Research on GPU Parallel Acceleration of Efficient Coherent Integration Processor for Passive Radar

Zirong Bu[1,2], Lijun Wang[1,2], and Huijie Zhu[1,2](✉)

[1] Science and Technology on Communication Information Security Control Laboratory, Jiaxing 314033, China

[2] No. 36 Research Institute of China Electronics Technology Group Corporation, Jiaxing 314033, China

Abstract. The traditional algorithm of cross-ambiguity function requires a large amount of computation and storage capacity, which brings difficulties to real-time processing. In addition, the long-integration time will cause range migration problems, resulting in a decrease in the SNR, and reducing the weak target detection ability of the system. Based on the characteristics of the passive radar, this paper adopts the method of combining segmental frequency domain pulse compression and Keystone transform to correct the range migration in the target detection, which improves the detection ability of weak targets. However, due to the relatively large amount of data and computation of the algorithm, this paper takes the advantages of graphics processing unit (GPU) with large data throughput and strong floating-point computing capabilities to propose an efficient coherent integration method which is suitable for GPU parallel processing.

Keywords: Passive radar · Range migration · Graphics processing unit (GPU)

1 Introduction

Passive radar exploits the opportunity signal in the environment for target detection, which has the advantages of high concealment, low power consumption, high concealment, and strong anti-interference ability [1]. The core problem of passive radar is to detect weak targets in strong interference. Usually, the direct wave and clutter signals received by the radar are much stronger than the target echo. After the direct wave and clutter are suppressed, the time delay-Doppler two-dimensional correlation (ambiguity function) still needs to be calculated to obtain accumulated benefits. However, the traditional algorithm of cross-ambiguity function requires a large amount of computation and storage capacity, which brings difficulties to real-time processing. In addition, the long-integration time will cause range migration problems, resulting in a decrease in the SNR, and reducing the weak target detection ability of the system. this paper proposes an efficient integration integration method which is suitable for GPU parallel processing.

X. Wang et al. (Eds.): AICON 2021, LNICST 396, pp. 415–422, 2021.
https://doi.org/10.1007/978-3-030-90196-7_35

In recent years, with the development of large-scale integrated circuit technology, the performance of the GPU has increased day by day, which provides a new solution for real-time signal processing of passive radar. Compared with traditional solutions such as LPGA, DSP and so on. GPU has the advantages of low cost, simple structure and easy development. A large number of research at home and abroad have confirmed the feasibility of real-time processing of passive radar signal using high-performance CPUs and GPUs. The coherent integration algorithms are more suitable for GPU implementation because GPU can be calculated in parallel, and has the advantages of sufficient storage space, large data throughput, and strong floating-point operating capabilities.

2 Efficient Coherent Integration Processor with Range Migration Compensation

2.1 Segmented Frequency Domain Pulse Compression (FDPC) of Passive Radar

Due to the long-integration time, the calculation amount of direct coherent integration is very huge, the segmented FDPC method can greatly reduce the computational complexity [4]. The signal is divided into segments of equivalent pulses, which can be understood as analog active radar pulse processing, and the continuous signal is divided into the form of fast-slow time. The segmentation method of the reference signal and the echo signal is shown in Fig. 1. Determine the slow time segment number M according to the maximum radial velocity of the target, so as to avoid the phenomenon of spanning the distance gate caused by the target movement during the integration time. The length of the reference signal within the segment is N, and the length of the echo signal within the segment is $N_t = N + N_d$, and N_d is determined by the maximum delay of system detection. Among them, the segmentation of the reference signal adopts the method of zero-filling at the end of the non-overlapping segment, and the echo signal adopts the method of overlapping segmentation.

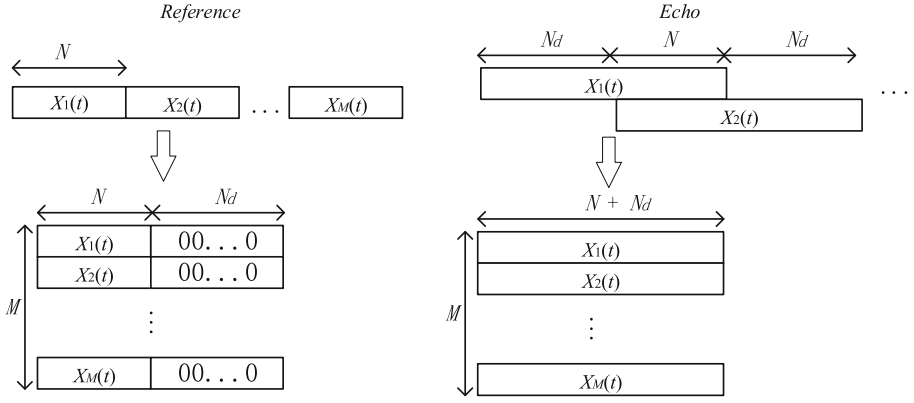

Fig. 1. Segmentation of reference signal and echo signal

Combining the above motion model and the continuous signal segmentation method, the fast-slow time form of the target echo baseband signal is expressed as:

$$s_e(t_n, t_m) = Au(t_n - \tau_0 - a_\tau t_m)e^{-j2\pi f_c \tau_0} e^{-j2\pi f_c a_\tau t_m} \tag{1}$$

Among them, $u(t)$ is the transmitting baseband signal, f_c is the carrier frequency, t_n means fast time, t_m means slow time, Then the reference signal and the echo signal are respectively Fourier transformed along the fast time dimension and then conjugate multiplied to obtain the frequency domain pulse compression result:

$$\begin{aligned} S(f, t_m) &= S_e(f, t_m)U*(f, t_m) \\ &= A|U(f)|^2 e^{-j2\pi(f_c+f)\tau_0} e^{-j2\pi(f_c+f)a_\tau t_m} \end{aligned} \tag{2}$$

U(f) means the Fourier transform of the transmitted baseband signal u(t), $S_e(f, t_m)$ means that the echo signal has a fast edge The Fourier transform of the time dimension, $U^*(f, t_m)$ represents the conjugate of u(t) along the fast-time Fourier change. It can be seen from the above formula that in the second exponential term, the distance Doppler frequency and slow time are coupled, so that the echo signal cannot be effectively accumulated along the same distance unit, and distance migration occurs.

In order to correct the change of Doppler frequency shift with frequency f, the envelope can be corrected by applying the keystone transform to the slow time scale transformation. Let t_n denote the new slow time

$$t_n = \frac{(f_c + f)t_m}{f_c} \tag{3}$$

Equation (2) can be expressed as

$$S(f, t_m) = A|U(f)|^2 e^{-j2\pi(f_c+f)\tau_0} e^{-j2\pi f_c a_\tau t_n} \tag{4}$$

At this time, the coupling between the range Doppler frequency and the slow time in the second exponential phase is eliminated, and the movement of the unit across the range is corrected. The result of the Keystone transform is transformed into the fast time-slow time domain by inverse Fourier transform in the fast time dimension to obtain

$$\begin{aligned} &\tilde{s}(t_f, t_m) = A'\tilde{u}(t_f - \tau_0)e^{-j2\pi f_c a_\tau t_m} \\ &A' = Ae^{-j2\pi f_c \tau_0} \end{aligned} \tag{5}$$

Then perform Fourier transform in the slow time dimension to realize the coherent integration of the signal to obtain the range-Doppler two-dimensional matrix $\chi(t_f, f_m)$, we can obtain target delay and Doppler information at the same time。

2.2 Filtering and Decimation in Slow Time Dimension

In the actual external source radar system, the actual Doppler frequency range of the target to be observed is smaller than the actual sampling rate of the slow time dimension, and the whole spectrum of the signal obtained by directly performing the FFT on the

slow time dimension, resulting in calculations redundancy. Therefore, we can decimate the slow time dimension by D times, because the frequency domain resolution of the radar is only related to the integration time. Therefore, the sampling rate of the slow time dimension can be reduced while the Doppler resolution is unchanged, and the distance migration compensation can be performed on the narrow frequency domain, which can effectively reduce the amount of calculation.

The FIR anti-aliasing filter is used for decimation. Suppose the FIR filter order is P, the filter coefficient is h(n), and the decimation factor is D. The pulse compression signal before filtering is $S(f, t_m)$, and its slow time dimension discretization form is $S(f, n)$, Then the signal after FIR filtering is expressed as

$$S_{fir}(f, n) = \sum_{k=0}^{P-1} S(n-k, f)h(k) \tag{6}$$

The signal after D times decimation is:

$$S_D(f, n_D) = Sfir(f, n_D D + P - 1) \tag{7}$$

Among them, n_D represents the sampling point of the slow time dimension after D times downsampling.

2.3 Range Migration Compensation Algorithm Based on Keystone Transform

Keystone Transformation Implementation Use Chirp-Z Transformation
Chirp-Z transform (CZT) can essentially be regarded as an improvement of the DFT + IFFT method. Through CZT, circular convolution can be used to replace linear convolution for calculation, which can reduce the amount of calculation. The Keystone transform based on CZT is expressed as

$$S_{er}(l, \frac{k}{\alpha}) = \mathrm{CZT}[S_{er}(l, m)] \tag{8}$$

$$S_{er}(l, m') = \frac{1}{M|\alpha|}\mathrm{IFFT}\left[S_{er}(l, \frac{k}{\alpha})\right] \tag{9}$$

3 Implementation of GPU Parallel Acceleration of Efficient Coherent Integration Algorithm

The GPU parallel acceleration implementation process of the efficient coherent integration algorithm is shown in Fig. 2.

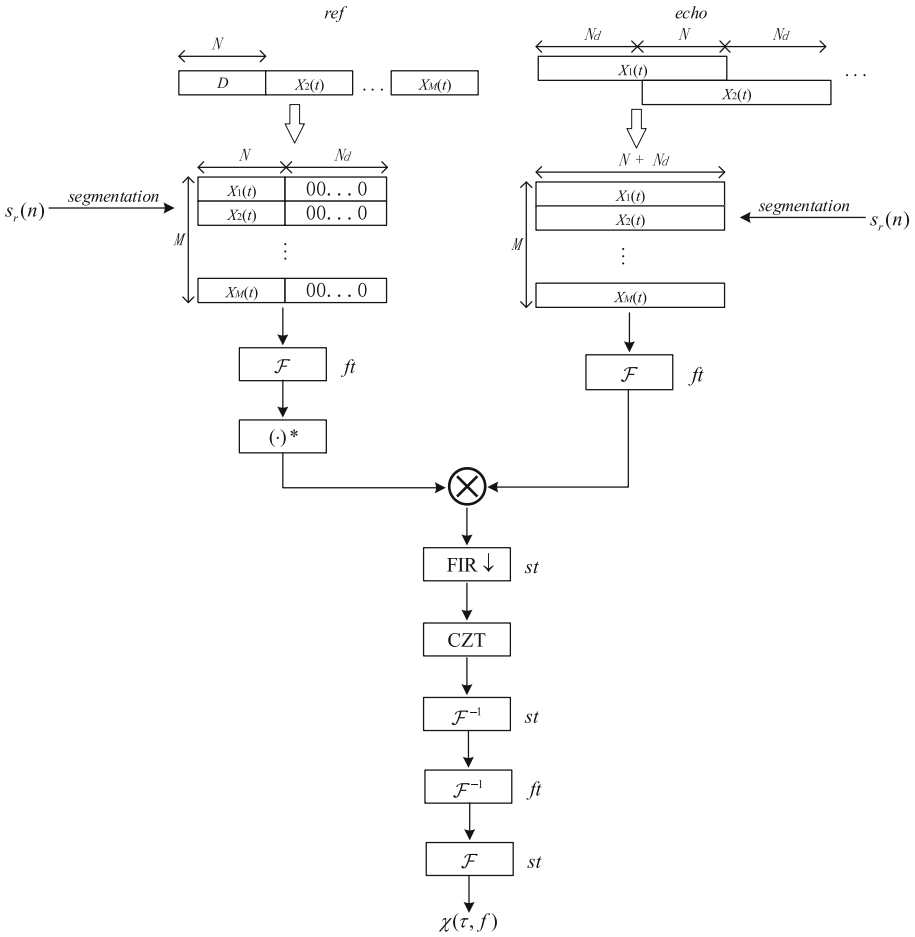

Fig. 2. The GPU parallel acceleration implementation process of the efficient coherent integration algorithm

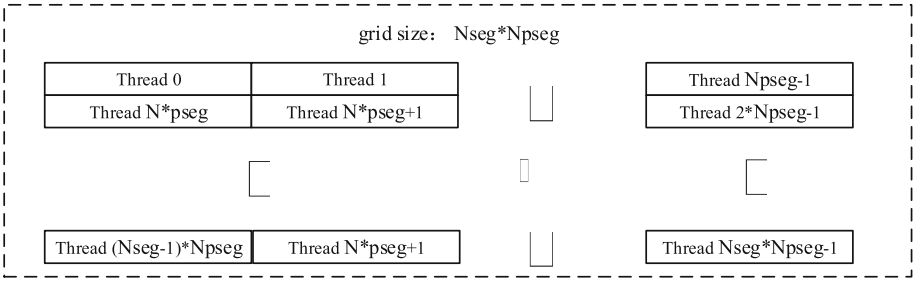

Fig. 3. The thread allocation of the kernel function

3.1 GPU Implementation Based on Fast Pulse Compression Algorithm

First, the reference signal needs to be overlapped and segmented. Using the characteristics of the GPU, it can be done through one kernel function. The thread allocation of the kernel function is shown in Fig. 3.

Among them, Nd is the fast time dimension corresponding to the maximum time delay of the signal, and Npseg is the number of sampling points of each segment of the signal, corresponding to the fast time dimension. Nseg is the number of segments, corresponding to the slow time dimension. GPU adopts two-dimensional thread block and two-dimensional thread indexing method, so that the x-direction index range of the allocated threads is 0~ Npseg-1, the y-direction index range is 0~Nseg, and each thread is related to S_r(n) and S_e(n) correspond one to one. The segment matrix of S_r(n) and S_e(n) can be calculated by the index value. The algorithm implementation flow chart is shown in Fig. 4.

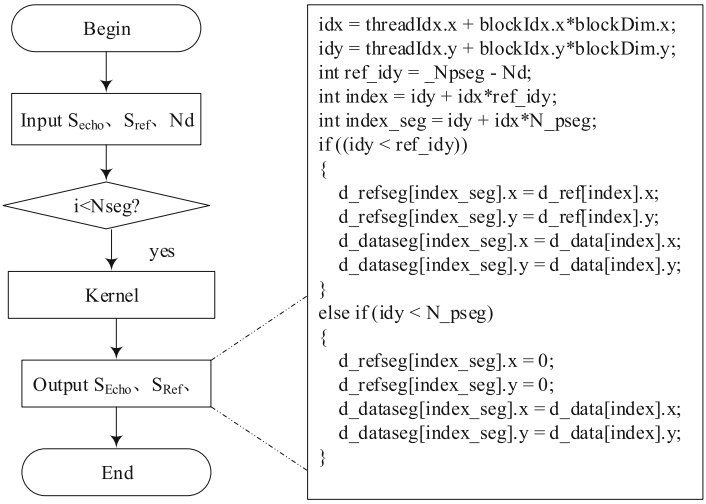

Fig. 4. Flow chart of the algorithm implementation

Then the segmented signals S_{Echo} and S_{Ref} are pulse compressed. The FFT part uses the FFT function in the cuFFT library to realize the parallel calculation of fast-time dimension FFT, and finally realizes the conjugate multiplication through the kernel function to complete the segmented pulse compression.

3.2 GPU Implementation of Decimation and CZT Transformation Based on Low-Pass Filter

Filtering and decimation aim to decimate the slow-time dimension of the pulse-compressed signal through FIR filtering, and then perform CZT transformation on the slow-time dimension. Because in CZT transformation, the slow-time dimension needs

to use a large number of rotation factors. When implemented in a GPU, the filter coefficients and CZT rotation factors can be stored in the GPU memory in advance to increase the computational efficiency. Similarly, the complex multiplication part is realized by the kernel function, and the final FFT and IFFT are realized by the cuFFT library.

4 Performance Valuation

The Parameters of signal used in the experiment is shown in the Table 1.

Table 1. Parameters of signal used in the experiment.

Carrier number	Carrier frequency	Bandwidth	Sample rate
1	910 MHz	7.56 MHz	10.08 MHz

The CPU software environment of the experiment is MATLAB R2019a, and the GPU software environment is visual studio 2015. The CPU and GPU model parameters and data processing parameters used in the experiment are listed in Table 2 and Table 3.

Table 2. CPU and GPU model parameters used in the experiment.

Parameters model	Number of cores	Clock frequency	RAM
Core(TM) i7-9700K	8	3.60 GHz	16 GB
GeForce RTX 2070	2304	1.62 GHz	8 GB

Table 3. Data processing parameters used in the experiment

Sample rate	Slow time dimension	Fast time dimension	Decimation	Delay
10.08 MHz	100	10000	2 s	14

The slow time dimension is designed to be 100 and the fast time dimension is 10000 according to the actual target motion. The GPU acceleration is implemented according to the method described above, and the GPU time-consuming analysis is shown in the following Fig. 5.

Through the processing of multiple pieces of data, the average time of the efficient coherent integration algorithm implemented on the CPU and GPU is 1.83 s and 0.013 s, respectively.

Analysis | GPU Details (Summary)

Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
void dpRadix0243C::kernel1Mem<unsigned int, double, fftDirection_t=-1, unsigned int=8, unsigned int=2, CO...	2	2.98004 ms	88	15552	0
void dpRadix0125C::kernel3Mem<unsigned int, double, fftDirection_t=-1, unsigned int=8, unsigned int=2, CO...	3	1.72873 ms	62	8000	0
void dpRadix0081B::kernel1Mem<unsigned int, double, fftDirection_t=-1, unsigned int=8, unsigned int=2, CO...	1	1.0731 ms	94	5184	0
void dpRadix0025B::kernel1Mem<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned int=3, C...	7	1.04865 ms	64	6400	0
__nv_static_41__28_bluestein_compute_75_cpp1_ii_d293d249__Z14bluestein_initId7ComplexIdEiEvT1_S2_S2_T_S3_...	4	939.55 µs	44	0	0
void dpRadix0025B::kernel1MemBluestein<unsigned int, double, fftDirection_t=1, unsigned int=32, unsigned i...	2	919.903 µs	80	6400	0
void dpRadix0025B::kernel1MemBluestein<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned ...	2	864.208 µs	64	6400	0
void dpRadix0008A::kernel1Mem<unsigned int, double, fftDirection_t=-1, unsigned int=128, unsigned int=4, C...	3	848.315 µs	72	0	0
void dpRadix0025B::kernel1Mem<unsigned int, double, fftDirection_t=1, unsigned int=32, unsigned int=3, CO...	2	809.073 µs	64	6400	0
void regular_fft<unsigned int=49, unsigned int=7, unsigned int=16, padding_t=1, twiddle_t=0, loadstore_modi...	3	528.715 µs	80	0	6272
void regular_fft<unsigned int=128, unsigned int=8, unsigned int=16, padding_t=1, twiddle_t=0, loadstore_mo...	2	426.248 µs	62	0	16384
void dpRadix0032B::kernel3MemBluestein<unsigned int, double, fftDirection_t=1, unsigned int=32, unsigned i...	2	392.873 µs	74	8448	0
void dpRadix0032B::kernel3MemBluestein<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned ...	2	338.074 µs	80	8448	0
fft_shift(double2*, double2*, int, int, int)	1	248.219 µs	18	0	0
KT_segment(double2*, double2*, int, int, int, double2*, double2*)	1	231.388 µs	18	0	0
data_filter_mulFFT(double2*, int, double2*, int)	1	179.613 µs	18	0	0
fft_shift_fy(double2*, int, int, int)	1	154.269 µs	16	0	0
conj_Complex_mul(double2*, int, double2*, int, int)	3	118.973 µs	18	0	0
void scal_kernel_val<double2, double>(cublasScalParamsVal<double2, double>)	1	103.87 µs	16	0	0
void regular_fft<unsigned int=3, unsigned int=3, unsigned int=256, padding_t=1, twiddle_t=0, loadstore_modi...	3	98.793 µs	40	0	0
norm_decimate(double2*, int, int, int, int, double2*)	1	97.79 µs	16	0	0
Complex_mul_fy(double2*, int, double2*, int, int, double2*)	1	76.927 µs	18	0	0
void dpRadix0032B::kernel1MemBluestein<unsigned int, double, fftDirection_t=1, unsigned int=32, unsigned i...	2	16.992 µs	85	8192	0
void dpRadix0032B::kernel1MemBluestein<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned ...	2	13.983 µs	82	8192	0
void dpRadix0032B::kernel1Mem<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned int=4, C...	1	12.8 µs	80	8192	0
void dpRadix0032B::kernel3Mem<unsigned int, double, fftDirection_t=-1, unsigned int=32, unsigned int=4, C...	1	8.256 µs	80	8448	0

Fig. 5. GPU time-consuming analysis

5 Conclusion

Aiming at the poor real-time performance of traditional mutual ambiguity algorithm and long-term integration of distance migration, this paper studies the method of combining segmental frequency domain pulse compression and Keystone transform, and proposes an efficient coherent integration algorithm suitable for GPU parallel processing. By filtering and decimating the slow time, the calculation amount is further reduced, and the distance migration is corrected, which improves the detection ability of weak targets. The effectiveness of the method is verified by the measured data.

References

1. Zeng, Y., Zhang, R., Lim, T.J.: Wireless communications with unmanned aerial vehicles: opportunities and challenges. IEEE Commun. Mag. **54**(5), 36–42 (2016)
2. Das, S.R., Beldingroyer, E.M., Perkins, C.E.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC Editor (2003)
3. Moqimi, E., Najafi, A., Aajami, M.: An enhanced dynamic source routing algorithm for the mobile ad-hoc network using reinforcement learning under the COVID-19 conditions. J. Comput. Sci. 1477–1490 (2020)
4. Bhuvaneswari, R., Ramachandran, R.: Denial of service attack solution in OLSR based manet by varying number of fictitious nodes. Clust. Comput. **22**(5), 12689–12699 (2018). https://doi.org/10.1007/s10586-018-1723-0
5. Surhone, L.M., Tennoe, M.T., Henssonow, S.F.: Temporally-Ordered Routing Algorithm. Mobile Ad Hoc Networks (2010)
6. Usman, M., Oberafo, E.E., Abubakar, M.A., et al.: Review of interior gateway routing protocols. IEEE ICECCO, pp. 1–5 (2019)