Parallel Implementation of the Range-Doppler Radar Processing on a GPU Architecture

Guanghui Zhao[1], Yongfei Liu[1], Shuping Zhang[2], Fangfang Shen[1], Yaohai Lin[3], Guangming Shi[1]
1. Xidian Univeristy, Xi'an, China
2. Beijing Huahang Radio Measurement & Research Institute, Beijing, 100013
3 . Fujian Agriculture and Forestry University, Fuzhou, China
ghzhao@xidian.edu.cn

ABSTRACT

Graphic processing units (GPUs) is widely used to accelerate the processing speed of the radar detection procedure, including the range compression, coherent integration and constant false alarm rate. Specifically, detailed parallel design of the radar algorithm and the thread programming are shown. The experimental results show that, by engaging the parallel technology into the radar processing procedure, much high speedup ratio can be obtained. Furthermore, precise target detection can be guaranteed.

Index Terms—graphic processing units (GPUs), pulse compression, coherent integration, constant false alarm rate

1. INTRODUCTION

A typical radar system transmits a wideband pulse, such as linear chirp, coded pulse, etc., then the radar system correlates the received target echo with the same pulse in a matched filter form. Normally, to achieve a high resolution radar detection performance, large bandwidth is adopted in the current radar system, therefore, according to the classical Nyquist sampling theorem, the sampling rate should be at least two times of the bandwidth. Specifically, to achieve a range resolution with scale of millimeter, the requirement of bandwidth will reach several hundreds of megahertz (MHz). Thus, a traditional radar receiver is equipped with a high-rate analog-to-digital (A/D) converter followed by a series of digital processing, and in short, the larger bandwidth, more complicated and expensive cost are needed.

Till now, many existing radar system processors are designed with multi-core DSP processors structures. More recently, new commercial off-the-shelf architectures such as multi-core platforms have been adopted in the design of the radar system, and have demonstrated orders-of-magnitude performance increase. And among these new schemes, parallel computing has been rejuvenated with the explosion of multicore technologies. And graphic processing units (GPUs) is an outstanding technology. NVIDIA shows that, its Fermi series GPU products can provide more than 1TF LOPS computation ability for single precision applications [1].

As a matter of fact, GPU is so popular in recent years that it is widely adopted in parallel applications, such as genetic analysis, seismic analysis, supercomputer design and deep learning. Recently, Bisceglie show the methodology of using GPU for radar imaging application [2]. In [3], by dividing raw data in blocks and implementing the range-Doppler algorithm (RDA), a novel parallelism is developed. In [4], a novel processor based on multiple GPU devices is provided for SAR imaging. In [5], both the range compression and azimuth compression are done in different processing cores.

In this paper, the GPUs is adopted to accelerate the speed for a classical radar signal processing. Specifically, we test the speedup ration of three typical algorithms, including the range compression, coherent integration and constant false alarm rate. The experimental results testify the superior capability of the adoption of the GPUs.

2. TYPICAL RADAR PROCESSING

In the framework of the monopulse radar system, to detect a target in the range-Doppler (RD) plane, we need to compute the parameters of the target, including range, angle and velocity from the raw data. Normally, these parameters are obtained via individual one-dimensional processing, such as pulse compression, coherent integration and constant false alarm rate (CFAR). Take the chirp signal for example, if a chirp is adopted in the monopulse radar system, which can be expressed as

$$s(t) = rect(t/T)\exp\left(j2\pi(f_c t + \mu t^2/2)\right) \quad (1)$$

where $rect(\cdot)$ represents the envelope signal, t and T stand for the fast-time in range and the pulse repetition period, respectively. f_c and μ are the carrier-frequency and modulating rate, respectively. For a target from range R_0, its echo

This work is supported in part by the National Science Foundation of China under Grants 61372071, 61401333 and 61201289, in part by the Young Star Science and Technology Project in Shaanxi province 2015KJXX-18, in part by the AreoSpace T.T.&C. innovation Program 201514A.

978-1-5090-4152-7/16 $31.00 © 2016 IEEE
DOI 10.1109/ISPDC.2016.18

can be written as

$$x_{mr}(t) = \delta(t - mT_r - \tau) \otimes s(t) \tag{2}$$

where δ stands for the impulse function, τ is the time-delay of the target. After range compression, the target echo can be expressed as

$$x_{mr0}(t) = T \sin c\left[\pi B\left(t - \frac{2R_0}{c}\right)\right] rect\left[t - \frac{2R_0}{c} / 2T\right] \exp\left[-j2\pi\left(\frac{2v}{\lambda}\right) mT_r\right] \cdot \exp\left[-j\frac{4\pi}{\lambda}R_0\right] \cdot \exp\left(j2\pi f_0 t\right) \tag{3}$$

It can be found that the peak position varies with respect to different pulse periods.

Now, with target echo recorded from multiple periods, we can obtain the information of the target's velocity via the procedure of coherent integration. In short, the fast Fourier transform (FFT) can be adopted to achieve the Doppler frequency, and the corresponding equation can be written as

$$x_{mr0}(t) = T \sin c\left[\pi B\left(t - \frac{2R_0}{c}\right)\right] rect\left[t - \frac{2R_0}{c} / 2T\right] \sin c\left(f_m - \frac{2v}{\lambda}\right) \cdot \exp\left[-j\frac{4\pi}{\lambda}R_0\right] \cdot \exp\left(j2\pi f_0 t\right) \tag{4}$$

It can be found that, after the range compression and coherent integration, the target can be obtained from the RD plane, with its coordinate $(2R_0/c, 2v/\lambda)$. As the target is move during the coherent integration, we need to use the CFAR to detect the target, and record the target's position accordingly. Next, we will show the procedure of the CFAR.

The detection threshold is computed so that the radar receiver maintains a constant pre-determined probability of false alarm. Normally, the relationship between the threshold value V_T and the probability of false alarm P_{fa} can be expressed as

$$V_T = \sqrt{2\psi^2 \ln\left(1/P_{fa}\right)} \tag{5}$$

If the noise power ψ^2 is assumed to be constant, then a fixed threshold can satisfy the above equation. However, due to many reasons this condition is rarely true. Thus, in order to maintain a constant probability of false alarm the threshold value must be continuously updated based on the estimates of the noise variance. The process of continuously changing the threshold value to maintain a constant probability of false alarm is known as Constant False Alarm Rate (CFAR). Three different types of CFAR processors are primarily used. They are adaptive threshold CFAR, nonparametric CFAR, and nonlinear receiver techniques. Adaptive CFAR assumes that the interference distribution is known and approximates the unknown parameters associated with these distributions. Nonparametric CFAR processors tend to accommodate unknown interference distributions. Nonlinear techniques attempt to normalize the root mean square amplitude of the interference.

The CA-CFAR processor is shown in Fig.1. Cell averaging is performed on a series of range or Doppler bins. The echo return for each pulse is detected by a square law detector. In analog implementation these cells are obtained from a tapped delay line. The Cell Under Test (CUT) is the central cell. The immediate neighbors of the CUT are excluded from the averaging process due to possible spillover from the CUT. The output of M reference cell is averaged. The threshold value is obtained by multiplying the averaged estimate from all reference cells by a constant K0, which is used as a scaling parameter. A detection is declared in the CUT if

$$Y_1 \geq K_0 Z \tag{6}$$

Cell-averaging CFAR (CA-CFAR) assumes that the target of interest is in the CUT and all reference cells contain zero mean independent Gaussian noise of variance ψ^2. Therefore, the output of the reference cells, Z, represents a random variable with gamma probability density function with 2M degrees of freedom. In this case, the gamma pdf is

$$f(Z) = \frac{z^{M/2-1} \exp\left(-z/2\psi^2\right)}{2^{M/2}\psi^M \Gamma(M/2)} \tag{7}$$

The probability of false alarm corresponding to a fixed threshold was derived earlier. When CA-CFAR is implemented, then the probability of false alarm can be derived from the conditional false alarm probability, which is averaged over all possible values of the threshold in order to achieve an unconditional false alarm probability. The conditional probability of false alarm when $y=V_T$ can be written as

$$P_{fa}(V_T = y) = \exp\left(-y/2\psi^2\right) \tag{8}$$

It follows that the unconditional probability of false alarm is

$$P_{fa} = \int_0^\infty P_{fa}(V_T = y) f(y) dy \tag{9}$$

where $f(y)$ is the pdf of the threshold, which except for the constant K0 is the same as that defined in (7). Therefore,

$$f(y) = \frac{y^{M-1} \exp\left(-y/2K_0\psi^2\right)}{\left(2K_0\psi^2\right)^M \Gamma(M)} \tag{10}$$

substituting (10) and (8) into (9) yields

$$P_{fa} = \frac{1}{(1+K_0)^M} \tag{11}$$

Observing of (11) shows that the probability of false alarm is now independent of the noise power, which is the objective of CFAR processing.

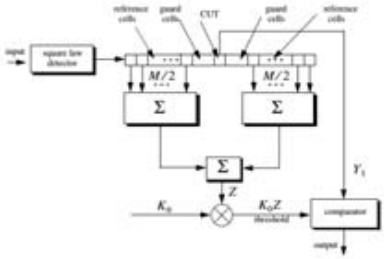

Fig.1 processing flow of CFAR

3. PARALLEL PROCESS WITH GPU ARCHITECTURE

In this section, we will discuss the utilization of GPU to speedup the process of range compression, coherent integration and CFAR.

A. Range Compression

According to the principle of the range compression, we split this algorithm to three parts: zero-padding, FFT, matrix -vector multiplication, IFFT. Here we use 3 kernel function and the CUFFT library function to realize the above process. The pseudocode is list below.

```
cufftHandle plan，planmany_Nbl  //build CUFFT handle
cufftPlan1d();                  //set handle
cufftPlanMany();                //set handle
for (int i=0;i<Nfft-Ntr+1;i++)
{
if (i == 0)
{
kernelhbbl();                   //zero-padding
cufftExecC2C();   //use CUFFT library function to realize the FFT process
kerneldotc();     //matrix-vector multiplication process
cufftExecC2C();   //use CUFFT library function to realize the IFFT process
kernelscale();    //rescale the magnitude of the results
}
else
{
if (i%N_tr == 0)
{
cudaMemcpy();     //remove the data
cudaMemset();     //clear the remainder data
}
}
}
```

B. Zero-padding

The processing of zero-padding is prepared for the following FFT transform, and the corresponding kernel is *kernelhbbl*(), and we need to split the RD plane into many grids, on one GPU, 45×4×32×32 threads are need, and after the processing, each matrix is assigned one value.

Now, we show the time cost of each step within the range compression in the GPU architecture.

Table I. Time cost of each step in range compression

Step	GPU/ms
Zero-padding	0.059392
FFT	0.161760
Matrix-vector multiplication	0.140320
IFFT	0.158720
Data scaling	0.047104

C. Coherent Integration

According to the theory of coherent integration, we split the whole procedure into the following procedure, i.e., smoothing while canceling and FFT process. It only need one kernel function and CUFFT library function, the pseudocode is list below.

```
cufftHandle planNtr；//build CUFFT handle
cufftPlanMany()；    //set handle
for (int i=0;i<Nfft-Ntr+1;i++)
{
huadongxiangxiao()； //smoothing and canceling
cufftExecC2C()；//use CUFFT library function to realize the FFT process
}
```

Similarly, we show the time cost of each step within the coherent integration in the GPU architecture.

Table II. Time cost of each step in coherent integration

Step	GPU/ms
Smoothing and canceling	0.059360
FFT	0.024576

D. CFAR

Here, we use the CA-CFAR to detect the target, the number of the reference cell is 26, the threshold is set to be 10.43, the number of the protectable cell is 8. During the coding in GPU, we split the whole procedure into two parts, i.e., transformation between the data format and detection with constant alarm rate. Two kernels are adopted to finish this target, and the psudocode is list below.

```
for (int i=0;i<Nfft-Ntr+1;i++)
{
abshuadongxx()；//transformation between the data format
kernelcfar ()；  //detection with constant alarm rate
}
```

Notes that the transformation between the data format is to transform the data format after coherent integration, since the radar signal is normally complex, we need to compute the magnitude of the real part and the image part, and the following computation of the square root. Then the complex data can be transformed as a real data. To facilitate the processing, a new function named *smoothing_kernel*() is defined and adopted, and correspondingly, 42×4×32×32 threads are opened for this process. The time cost of the CFAR is shown below.

Table III. Time cost of each step in CA-CFAR

Step	GPU/ms
CFAR (video memory)	0.841728
CFAR (share memory)	0.327648

4. EXPERIMENTS

In this section, we test the speedup of the utilization of the GPU in the radar target detection procedure. The version of the GPU is GV-N970G1 GAMING-4GD GTX970. The radar parameters are: the carrier-frequency and bandwidth are 240MHz and 0.5MHz, respectively, the pulse-width and pulse repetition period are 0.1ms and 1ms, respectively. The sampling frequency is 1MHz, 128 periods are used for coherent integration, the range and the velocity of the target are 115km and 600m/s, respectively.

Firstly, we do the process on MATLAB platform without the help of GPU. In Fig.2, we show the time cost of each algorithm by CPU, where the data scales for test are: 128×768, 128×1102, 128×1435, 128×1768 and 128×2102, respectively.

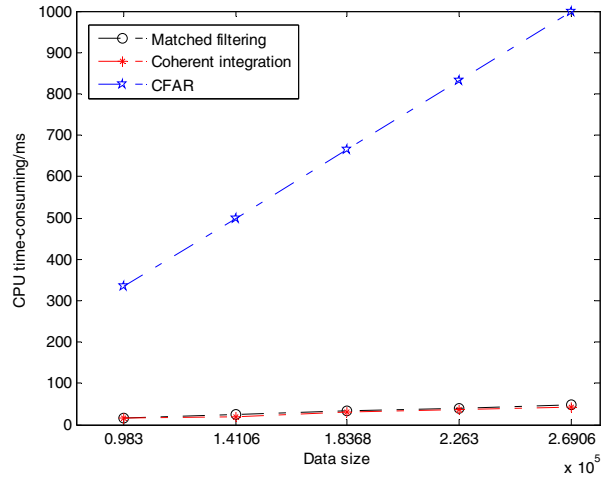

Fig.2 Time cost of the procedure by CPU

It can be found that, as the scale of the RD plane increases, the time cost of the whole procedure increases significantly. Next, we do the same procedure on a GPU architecture, and the time cost is shown in Fig.3.

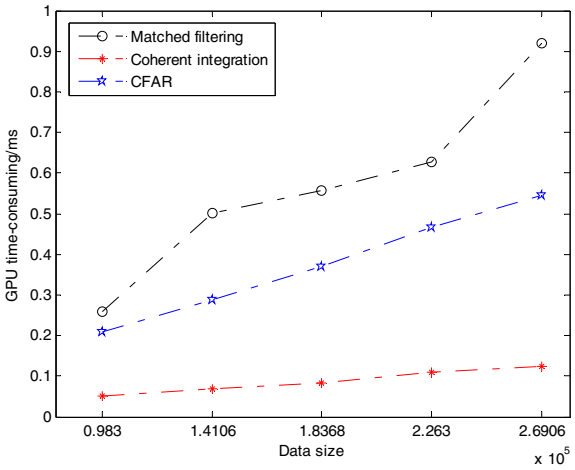

Fig.3 Time cost of the procedure by GPU

Compared with Fig.2, Fig.3 shows a significantly reduced time cost, and for the largest scale, the maximum time cost reduced 1000 times. To show the advantage of the GPU platform, the speedup ratio between CPU platform and GPU architecture is shown in Fig.4, where the largest ratio is up to 1800, which demonstrates the effectiveness and efficiency of the engaging GPU for radar processing.

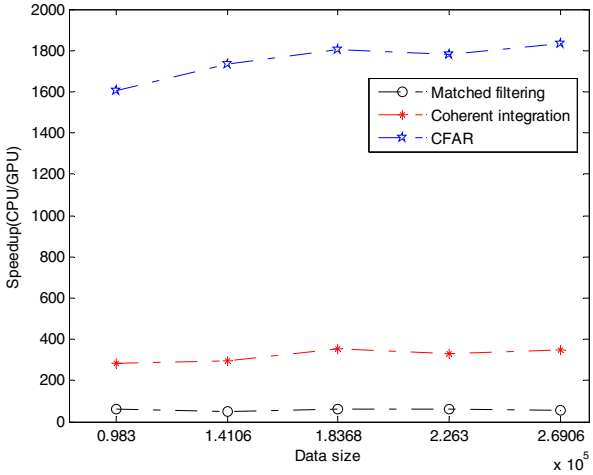

Fig.4 Speedup ratio between CPU platform and GPU architecture

5. CONCLUSION

In this paper, the GPU platform is investigated to accelerate the processing speed of the radar algorithm, the experimental results show the effectiveness and efficiency of the proposed approach.

6. REFERENCES

[1] X. Jang, Z. Minhui, W. Yirong, and P. Hailiang, "Parallel programming in SAR imaging processing," in Geoscience and Remote Sensing Symposium, 1999. Proceedings. IEEE 1999 International, 1999, pp.567-568.

[2] Maurizio di Bisceglie, Michele D i Santo, Carmela Galdi, Riccardo Lanari, Nadia Ranaldo, "Synthetic Aperture Radar Processing with GPGPU", IEEE Signal Processing Magazine, vol. 27, no. 2, pp.69-78, March 2010.

[3] R. Albrizio, G. Aloisio, A. Mazzone, and N. Veneziani," Multiprocessors architectures for SAR data processing," in Geoscience and Remote Sensing Symposium, 1991. Proceed- ings, 1991, pp. 267-270.

[4] Carmine Clemente, Maurizio di Bi sceglie, Michele Di San, Nadia Ranaldo, Marcello Spinelli, "Processing of Synthetic Aperture Radar Data with GPGPU," IEEE Workshop on Signal Processing Systems SiPS: Design and Implementation, p 309-314, 2009.

[5] D. B. Kirk and W. m. W. H. wu, Programming Massively Parallel Processors - A Hands-on Approach. Second Edition. Waltham, MA, USA: Morgan Kaufmann, 2013.