-1}}. \tag{33}$$

Then, $a(p)$, e_p, b_p and σ_p^2 are updated at the same time through:

$$e_m(tid) = e_{m-1}(tid+1) + \hat{k}(m) b_{m-1}(tid+1), \tag{34}$$

$$b_m(tid) = b_{m-1}(tid) + \hat{k}(m) e_{m-1}(tid), \tag{35}$$

$$a_m(j) = a_{m-1}(j) + \hat{k}(m)a_{m-1}(m-j-1), \tag{36}$$

$$\sigma_m^2 = [1-\hat{k}(p)^2]\sigma_{m-1}^2, \tag{37}$$

where $j = 0, 1, \cdots, m$, m indicates the iteration number, $m = 1, 2, \cdots, p$, and tid is the index of the current thread, $tid = 0, 1, \cdots, N-p$. Figure 3 shows the operation flow of the P-Burg method in one iteration on a GPU.

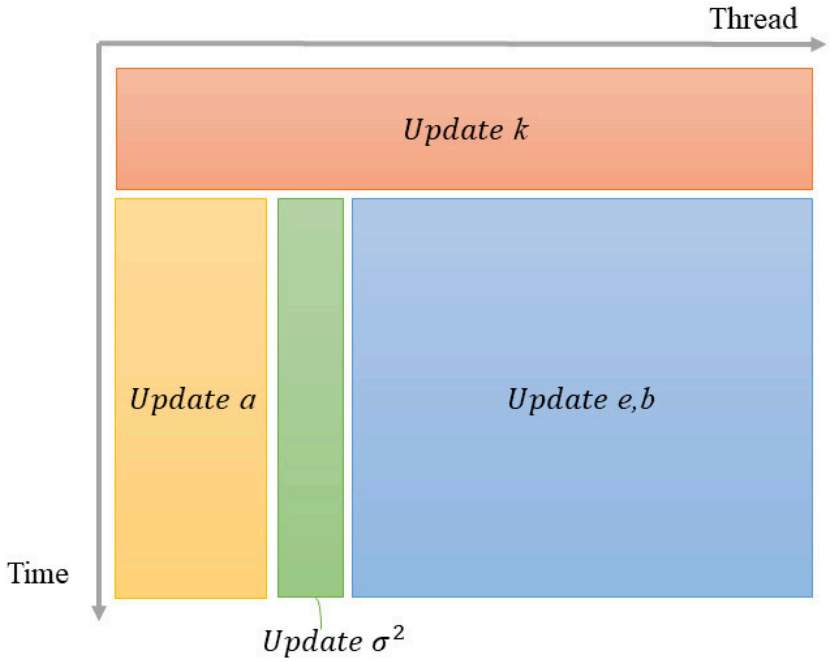

Figure 3. The flowchart of the parallel update in one iteration.

When we update a based on Equation (36), it should be updated for all $a_p(j)$ at the same time, and we can set m threads to update the m parameters, listed as follows,

$$a_m(tid) = a_{m-1}(tid) + \hat{k}(p)a_{m-1}(m-1-tid), \tag{38}$$

where $tid = 0, 1, \cdots, m-1$.

It should be noted that model order p determines the number of iterations, and we do not take the cuBLAS library (an implementation of basic linear algebra subprograms on top of CUDA runtime) into account. The reasons are listed as follows. We do not have enough sampling points from the interferogram in the experiments for a long dot reduction. At this order of magnitude, our routine for dot product computation has almost the same performance as the one included in the cuBLAS library. When enough data are available, the performance is much better with the dot product in the cuBLAS library. Another important reason is that in Equation (33), we need three different dot products, and there are operations between the three values. If we used cuBLAS, we would still need to create a new kernel for the only three values, which would be hardly efficient on the GPU. In our routine, we can get the result of the three dot products at the same time and calculate them in the same kernel. This is more flexible in solving our problems.

3.3. Parallel Processing Mechanism

Generally speaking, parallel tasks are completed by the collaboration of the CPU and GPU, where the CPU is responsible for flow control and data cache, while the GPU is dedicated to solving problems that can be represented as data parallel computing, that is the same programs are executed in a parallel way on many data elements. However, resources on the GPU are limited, especially for batch

processing. With the limited memory and threads, we take advantage of the characteristics of the GPU with overlap to design an asynchronous parallel processing mechanism for spectrum reconstruction when it comes to batch processing.

Stream plays an important role in accelerating applications. It represents a GPU operation queue where the operations will be executed in a specified order. We can add some operations in the stream, such as kernel function, memory copy, etc. The order in which these operations are added to the stream is also their execution order. Each stream can be thought of as a task on the GPU, and these tasks can be executed in parallel.

Figure 4 shows an example of the parallel processing mechanism based on four streams. In the figure, the kernel in the serial pipeline comprises four small P-Burg kernels, which are implemented one after the other. In the parallel pipeline, the four P-Burg kernels are arranged in the four streams, respectively, which could be executed in an asynchronous way. In synchronization mode, we successively execute the memory copy from the host (CPU) to the device (GPU) and the kernel and memory copy from the device to the host. However, in the asynchronous parallel processing mechanism, tasks are divided into several streams. When Stream 0 executes a kernel, data transfer from the host to the device in Stream 1, and when Stream 1 executes the kernel, memory copies from the device to the host and from the host to the device are executed in Stream 0 and Stream 2, respectively. Thus, the performance improves because the GPU that support overlap can execute the memory copy between the device and the host while executing our P-Burg kernel function.

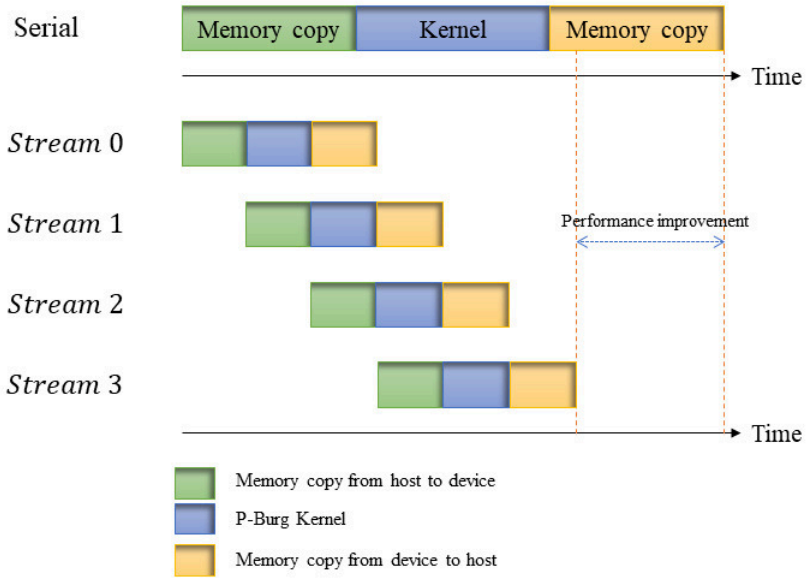

Figure 4. Example of the parallel processing mechanism based on overlap.

3.4. Comparison with FFT

The interferogram cannot be obtained ranging from negative infinity to positive infinity in Equations (1) and (2). The optical path difference Δ in practical application satisfies the following relation,

$$-L \leq \Delta \leq +L, \tag{39}$$

where L is the maximum optical path difference. In Fourier transform spectroscopy, the resolution of FFT is the reciprocal of the data length N. This value, which falls out of the scanning range, is considered as zero, which is equivalent to multiplication of the signal and a rectangular window. In the spectrum, a rectangular window has many side lobes, leading to spectrum leakage. We could choose an appropriate function such as the Hamming window, the triangular window, the Happ–Genzel

window and Bessel window in apodization technology. Through apodization with a window function, the FHWM increases. Meanwhile, the AR model is to estimate the interferogram rational value out of the scanning range instead of zeros according to its principle. From this point, this means more data are used to reconstruct the spectrum, and then, the resolving power is improved. In addition, the model solved by P-Burg has fewer side lobes, and its FWHM is smaller than the one of FFT, which will be verified in Section 4.

In terms of the operation time of the algorithms, FFT has the best performance because of its low complexity ($NlogN$). Even using our parallel Burg recursive solution, the model still needs much more time to estimate the parameters.

4. Experiments

Our experiments were implemented using C/C++ and CUDA C under Ubuntu 16.04. The computer configuration was as follows: Inter Xeon CPU E5-2609, 16 GB RAM, and Quadro K620 graphics card. Table 1 lists more information about the K620 card.

Table 1. Details of Quadro K620.

CUDA driver version	9.0
CUDA capability	5.0
Global memory	1992 MB
GPU Max clock rate	1124 MHz
Shared memory per block	49,152 B
Max dimension of a block	(1024 1024 64)
Max dimension of a grid	(2,147,483,647 65,535 65,535)

The famous JPL Lab provides the spectrum for our simulations, and it could be seen as an ideal spectrum, shown by Figure 5. Its corresponding interferogram was transformed, which could be seen as an ideal interferogram. Our actual laser and white light interferograms were provided by the LASIS interferometer in our practical application, as shown in Figures 6 and 7 respectively.

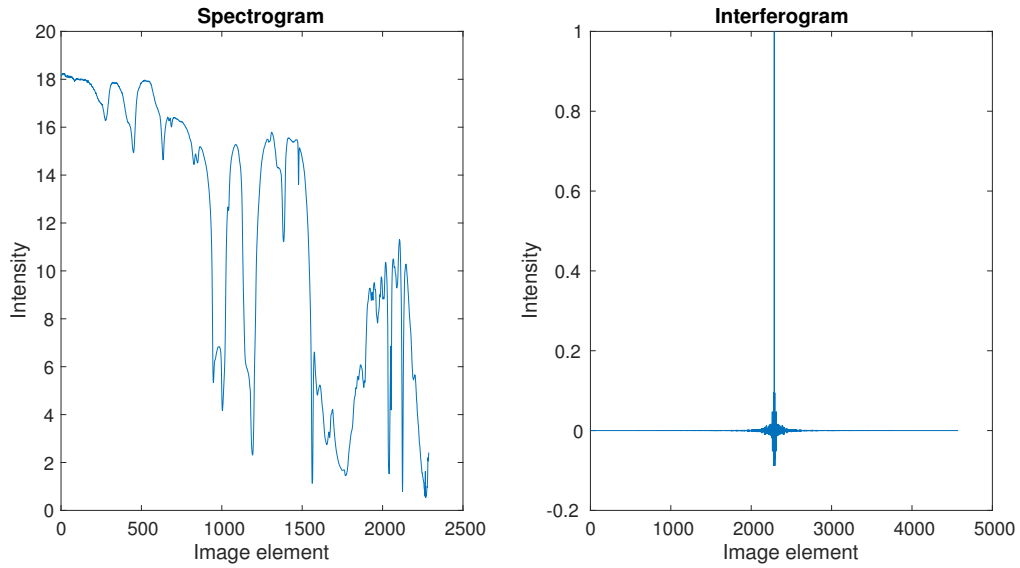

Figure 5. Spectrogram and its corresponding interferogram from JPL Lab.

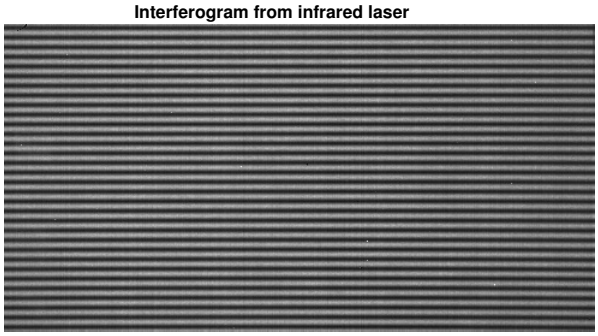

Figure 6. A frame of the laser interferogram whose wavelength is 1550 nm.

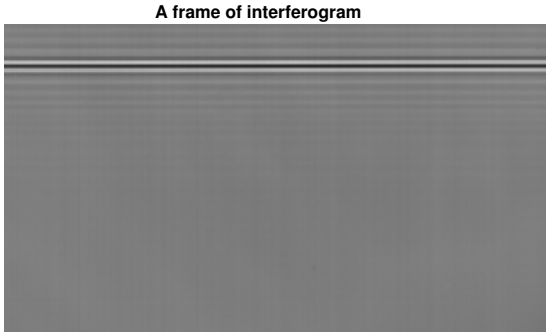

Figure 7. A frame of the white light interferogram from LASIS.

4.1. Reconstruction Result

We compared the Yule–Walker and LS methods to solve AR model using the ideal interferogram from Figure 5. The result of spectrum reconstruction is illustrated in Figure 8. We can see that both the Yule–Walker method and LS method provided a high level consistency, and there was little difference between them. It should be noticed that the construction spectrum using the AR model was a power spectrum, and the general tendency of the original spectrum was recovered by the two methods.

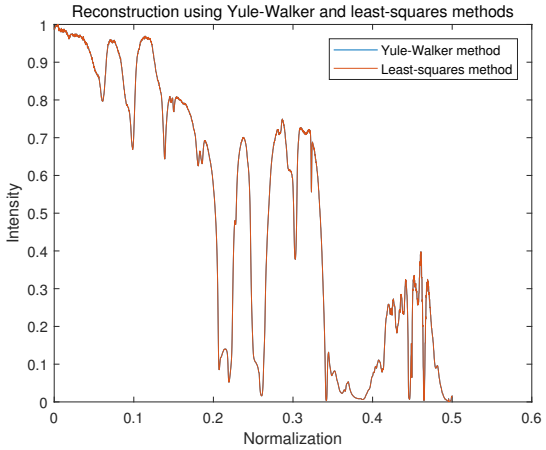

Figure 8. Spectrum reconstruction using the Yule–Walker method and least-squares method.

Figure 9 shows the result of two Yule–Walker recursive algorithms, the Levinson–Durbin algorithm, and the Burg (P-Burg) algorithm. It can be seen that these two methods could estimate the parameters of AR well, and because matrix inversion was not involved during the solving process, both were much faster than Yule–Walker and LS. The reconstruction results from Figures 8 and 9 are almost the same.

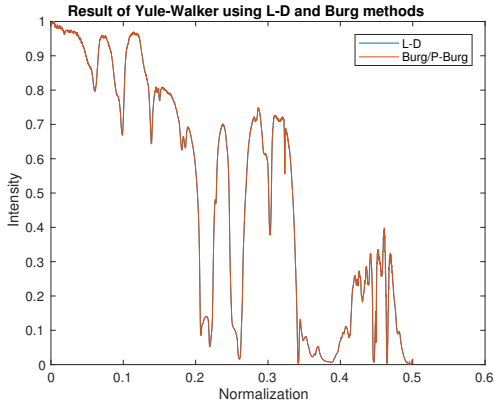

Figure 9. The result of the spectrum reconstruction using the Yule–Walker recursive algorithm, Levinson–Durbin (L-D), and Burg/parallel Burg (P-Burg).

4.2. FFT and P-Burg

Figure 10 shows the result of spectrum reconstruction using FFT and P-Burg for a simulated interferogram. We can see that the P-Burg method can identify the two spectrum peaks (Figure 10d), while there is only one peak in the power spectral density using FFT (Figure 10c). FFT can only identify them when the length of the signal increases. Figure 11 shows the reconstruction result of the laser interferogram (Figure 6). Obviously, the spectrum of FFT has more side lobes and larger FWHM than the one reconstructed by P-Burg. It can be verified that P-Burg has better performance in power resolution for spectrum reconstruction from the two figures.

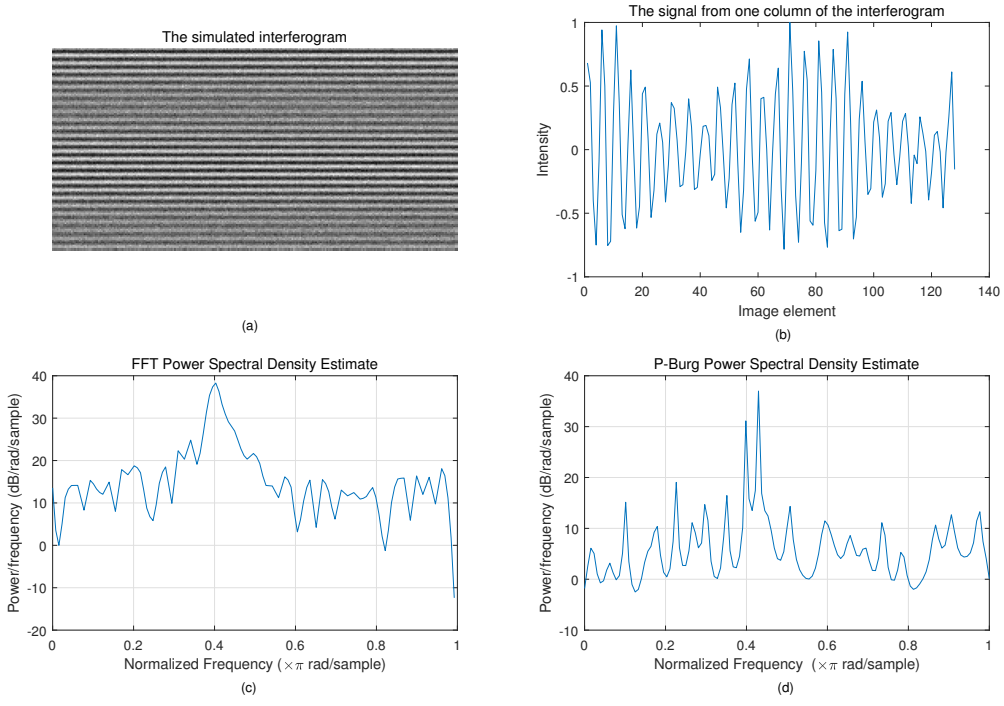

Figure 10. Spectrum reconstruction using FFT and P-Burg for a signal from the simulated interferogram. (**a**) The simulated interferogram. (**b**) A signal from one column of the interferogram. It consists of two different-frequency sine signals, whose frequency is very close, and white noise. The length of the signal is 128. (**c**) The power spectral density using FFT. (**d**) The power spectral density using P-Burg.

Table 2 shows the runtime of FFT and P-Burg for our laser interferogram where the length $N = 256$ and the model order p is set to five. The FFT has the best performance in runtime. Although FFT is much faster to perform than P-Burg by one order of magnitude, P-Burg tries to keep the high resolution in spectrum reconstruction while meeting the real-time requirements. An interesting thing is that the parallel FFT on the GPU runs slower than the FFT on the CPU. This is because when N is small, the time consumed by communication between threads can be comparable to the computation time. This leads to a decline in parallel FFT performance.

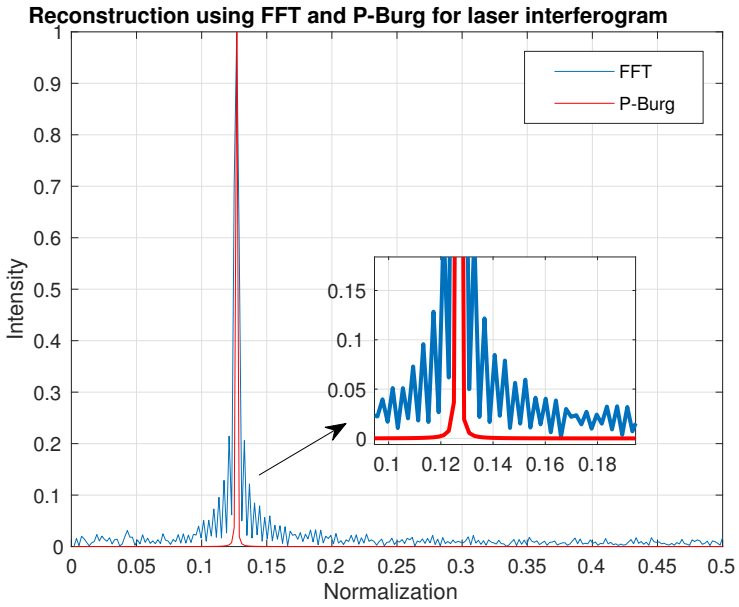

Figure 11. Spectrum reconstruction using FFT and P-Burg for the laser interferogram.

Table 2. Runtime of P-Burg and FFT with apodization.

Method	Runtime (ms)
P-Burg	0.116
FFT with apodization	0.011
Parallel-FFT with apodization	0.020

4.3. Performance of Burg and Parallel Burg

Figure 12 shows the runtime cost by Burg and parallel Burg with the increase of the model order p. The length N is 4573. For every order p, we ran the algorithms a thousand times and measured the performance by the average of the total time for the two methods, respectively. The runtime cost by the Burg and parallel Burg methods could increase as the model order becomes larger. In addition, the time cost by parallel Burg was much lower than that cost by Burg, and it can be concluded that the parallel Burg algorithm had a higher performance in algorithm efficiency compared with the traditional Burg method.

Table 3 shows the ratio and improvement between Burg and parallel Burg. The ratio, improvement, and MSE were calculated by the following equation:

$$ratio = \frac{Time_{Burg}}{Time_{P-Burg}}, \tag{40}$$

$$improvement = \frac{Time_{Burg} - Time_{P-Burg}}{Time_{Burg}}. \tag{41}$$

$$MSE(a) = \frac{1}{p+1}\sum_{i=0}^{p}[a(i)_{P-Burg} - a(i)_{Burg}]^2. \quad (42)$$

In the table, the time consumption of Burg is about 3.8-times that of P-Burg with the model order ranging from 1000–2000, that is the efficiency of P-Burg was 3.8-times as high as that of Burg. The improvement was about 74%, which means the performance increased by 74%. In terms of the estimation accuracy, the estimated parameters of P-Burg was almost the same as the serial result. The difference was only the system calculation error. In fact, the other solutions of AR model had consistent parameter estimation when the length N and the order p were the same. This is, in total, a satisfactory result.

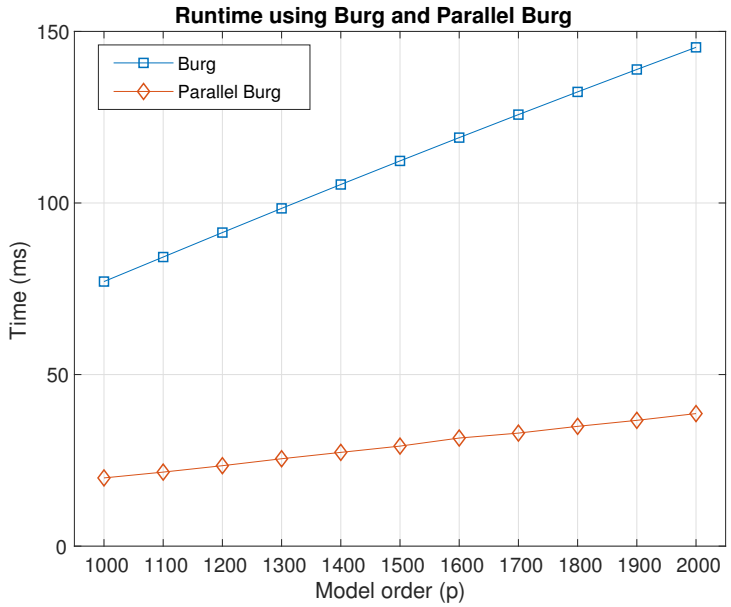

Figure 12. Runtime between the Burg algorithm and the parallel Burg algorithm with model order.

Table 3. Performance comparison between Burg and parallel Burg.

p	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
ratio	3.88	3.90	3.90	3.86	3.86	3.85	3.78	3.82	3.79	3.79	3.76
improvement (%)	74.2	74.4	74.3	74.1	74.1	74.0	73.5	73.8	73.6	73.6	73.4
MSE ($\times 10^{-32}$)	8.502	7.545	7.331	6.764	7.215	7.403	5.498	6.289	4.890	5.263	4.804

Despite the high performance of P-Burg, it should not be forgotten that the parallel Burg method was executed on the GPU and occupied many GPU resources, including memory and threads. That is the advantage of the GPU in accelerating applications.

4.4. Batch Processing for Large Data Using P-Burg

The GPU is a powerful tool for dealing with large amounts of data, and it plays a very important role in batch processing or data parallelism. That means if we compute an original signal with length N, then we have T groups of similar signals, and these groups of signals are operated the same as the original one. In our experiment and simulation, we set $N = 4573$, and T ranged from 10–100. Compared with the CPU, the GPU had better performance for computing, proven by Table 4, and the efficiency ratio was about 5–10.

Table 4. Batch processing on CPU and GPU.

Groups	10	20	40	60	80	100
CPU (s)	1.119	2.244	4.490	6.754	8.959	11.250
GPU (s)	0.135	0.290	0.858	1.311	1.758	2.214

In the table, we can see that the time required for processing on the CPU linearly increased as the value of T were from 10–100, while the runtime cost on the GPU also increased gradually. That may be different from the ideal situation since more and more threads and memory are configured to complete the task, and it should not increase with T. The reason is that thousands of threads need to collaborate with each other, and these threads do not work at the same time, though they are scheduled simultaneously. Meanwhile, the more data, the more the address operations, which would result in much time delay. Furthermore, large data transmission takes much time including data copy from the CPU to the GPU and from the GPU to the CPU.

It should be recognized that we have enough resources for each group of data.

Table 5 shows some parameters from profiling information about memory transfers and GPU utilization (the number of groups was set to 100), where HtoD means memory copy from the host to device and DtoH means memory copy from the device to host. The computation occupies most of the time. In total, the parallel pipeline we designed performed well. However, the memory copies did not fully use the available host to device bandwidth because the throughput of HtoD was only 6.305 GB/s. It could be better optimized for larger throughput.

Table 5. Some details of profiling about GPU utilization. HtoD, host to device; DtoH, device to host.

HtoD: total bytes	12.174 MB
HtoD: throughput	6.305 GB/s
DtoH: total bytes	2.402 MB
DtoH: throughput	6.427 GB/s
Compute utilization	79.9%
Dot kernel proportion	43.6%
Sum kernel proportion	1.2%
Update kernel proportion	55.2%

4.5. Parallel Mechanism with Overlap

With the resource limitation on the GPU, it should be realized that more workload will be arranged on the threads for batch processing. In other words, if we use n threads working on one group of data, with the limitation of threads, we could only use n threads for T groups, that is each thread will be responsible for more calculations.

Figure 13 describes the performance of different parallel processing mechanisms. P-Burg without overlap means that it maintains the data copy from the host to the device, the kernel calculation and data copy from the device to the host, which is the specific execution order for many groups on the same threads without multiple streams. We take advantage of overlap and use multiple streams for P-Burg with m streams, $m = 2, 3, 4, 8$.

From Figure 13, under different data groups, the performance of the parallel processing mechanism based on multi-stream was higher than that of common parallel mode. Table 6 shows the performance improvement using different numbers of streams, compared with the serial mechanism (one stream). The multiple streams on the GPU could improve the processing efficiency by about 15%–25% for many batches of data. Compared with P-Burg with different numbers of streams, the performance is almost the same, except the two streams. Strictly speaking, the parallel mechanism of P-Burg with three streams may have higher performance with the increasing amount of data.

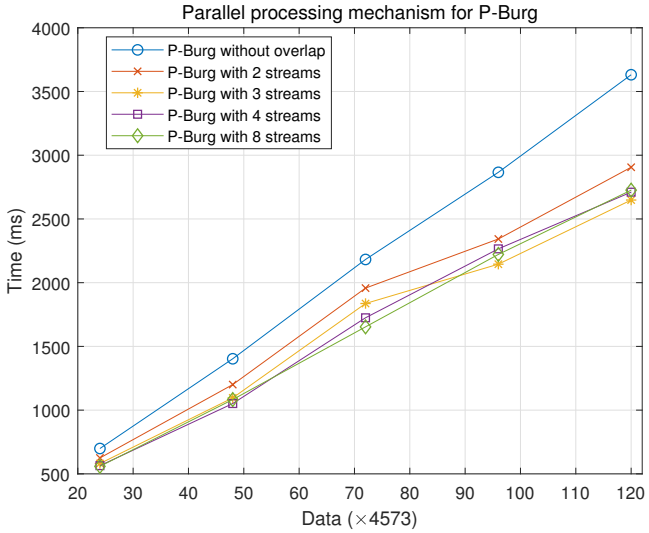

Figure 13. Runtime of the parallel processing mechanism for P-Burg with the limitation of threads and memory.

Table 6. Performance improvement in different numbers of streams (%).

Groups	24	48	72	96	120
2 streams	10.59	14.43	10.29	18.25	19.99
3 streams	16.86	21.87	15.84	25.15	27.07
4 streams	19.28	25.05	20.97	20.95	25.37
8 streams	20.04	22.81	24.23	22.48	24.93

4.6. Practical Application

Figure 14 shows the reconstruction of a frame of the interferogram from Figure 7 using the P-Burg method. Before that, we removed the trend item of interferogram and corrected the phase error. The figure is a 3D depiction arranging all the transform results of each column in Figure 7 together. We get the result of a frame within only about 50 ms.

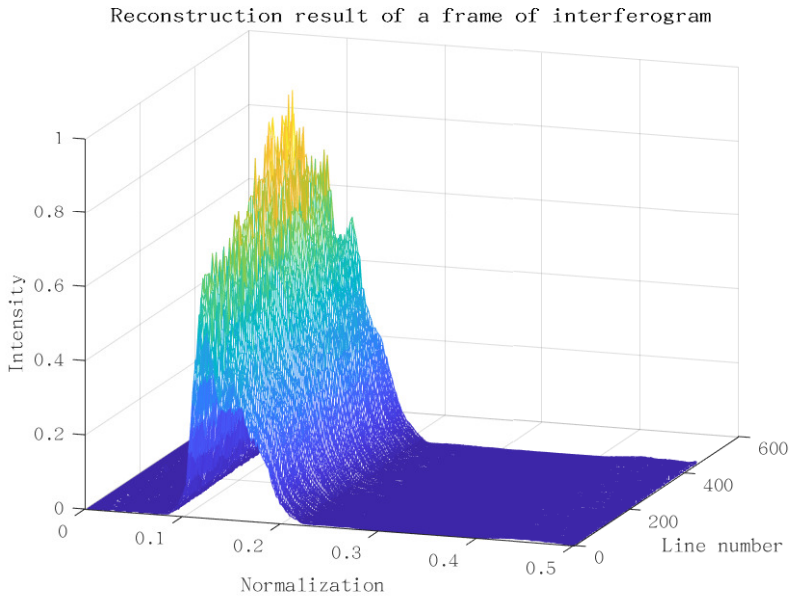

Figure 14. Processing result of a frame of the interferogram.

5. Conclusions

In the field of spectrum reconstruction, both the resolution and time spent in data processing are of vital importance. However, high resolution spectral reconstruction has high algorithm complexity. It needs more time to complete the task.

In this paper, a parallel high resolution algorithm in spectrum reconstruction dealing with the interferogram has been explored for fast operation. To make full use of the resources of the GPU, we design spectrum reconstruction with the P-Burg method based on the overlap. In addition, the Yule–Walker and least-squares methods solved for the AR model have been discussed for comparison. The analytical and experimental results are in good agreement. It is important to note that the AR model is sensitive to model order, though it has higher resolution in spectrum analysis. Therefore, it is necessary to select an appropriate model order using model selection criteria such as AIC, BIC, etc.

Author Contributions: W.Z. proposed the parallel Burg algorithm for high resolution spectrum reconstruction and designed the simulations and experiments. D.W., Z.S., X.W., G.L., and Z.L. helped to revise the experimental methods and article structure.

Funding: The work presented in this paper was fully supported by a grant from the Youth Innovation Promotion Association (No. 1188000111).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Marshall, A.G.; Verdun, F.R. *Fourier Transforms in NMR, Optical, and Mass Spectrometry: A User's Handbook*; Elsevier Science: Oxford, UK, 2016.
2. Köhler, M.H.; Schardt, M.; Rauscher, M.S.; Koch, A.W. Gas Measurement Using Static Fourier Transform Infrared Spectrometers. *Sensors* **2017**, *17*, 2612. [CrossRef] [PubMed]
3. Mueller, D.; Ferrão, M.F.; Marder, L.; da Costa, A.B.; de Cássia de Souza Schneider, R. Fourier Transform Infrared Spectroscopy (FTIR) and Multivariate Analysis for Identification of Different Vegetable Oils Used in Biodiesel Production. *Sensors* **2013**, *13*, 4258–4271. [CrossRef] [PubMed]
4. Réhault, J.; Borrego-Varillas, R.; Oriana, A.; Manzoni, C.; Hauri, C.P.; Helbing, J.; Cerullo, G. Fourier transform spectroscopy in the vibrational fingerprint region with a birefringent interferometer. *Opt. Exp.* **2017**, *25*, 4403-4413. [CrossRef] [PubMed]
5. Amenabar, I.; Poly, S.; Goikoetxea, M.; Nuansing, W.; Lasch, P.; Hillenbrand, R. Hyperspectral infrared nanoimaging of organic samples based on Fourier transform infrared nanospectroscopy. *Nat. Commun.* **2017**, *8*, 14402. [CrossRef] [PubMed]
6. Xiangli, B.; Cai, Q.; Du, S. Large aperture spatial heterodyne imaging spectrometer: Principle and experimental results. *Opt. Commun.* **2015**, *357*, 148–155. [CrossRef]
7. Ferrec, Y.; Taboury, J.; Sauer, H.; Chavel, P.; Fournet, P.; Coudrain, C.; Deschamps, J.; Primot, J. Experimental results from an airborne static Fourier transform imaging spectrometer. *Appl. Opt.* **2011**, *50*, 5894–5904. [CrossRef] [PubMed]
8. Bell, R. *Introductory Fourier Transform Spectroscopy*; Elsevier Science: Saint Louis, MO, USA, 2012.
9. Zhang, C.; Jian, X. Wide-spectrum reconstruction method for a birefringence interference imaging spectrometer. *Opt. Lett.* **2010**, *35*, 366–368. [CrossRef] [PubMed]
10. Su, L.; Yuan, Y.; Xiangli, B.; Huang, F.; Cao, J.; Li, L.; Zhou, S. Spectrum Reconstruction Method for Airborne Temporally–Spatially Modulated Fourier Transform Imaging Spectrometers. *IEEE Trans. Geosci. Remote Sens.* **2014**, *52*, 3720–3728. [CrossRef]
11. Zhang, W.; Wen, D.; Song, Z. Spectrum reconstruction in interference spectrometer based on sparse Fourier transform. *Optik* **2018**, *154*, 157–164. [CrossRef]
12. Zhang, W.; Wen, D.; Song, Z; Liu, G.; Wei, X.; Li, Z. Spectrum reconstruction in Fourier transform imaging spectroscopy based on high-performance parallel computing. *Appl. Opt.* **2018**, *57*, 5983–5991. [CrossRef] [PubMed]

13. Wang, S.; Li, L.B.; Pi, H.F. Research of Spectrum Signal-to-Noise Ratio of Large Aperture Static Imaging Spectrometer. *Spectrosc. Spectr. Anal.* **2014**, *3*, 851–856.
14. Jian, X.; Zhang, C.; Zhao, B.; Zhu, B. The application of MUSIC algorithm in spectrum reconstruction and interferogram processing. *Opt. Commun.* **2008**, *281*, 2424–2428. [CrossRef]
15. Han, G.; Liu, X.; Hu, B.; Wang, C. Interferogram spectrum reconstruction using modern spectral estimation. In Proceedings of the 2011 International Conference on Electronics, Communications and Control (ICECC), Ningbo, China, 9–11 September 2011; Volume 281, pp. 2112–2115.
16. Schmidt, R. Multiple emitter location and signal parameter estimation. *IEEE Trans. Antennas Propag.* **1986**, *34*, 276–280. [CrossRef]
17. Antoniou, A. *Digital Signal Processing*; McGraw-Hill Education: New York, NY, USA, 2016.
18. Box, G.E.P.; Jenkins, G.M.; Reinsel, G.C.; Ljung, G.M. *Time Series Analysis: Forecasting and Control*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
19. Cheng, J.; Grossman, M.; McKercher, T. *Professional Cuda C Programming*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
20. Cook, S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*; Elsevier Science: Oxford, UK, 2012.
21. Sanders, J.; Kandrot, E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*; Addison-Wesley Professional: Boston, MA, USA, 2010.
22. Griffiths, P.R.; De Haseth, J.A. *Fourier Transform Infrared Spectrometry*; Wiley-Interscience: Hoboken, NJ, USA, 2007.
23. Norton, R. H; Beer, R. New apodizing functions for Fourier spectrometry: Errata. *JOSA* **1977**, *67*, 419. [CrossRef]
24. Egbert, S.; Carpenter, C. Comparison of FTIR apodization functions using modeled and measured spectral data. *J. Appl. Eng. Math.* **2017**, *4*, 1–4.
25. Gero, J.; Revercomb, H.; Tobin, D.; Knuteson, R.; Taylor, J. A Highly Accurate Correction for Self Apodization Effects on Fourier Transform Spectrometer Spectra. In *Light, Energy and the Environment 2018 (E2, FTS, HISE, SOLAR, SSL), OSA Technical Digest (Optical Society of America, 2018)*; Paper FW2B.4; OSA: Washington, DC, USA, 2018.
26. Forman, M.L.; Steel, W.H.; Vanasse, G.A. Correction of Asymmetric Interferograms Obtained in Fourier Spectroscopy. *JOSA* **1966**, *56*, 59–63. [CrossRef]
27. Sanderson, R.B.; Bell, E.E. Multiplicative correction of phase errors in Fourier spectroscopy. *Appl. Opt.* **1973**, *12*, 266–270. [CrossRef] [PubMed]
28. Smith, B.C. Fundamentals of Fourier Transform Infrared Spectroscopy; CRC Press: Boca Raton, FL, USA, 2011.
29. Broersen, P.M.T. Finite sample criteria for autoregressive order selection. *IEEE Trans. Signal Process.* **2000**, *48*, 3550–3558. [CrossRef]
30. De Waele, S.; Broersen, P.M.T. Order selection for vector autoregressive models. *IEEE Trans. Signal Process.* **2003**, *51*, 427–433. [CrossRef]
31. Fenga, L.; Politis, D.N. LASSO order selection for sparse autoregression: a bootstrap approach. *J. Stat. Comput. Simul.* **2017**, *87*, 2668–2688. [CrossRef]
32. Montgomery, D.C; Jennings, C.L.; Kulahci, M. *Introduction to Time Series Analysis and Forecasting*; Wiley: Hoboken, NJ, USA, 2015.
33. Zhang, X. *Matrix Analysis and Applications*; Cambridge University Press: Cambridge, UK, 2017.
34. Levinson, N. The Wiener (Root Mean Square) Error Criterion in Filter Design and Prediction. *J. Math. Phys.* **1946**, *25*, 261–278. [CrossRef]
35. Durbin, J. *The Fitting of Time-Series Models*; Review of the International Statistical Institute; International Statistical Institute: Voorburg, The Netherlands, 1960; pp. 233–244.