GPU-Based Parallel Algorithm for Wideband Signal Timing Recovery

Mohammad Hasan Shammakhi
Department of Electrical Engineering
Amirkabir University of Technology
mh.shammakhi@aut.ac.ir

Parnia Haji Faraji
Department of Electrical and Computer Engineering
University of British Columbia
parnia@ece.ubc.ca

Milad Mohammadi Bidhandi
Department of Electrical Engineering
Amirkabir University of Technology
mealadmmb@aut.ac.ir

Mohsen Hosseinzadeh
Department of Electrical Engineering
Amirkabir University of Technology
m.hosseinzadeh83@aut.ac.ir

Abstract
Symbol timing recovery is a complex calculation process that detects and corrects timing error in a coherent receiver. This paper presents a new implementation of GPU-based symbol timing recovery based on the parallel version of Gardner's method to minimize the timing error. Gardner's method utilizes a sequential process that relies on feedback error. The proposed method is a fast parallel implementation method on a GPU for time-error detection (TED) using the parallel timing recovery structure of sample signal blocks which makes fast error detection possible. We calculate the interpolation filter coefficients before timing recovery to detect the timing error of the symbols. We then compare the performance of timing recovery for different parallel techniques on different GPUs to minimize error and improve processing speed, up to 100 times, compared to Gardner's method. Performance evaluations show that we achieved a very high rate of timing recovery (50 Msymb/sec on GTX 1050 Ti) by optimizing the GPU implementation.

Keywords: Gardner's method, timing recovery, GPU, Cuda, synchronization, coherent receiver, digital communication

Introduction
In a telecommunication receiver, synchronization between transmitter and receiver is required in order to recover the correct symbols. In analog systems, the synchronization could be in a feed-forward or a feed-backward path by reproducing a time waveform from the input signal, so the phase and frequency of the local clock with the received signal will be synced [1]. When data is transmitted over a wireless channel, the synchronization is lost due to different types of noise, including phase noise, receiver thermal noise, fading and frequency offset. Therefore, a timing recovery subsystem is required to sample data at the right moment and detect its peak for correct simultaneous timing recovery (STR) [2]. One-time sampling at the receiver is ineffective due to the noise, such as Gaussian white noise. Therefore, the receiver noise could get diminished using an adaptive filter and increase the signal-to-noise ratio (SNR) of a sampled point (due to increased correlation). The goal is to get the best SNR while avoiding inter symbol interference. To maximize the SNR for detection, the demodulator must form an inner multiplication between the input signal and the reference signal [3]. This means that the locally generated reference signal must be synchronized with the received signal, then we need a proper timing recovery to extract the output symbol to compensate for the difference between the transmitter and receiver clock so that the symbol is extracted correctly. Because the transmitter's DAC and the receiver's ADC have different sampling times and they work independently of each other, there is a mismatch in sampling time that must be compensated for on the receiver side.
For decades, engineers have tried to design and implement intelligent receivers. With technology advancement, the timing recovery process was transferred to the digital domain, although the concepts in the analog and digital domains are the same [4]. The timing recovery process uses a delay-locked loop (DLL) which has three main components: Timing error detection process (TED), Loop filter (LF) for phase and frequency offset detection, and a controlled oscillator, such as a numerically controlled oscillator (NCO) in order to regulate the sampling time so that the peak of the input signal corresponds to the reference signal. There are several widely used methods for TED that will be discussed shortly, such as the Gardner method [5], Müller-Müller algorithm (M&M) [6], early-late algorithm (ELGA) [2], and Maximum probability (ML) - based on TED [7].
It is desired to apply TED with parallel tasks while maintaining the highest SNR and accuracy with the lowest possible sampling rate to speed up timing recovery by using data independence characteristics. Therefore, our method is focused on a Gardner-based TED [3]. ML method searches for correlation output peaks using the derivation of match filter (DMF). ELGA (early-late) is the former model of TED which finds the derivative by approximation using the early, prompt and late. This provides a relatively low-complexity structure for a high-performance system, which is crucial in terms of designing an efficient transceiver from a resource point of view. However, it has complex calculations. It needs three SPS and high-order filters. M&M requires only one SPS, but

the carrier recovery must be done before STR [8]. STR interpolation methods have been well discussed in the past. Due to the fact that the delay between the transmitter and the receiver clocks has a fractional part, the target timing recovery algorithm requires an Interpolation filter that could come in different types. The cubic spline method is one of the most efficient ones [9].

The Graphics processing unit (GPU) is a satisfactory computing resource in systems that are capable of simultaneous and parallel processing. Interpolation of symbols and the interpolation computation of the input data for input data are independent of each other. This fact makes it possible to use various forms of parallelization within the GPU to speed up the overall processing performance, which ultimately results in increasing the speed of overall error detection owing to obtaining timing recovery error directly from its output [3]. Also, symbols independence and their timing error computations independence reduce the detection time while the accuracy and efficiency (throughput) of the system remains intact.

To fit the repetitive and compatible nature ofDLLs, this paper presents a specific breakthrough in GPU programming. With the precise implementation and the availability of a large number of threads and cores in the GPU, it is also possible to perform STR on multiple input samples. Instead of sample-by-sample processing in traditional digital receivers, our methodsuggests block processing for several symbols simultaneously to further improve throughput, which is very desirable for modern wireless communication systems. The rest of the article is organized as follows:Section two discusses the details of a Gardner-based TED and presents an overview of the NVIDIA CUDA programming language. In section three the proposed design of TED on GPUs is explained. Section four describes the detailed design and implementation of our method, with a summary of our results. Finally, section five concludes the paper.

Background and Related Work

Gardner's Method

Gardner is an algorithm that detectstiming error for data streams. This algorithm includes interpolation, timing error detection and loop filter sections. The Gardner timing recovery algorithm requires two samples per symbol to be completed and it needs to know the previous symbol's timing in order to estimate the timing error of the current symbol.

Once the timing error e is computed, the Gardner timing adjustment algorithm is performed. If $e = 0$,no timing adjustment is required for the next symbol, If $e < 0$, a timing advance is required for the next symbol, and If $e > 0$, a timing delay is required for the next symbol.

The block diagram of Gardner's algorithm is shown inFigure 1. The Gardner algorithm is used in a closed control loop fashion in the component. This method is a non-data-aided feedback method that is independent of carrier phase recovery. It is used for baseband systems and modulated carrier systems. The Gardner method is similar to the early-late gate method but this method performs better in systems with high SNR values since it has lower self noise than the early-late method.

The Gardner algorithm is a sequential algorithm that depends on the error,due to timing recovery of previous symbols to extract each symbol,also, the loop filter used in the Gardner algorithm is of second-order which corrects the sampling error via the PI controller[1].

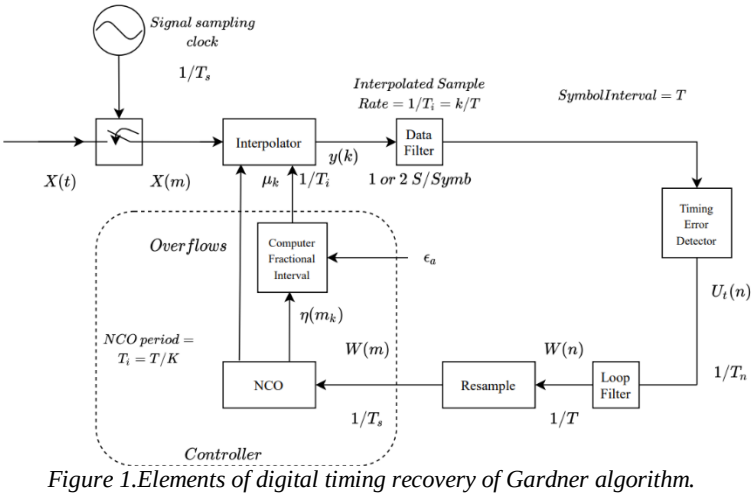

Figure 1.Elements of digital timing recovery of Gardner algorithm.

The computer fractional interval section is responsible for determining symbol delays relative to ADC output samples. AsFigure 2depicts, the symbol k is at the point kT_j. The closest sampled points are $\mu_k T_S$,$(\mu_k - 1)T_s$,$(\mu_k + 1)T_s$ and $(\mu_k + 2)T_s$. Using cubic interpolation, the value of the symbol k is estimated based on these four samples. The timing recovery error is then calculated based on the Gardner criterion. The error is calculated as (*1*)[2].

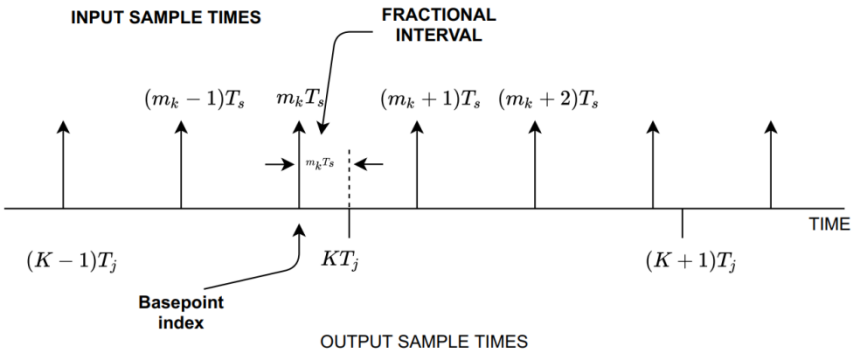

Figure 2.Sample time relations[3]

$$E(r) = x_d^2(rT+\tau) = x^2(\tau+rT) + x^2(\tau+(r-1/2)T) - 2x(\tau+rT)x(\tau+\left(r-\frac{1}{2}\right)T)$$
$$u_t(r) = x(\tau+(r-1/2)T)\{x(\tau+rT) - x(\tau+(r-1)T)\} \quad (1)$$

After calculating the error, the jitter value is calculated in terms of ted and $\int ted$. The value of μ_{new} based on jitter is then calculated as (*2*)[4].

$$ted = real\left(conj\left(x\left(r-\frac{1}{2}\right)\right)\{x(r)-x(r-1)\}\right)$$
$$jitter = a*ted + b*\int ted; \quad (2)$$
$$\mu_k = int[k\frac{T_i}{T_s}]$$
$$\mu_k = k\frac{T_i}{T_s} - m_k$$

Following this method, the position of the symbol is obtained[5].

CUDA

In November 2006, NVIDIA® CUDA® (CUDA or Compute Unified Device Architecture) introduced a parallel computing platform and general programming model that is available for software developers as a variety of functions in programming languages. It uses the parallel computing engine in NVIDIA GPUs to solve many complex computing problems in a more efficient way than CPUs in parallel processing.[6]

GPUs have been used in the past for computer graphics, but with the development of GPU technology in recent years, they are used for a wider range of applications, especially those involving large array processing and data matrices. Modern GPU platforms consist of one or more processor cores and one or more GPUs with powerful arithmetic resources that can run a large number of threads (computationally) at the same time.

The NVIDIA GTX 1050 TI hardware used in ourtests has 216 processor cores, providing a total of more than 165,000 active threads. GPUs process active threads simultaneously, and to increase the efficiency of such simultaneous execution, several threads can be allocated separately and perform various calculations until fully executed. To optimize our design, not only we use the memory hierarchy in GPUs, especially using register, shared memory (SM) and constant memory (CM), but also we utilized multiple existing processors (MP).[7] To make effective use of the GPU platform, the implementation should be in a way that the GPU threads are kept as busy as possible. This means that independent parallel execution opportunities must be identified and distributed in the GPU for efficient use of resources. In our platform, data transfer between the CPU and the GPU is done via the PCI Express bus.Although this kind of data transferring reaches high-speed rates, the power consumption is too large. Therefore, prior to transferring the results to the CPU, such transmissions must be diminished by maximum processing on the GPU.[6]

Proposed Method

Our design is based on the Gardner method shown in Figure 1. The main problem with Gardner's algorithm is its sequential nature. This characteristic limits the number of parallel paths that can be used to implement the algorithm onGPU and FPGA hardware.Even though, the modern CPUs which have the ability to perform multiple operations

simultaneously, due to the fact that in the Gardner algorithm the current data which is being processed is dependent on the former data, There is need to perform operations sequentially. The time-consuming interpolation operation to reach the target points in the Gardner algorithm increases the extraction time of each symbol. Our first step is to separate the interpolation operations as much as possible to increase the timing recovery processing rate. This technique doubles the system rate on our target hardware.

As Gardner's article points out, extracting symbols from samples requires incorrect interpolation of data. The cubic spline is the most efficient algorithm in terms of memory required to calculate and interpolate the desired points. As shown in Figure 2the neighbor samples of the output symbols are used to make the calculation more accurate. To calculate the signal value at the point $m_k T_s + \mu_k T_s$,(*3*)is used.The signal value at this point is obtained by replacing x with μ_k.

The right part of the (*3*)is derived using the errors obtainedof the previous data.The prediction of candidate signal samples as s_0 .s_1, s_2, s_3 is impossible. Due to the fact thats_0 to s_3 are four consecutive samples from the set of sampled points of the signal, these values can be calculated in advance for all s1 data parallel and in a short time, instead of obtaining them sequentially and calculating the coefficients in the M matrix,. In this way, calculating the desired points has much less computational complexity.

$$s(x) = [1 \quad x^1 \quad x^2 \quad x^3] \times \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ -1/3 & -1/2 & 1 & 0 \\ 1/2 & -1 & 1/2 & 0 \\ -1/6 & -1/2 & -1/2 & 1/6 \end{bmatrix}}_{Coef\_matrix} \times \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \qquad (3)$$

Therefore, we divide the timing recovery operation into two parts after applying the match filter. The first part is related to cubic interpolator calculations while the second one uses the timing recovery and timing error calculations according to equation(4) in order to calculate $C_0$to C_3.

$$\begin{aligned} C_0 &= S_1 \\ C_1 &= {}^{-1}/_3\, S_0 - {}^1/_2\, S_1 + S_2 - {}^1/_4\, S_3 \\ C_2 &= {}^1/_2\, S_0 - S_1 + {}^1/_2\, S_2 \\ C_3 &= {}^{-1}/_6\, S_0 - {}^1/_2\, S_1 - {}^1/_2\, S_2 + {}^1/_6\, S_3 \end{aligned} \qquad (4)$$

So, for every S_0 to S_3 that will be 4 consecutive instances of the signal, we will have the corresponding C_0 to C_3. Therefore, before retrieving the time and calculating the timing error, we calculate the values $C_0(n)$to $C_3(n)$ for all samples. The time required to perform this kernel on the GPU is very short and its speed will be about 1% of the total processing time. There will be two ways in order to calculate the values of $C_0(n)$ to $C_3(n)$. In the first method, each thread is responsible for calculating the real and imaginary parts of $C_0(n)$to $C_3(n)$Therefore, we need thread as much as the number of dataframes. This algorithm is performed as follows.

$$\begin{aligned} &while\ (n < data\ length) \\ &\quad C_0(n) = S(n-1) \\ &\quad C_1(n) = -\frac{1}{3}S(n) - \frac{1}{2}S(n-1) + S(n-2) - \frac{1}{6}S(n-3) \\ &\quad C_2(n) = \frac{1}{2}S(n) - S(n-1) + \frac{1}{2}S(n-2) \\ &\quad C_3(n) = -\frac{1}{6}S(n) + \frac{1}{2}S(n-1) - \frac{1}{2}S(n-2) + \frac{1}{6}S(n-3) \\ &\quad n = n + gridDim \times blockDim \end{aligned} \qquad (5)$$

In the second method, each thread is responsible for calculating the real and imaginary parts of one of $C_0(n)$ to$C_3(n)$. Then, the number of threads needed for this method is four times as much as the signal length.

Each one of the proposed methods may give us a better result depending on the number of different cores, hardware and frame length. In the third method, each multiplication is done by one single thread.

The next steps after the preprocessing are symbol extraction and the timing recovery error calculation. As stated in(2), The time recovery error is calculated as follows.

$$
\begin{aligned}
&val0 = C_0 + mu_{frac} \times (C_1 + \mu_{frac} \times (C_2 + \mu_{frac} \times C_3)\\
&ted = real(conj(val1) \times val2 - val0)\\
&tr(out) = val1\\
&jitter = a \times ted + b \times tedacc\\
&tedacc = tedacc + ted\\
&\mu_{new} = \mu_{frac} + \frac{SPS}{2} + jitter\\
&\mu_{frac} = mod(\mu_{new}.1)\\
&val2 = val1\\
&val1 = val0\\
&i = i + floor(\mu)
\end{aligned}
\qquad (6)
$$

The goal is to parallelize the calculation of timing recovery error.To overcome this, the data is processed in groups so that the extraction ofn_par symbols would be simultaneous and their timingrecovery error is calculated. Then, based on the error obtained from the n_par symbol, the next symbols are determined. Assume that s is the position of the last extracted symbol. The last estimated value of μ is divided into two parts μ_{frac} and μ_{pre}, Which are fractional and decimal parts of μ respectively. To calculate the $\mu$of the next n_pre symbols, we have:

$$
\begin{aligned}
&for\ k = 1{:}\ 2 \times N\\
&\quad \mu\_new(k) = i + \mu_{frac} + k \times \mu_{pre}\\
&\quad \mu_{frac}\_new(k) = mod(\mu_{new}(k).1)\\
&\quad i(k) = floor(\mu_{new}(k))
\end{aligned}
\qquad (7)
$$

In equation(7), $i(k)$ is the index number varying from C_0 to C_4that will be used for interpolation. Then, $val\ (k)$ is calculated as follows:

$$val(k+1) = CubicInterpolation(S_k.\mu_{frac\ -new}(k)) \qquad (8)$$

As mentioned before, each n_par symbol of the signal is calculated simultaneously from one single reference point. Therefore, if the estimation error for the first symbol in a block is equal toe_1, then the estimation error for the second one will be accumulated, so we will have:

$$e_2 = 2e_1 \qquad (9)$$

Similarly, the error for the n_par symbol is:

$$e_{n\_par} = n\_par \times e_1 \qquad (10)$$

To calculate TED for each n_par symbol, TED will be updated. The timing error of the first symbol in the block will be $ted = e_1$, and the error for the second symbol will be $e_2 = 2 \times ted$and finally, the error for the last symbol will be $e_{n\_par} = n\_par \times ted$. So, TED is equal to equation (11):

$$\overline{ted} = \frac{\sum_{i=1}^{n\_par} \frac{e_i}{i}}{n_{par}} \qquad (11)$$

Therefore, to calculate the error values, we will have:

$$
\begin{aligned}
&ted\_new = \frac{1}{n\_par} \sum_{i=1}^{n\_par} \frac{e_i}{i}\\
&jitter = n\_par \times \alpha \times ted_{new} + b \times tedacc\\
&tedacc = tedacc + ted_{new}\\
&\mu_{frac} = \mu_{frac\ -new}(2 \times n\_par)
\end{aligned}
\qquad (12)
$$

These values will be used for the next block's calculation.

Design and Implementation

In order to compare the CPU and GPU performance, we implemented the model described first in MATLAB and then in C ++ and CUDA. To evaluate the test results, we generated different signals from BPSK, QPSK, 8PSK, 16QAM modulations with RRC pulse shaping and different SPS (4, 8, 16) with two different Roll_of_factor (0.25 and 0.35). Finally, the N_ISI had two differentvalues of 10 and 100. One million symbols are made from all these signals. The AWGN noise is added to the generated signal for different SNRs and then frequency offset is applied to generated signal. We also utilized several satellite signals such as DVBS and DVBS2 protocols to evaluate the results. They also inherently have phase and frequency offset. On the receiver side, we have used an RRC match filter with length $SPS \times N\_ISI + 1$, with the Roll_of_factor of 0.3.
The CPU used for the test is RYZEN 5 2500 3.4GHz (6 cores) and Core i7-10700k 3.8GHz and the GPU used in this work is NVIDIA GTX 1050, the technical specification is shown in Table1:

Table1: Technical specifications of NVIDIA GTX 1050 graphic card

CUDA driver version/run time version	11.1
Total amount of global memory	4096MBytes
CUDA cores	768
GPU max clock rate	1493MHz
Memory bus width	128-bit
Total amount of constant memory	65536 Bytes
Total amount of shared memory per block	49152 Bytes

The first kernel is run on the firststream and the process completion event of this kernel is connected to the second stream, which is responsible for the timing recovery of the symbols and timing error detection.

Parallel Timing Recovery

As stated in Equations (6)to (12) we must assign $2 \times n\_par$ threads to the kernel. For the timing recovery, a long stream of data needs to be processed, which means that the data must be framed and transferred to the kernel. We choose 10000 to 1000000 symbols -depending on sps for the desired GPU. In the designed processing kernel function, the value of i increases until there are not as many symbols as $\mu\_new = (2 \times n\_par)$ at the end of the frame. Therefore, as soonas the processing stops, the main kernel parameters such as μ, $tedacc$ and the last part of the frame are moved to the next frame so that processing can continue from the correct point while the last processing state is kept. The Matlab and CUDA code on the algorithm is presented in [8].

The performance of the algorithm is discussed from two different points of view. EVM (%), RMS, peak EVM (%), and Avg EVM (dB), peak EVM (dB) and avg MER are the parameters used to measure the recovery accuracy. The error vector magnitude or EVM is a measure used to quantify the performance of a digital radio transmitter or receiver. An error vector is a vector in the I-Q plane between the ideal constellation point and the point received by the receiver. The root mean square (RMS) average amplitude of the error vector, normalized to ideal signal amplitude reference. The modulation error ratio or MER is another measure used to quantify the performance of a digital radio transmitter or receiver in a communications system using digital modulation. The modulation error ratio is equal to the ratio of the root mean square (RMS) power (in Watts) of the reference vector to the power (in Watts) of the error. As shown inFigure 3, the number of symbols required to reach the knee of the ted diagram in convergence is another parameter that determines the locking time of the algorithm.

Table *1* shows the degree of accuracy and speed in convergence.

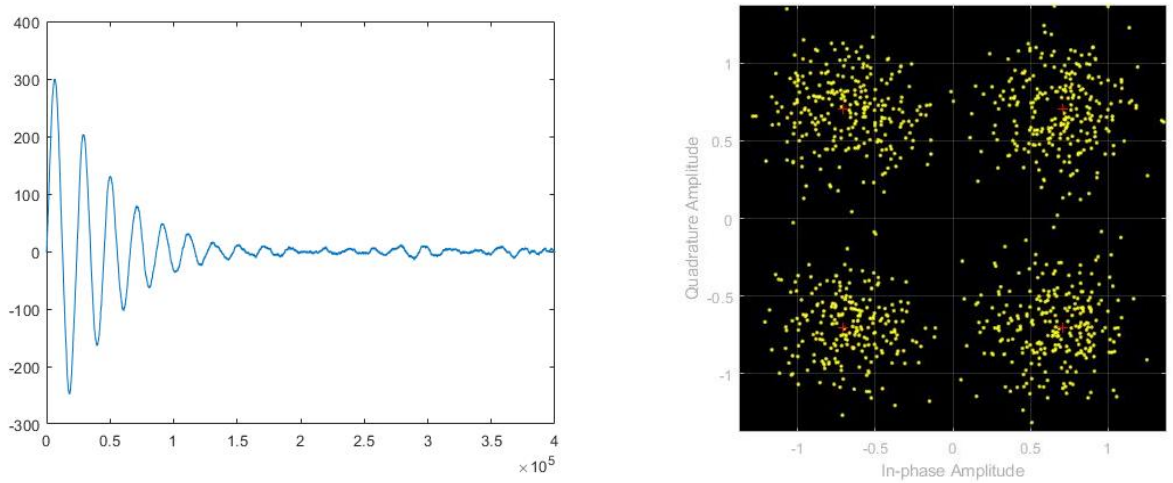

Figure 3:Convergence speed and constellation result of the proposed method

Table 1: Comparison of accuracy and convergence speed of our proposed method andClassic Gardner

Method	Classic Gardner	Parallel Gardner		
		N_Par =16	N_Par = 32	N_Par = 64
RMS EVM(%)	32.1	32.2	32.3	32.6
Peak EVM(%)	82.2	82.5	82.5	83.8
Avg EVM(db)	-9.5	-9.5	-9.5	-9.6
Peak EVM(db)	-1.7	-1.7	-1.7	-1.7
Avg MER(db)	9.5	9.5	9.5	9.6
N Symbol to Lock	$1.10 * 10^4$	$1.11 * 10^4$	$1.15 * 10^4$	$1.25 * 10^4$

The results of the implementation of the algorithm on the NVIDIA GTX 1050 for n_par16, 32,64 and the maximum processing rate are given in table 3.

	CPU(Ryzen 2600)		GPU		
Method	With MF and Interpolation	Without MF and Interpolation	N_Par=16	N_Par = 32	N_Par = 64
Max Symbol Rate Process (M symb/sec)	0.49	1.01	17.8	26.78	50.3

Table 2: maximum symbol rate processing capability using Gardner's algorithm and proposed method

As it is shown in table 3, while n_par increases, algorithm's performance speeds up. Also, the RMS_EVM value on the constellation goes up, which drops the accuracy of timing recovery .

Conclusion

In this paper, a novel method was introduced to increase the Gardner's timing recovery processing rate in a telecommunication receiver. The method is fitting for hardware with parallel processing capabilities such as FPGA and GPU. The purpose of this processing block is to recover the symbols sent on the transmitter side and to compensate for the error caused by the independence of the transmitter and receiver clock. This method has the ability to recover the correct transmitted symbol based on the sampled signal. It also resolves the timing error in symbols calculation over time. Our first innovation is the use of parallelization techniques in calculating cubic interpolation coefficients based on input data to calculate the signal at the point. We have also provided a method for calculating the value of ted in signals. The combination of these two methods has enabled us to process wideband signals. Our proposed method, considering 64 parallel paths in the calculation, is able to process signals with a rate symbol equal to 50 MegaBytes per second using medium GPUs, which is 100 times higher than the processing rate of Gardner's method in the CPU.

Acknowledgments

This research was supported by SUNYAR Company. We thank our colleagues from Sunyar Company for assisting in this project by helping with contributions in this research.This research was developed with funding from the SUNYAR Company under grant SDR004-0001.

References

[1] B. E. a. F. H. C. Dick, "Architecture and simulation of timing synchronization circuits for the FPGA implementation of narrowband waveforms," in *SDR Technical Conference and Product Exposition*, 2006.

[2] F. M. G. a. R. A. H. L. Erup, "Interpolation in digital modems. II. implementation and performance," *IEEE Transactions on Communications*, vol. 41, no. 6, p. 998–1008, 1993.

[3] F. J. H. a. M. Rice, "Multirate digital filters for symbol timing synchronization in software defined radios," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 12, p. 2346–2357, 2001.

[4] F. H. a. M. R. C. Dick, "Synchronization in software radios-carrier and timing recovery using fpgas," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 2000.

[5] F. M. Gardner, Phase lock Techniques, third ed., WileyInterscience, 2005.

[6] Nvidia, *NVIDIA CUDA Compute Unified Device Architecture:Programmer Guide*, 2007.

[7] J. G. M. a. M. Cheng, Professional CUDA c programming, John Wiley & Sons., 2014.

[8] Mh.Shamakhi, "https://github.com/mhshammakhi/," 2020. [Online]. Available: https://github.com/mhshammakhi/SDR_GPU/tree/main/Gardner_Parallel/.

[9] B. Sklar, Digital Communications: Fundamentals and Applications, 2001.

[10] W. L. P. a. S. S. B. S. C. Kim, "GPU-BASED ACCELERATION OF SYMBOL TIMING RECOVERY," in *Design and Architectures for Signal and Image Processing*, 2012.

[11] K. M. a. M. Muller, "Timing recovery in digital synchronous data receivers," *IEEE Transactions*, vol. 24, no. 5, p. 516–531, May 1976.

[12] M. R. a. J. R. J. Vesma, "Comparison of efficient interpolation techniques for symbol timing recovery," in *IEEE Global Telecommunications Conference*, 1996.

[13] F. J. Harris, Multirate Signal Processing for Communication Systems, Prentice Hall, 2004.

[14] M. M. a. S. A. F. H. Meyr, Digital Communication Receivers, Synchronization, Channel Estimation, and Signal Processing, Wiley-Interscience,, 1997.

[15] F. Gardner, "A BPSK/QPSK timing-error detector for," *IEEE Transactions on Communications*, vol. 34, no. 5, p. 423–429, may 1986.

[16] M. Frerking, Digital Signal Processing In Communications Systems, Springer, 2010.

[17] U. M. a. A. N. D'Andrea, "Synchronization Techniques for Digital Receivers, Springer, 1997.," *Springer*, 1997.