Design of GPU Based Non-coherent Signal Tracking Module for Real-time GNSS SDR

Jong-Il Park[1,*], *Kwi Woo* Park[1], and *Chansik* Park[1]

[1]Dept. of Control and Robotics Eng. Chungbuk National University, South Korea

Abstract.
In this paper, we design and implement GPU-based non-coherent GPS signal tracking module for real-time SDR. When using CPU and GPU simultaneously, the signal tracking module should be designed considering the memory bottleneck between the two processors. The basic non-coherent module, which accumulates the 1msec correlation value 20 times, is changed to accumulate M(20/N) times of Nmsec units. From the experiments using real GPS signals, the computational performance of N=20 is improved by 80% compared to N=1. Therefore, the implemented SDR using notebook computer can tracks more channels simultaneously in the real time.

1 INTRODUCTION

In Global Navigation Satellite System(GNSS), Software Defined Radio(SDR) has replaced traditional receivers. Hard-wired analog components of traditional receivers are replaced with digital components in order to allow for greater flexibility. SDR allows a user to change the software to quickly modify how it operates, and better maintenance than traditional receivers [1-2]. Also, real-time GNSS SDR is possible as GPU and GPGPU performance is improved [1].

GNSS signal can be used anywhere in the world, however, GNSS signal is restricted from receiving if there are obstacles such as high buildings, wood and etc. [6]. To improve the performance of receiving GNSS signal, three methods are commonly adopted: 1) extended Kalman filter to signal tracking loop, 2) AGPS (Assisted GPS) and 3) non-coherent integration method for signal tracking and signal acquisition [3]. In this paper, the non-coherent integration method was used to improve the reception performance of GNSS SDR.

Non-coherent integration signal tracking module accumulates correlation values of 1msec GPS signals for integration time (T). As the integration time increases, the accuracy of measurements is increased, however, the loop filter in DLL (Delay Lock Loop) and PLL (Phase Lock Loop) should be redesigned to track the signal continuously. To extract GPS navigation bit, 20msec is maximum integration time. In GNSS SDR sudden changes in integration time makes tracking loop unstable, therefore, step by step changes are required. In this paper, 1, 2, 4, 5, 10, and 20msec integration times are used sequentially. After navigation bit extraction, the integration time is fixed to 20msec.

GPU-based non-coherent signal tracking module is implemented to support parallel processing with CUDA (Compute Unified Device Architecture). Computational efficiency of parallel processing using GPU is affected by 3 factors: 1) memory copy from CPU to GPU, 2) kernel call and operation time and 3) memory copy from GPU to CPU.

To compare the computational efficiency of the GPU-based signal tracking loop with 20msec integration time, the elapsed time of above 3 factors are measured. As main computational burden, memory bottleneck between CPU and GPU is analysed by changing the load balance. The 20msec IF data is divided into N(msec) x (20/N) to perform N parallel processing concurrently.

An experiment was conducted to compare the computational efficiency of GPU-based signal tracking loop by changing *N*.

2 Implementation of Non-coherent integration SDR using GPU

2.1 Characteristics of CUDA operations

The CUDA platform is a software layer that provides direct access to the GPU's virtual command set and parallel computation elements. The CUDA allow access to the memory and command sets of parallel computing elements of the GPU. If the calculations performed by the program are suitable for parallel processing operations, performance is improved by using GPU. Fig. 1 shows the parallel processing computation process of CUDA.

* whddlf915@naver.com

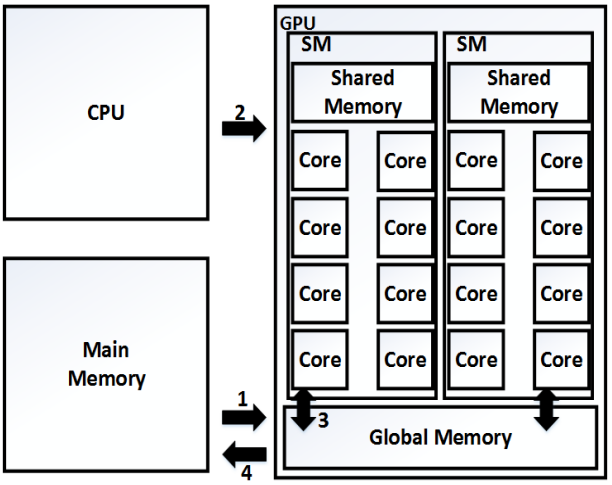

Fig. 1. Processing flow on CUDA

The CUDA processor operates in the order of the number shown in the arrow. First, data related to parallel computation is copied from main memory to global memory of GPU. Second, the CPU instructs the GPU process. Third, parallel operations are performed on the core of the GPU using data stored in the global memory. When parallel processing operations are performed, each Streaming Multi-processor(SM) in the GPU generates several blocks that perform parallel operations. Each block produces many threads that perform the operation in practice. The results are stored in the global memory after performing the procedure. Finally, copy the results stored in the global memory into the main memory.

2.2 Tracking structure of SDR

The SDR explained her is a real-time SDR based on GPU that is integrated into our lab Fig. 2. Shows the tracking loop behaviour of the implemented SDR. The white part operates on the CPU and the painted part operates on the GPU. The implemented SDR is a structure that tracks 1msec of IF signals. Copy received IF signal data to GPU, correlate with parallel computation, and store data back to CPU. Signal processing, such as PLL/DLL, bit extraction, and navigation, using correlation values from GPU. The tracking result is passed to the receiving part of the signal data for use in the next 1msec IF signal processing.

Fig. 3 shows the correlation part of Fig. 2. Correlator, which is mainly a repetitive operation, was implemented using CUDA. Correlator uses IF signal data and channel information stored in global memory of GPU. GPU generates blocks and threads, and blocks produce as many as the satellite number of successful signal acquisitions as IE, IP, IL, QE, QP and QL, which are correlated values of satellites. Each block has 1024 threads performing the operation, because the maximum number of threads that can be active on the GPU is 1024. The computation results of 1024 threads on each block are transferred to the shared memory, the shared memory is the memory shared within each block. Reason for storing computational results in shared memory is that data is sent to threads much faster

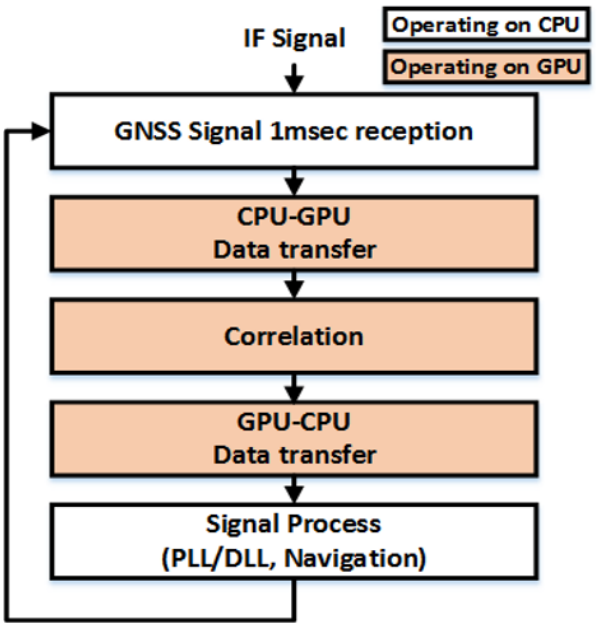

Fig. 2. Signal tracking processing

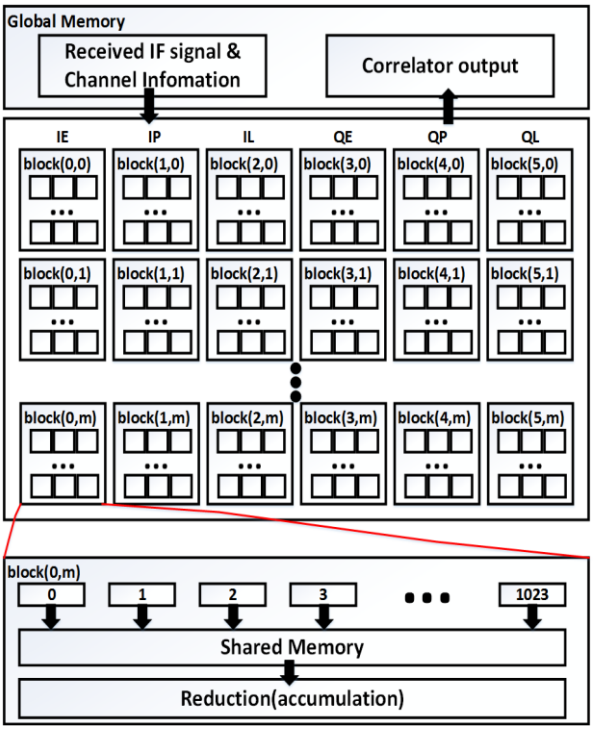

Fig. 3. Correlator implemented using CUDA

than global memory. After transfer to shared memory, add 1024 computational results to obtain correlation output. Add 1024 computational results by reduction method. The reduction method reduces the time complexity of $O(\log_2 n)$ that is faster than the existing $O(n)$. The correlation value obtained by adding 1024 computational results is sent to the global memory.

2.3 Non-coherent integration

Non-coherent integration is a method to integration correlation values to improve reception of receivers. The number set in integration time is an abbreviation of 20.

Because the GPS data transmission speed is 50 Bps and takes as much as 20msec per bit. When the integration time is *T*, *T* is defined as $T = \{1, 2, 4, 5, 10, 20\}$. If T is set to a number other than the abbreviation of 20, the data bit and the bit edge are unknown, so signal processing is not possible[3].

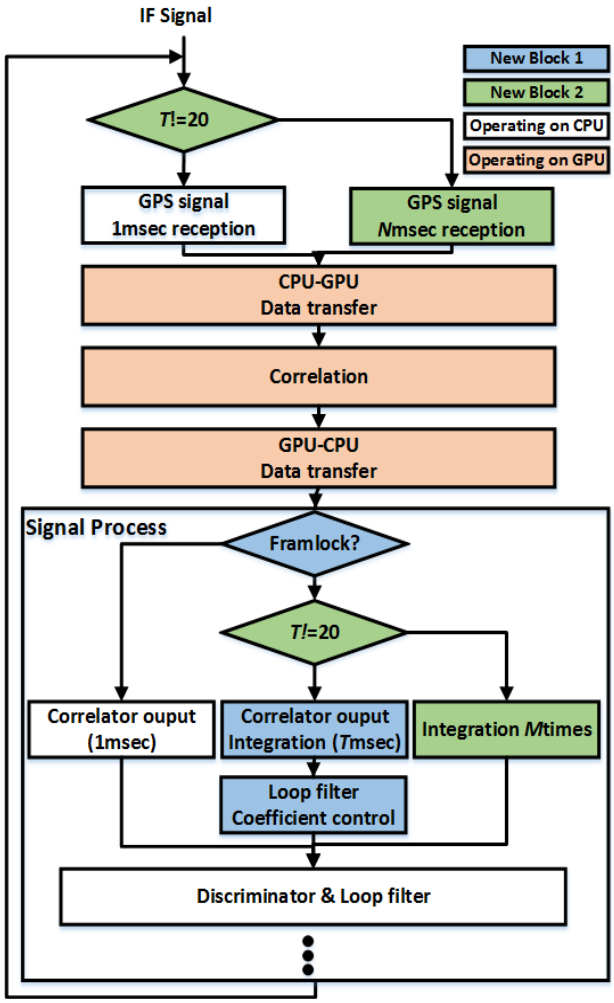

Fig. 4. Modified signal tracking processing

Fig. 4 shows the addition of non-correct integration using GPU in Fig. 2. New Block 1 is the part where non-coherent integration is added. Framelock of New Block 1 checks the start of the satellite signal's bit and whether the parity check of the satellite data have been completed. If the bit start point is ignored and integrated, the bit edge is unknown, so the data bit is unknown. If the Framelock is verified, integrate the output of the correlator by *T*msec. *T* increases to 20. Adjust the coefficient of the loop filter according to the *T* value. Depending on the *T*, the natural frequency and noise bandwidth of the loop filter change[2, 4].

2.4 Considering GPU operation non-coherent integration

Parallel processing operations are suitable for calculations where many data elements can run simultaneously in parallel. So we implemented correlator as CUDA. To use CUDA, data should be transferred from the CPU to the GPU. Number of times data is transferred from CPU to GPU should be reduced. It takes a lot of time to transfer data. Also, CUDA is most efficient in processing a lot of data with a lot of threads[5]. Therefore, to increase the efficiency of the CUDA, it is necessary to reduce the number of times data is transferred from the CPU to the GPU to the CPU and compute a lot of data at once.

Implement the structure shown in New Block 2 of Fig. 4 to compare the computational efficiency of the GPU. Comparison of the computational efficiency of GPU operates when *T*=20. When T=20, the length(*N*) of the IF signal data is changed. As the length of the IF signal data changes, the integration number(*M*) of non-corrections changes. *N*x*M* should be 20 to maintain the performance of the non-coherent integration *T*=20. As *N* changes, the number of times transferred from CPU to GPU and GPU to CPU changes, and the amount calculated at once by the GPU changes.

3 Experiment and Analysis

3.1 Experimental environment and purpose

Table 1 Laptop specification

OS	Windows 10 Home K 64-bit
CPU	Intel Core i7-7820HK
GPU	Geforce GTX 1080
Memory	DDR4-16GB
IDE	MS Visual Studio 2013, C++
CUDA version	Release 8.0

The experiment was conducted using the AORUS laptop. The specifications of the AORUS laptop are shown in Table 1. The specifications of the laptop, the OS is Windows 10, CPU is Intel Core i7-7820HK and GPU is GeForce GTX 1080. GPU has 2,560 cores, 1607MHz clock, 10Gbps memory clock, and 256bit memory bus. GPS signal data was obtained using an antenna installed on the roof of E10 of Chungbuk National University. The GPS signal's quantification bit is 16 bits and the sampling frequency is 25MHz.

Confirm the results of the two experiments using the signal received. The first experiment verifies the performance of the non-coherent integration method according to the *T* to verify feasibility of implementation. The performance of the non-coherent integration method is checked if the Doppler frequency, which is the output of the PLL, is properly tracked. The second experiment compares the computational time by adjusting the *N* and *M* of *N*x*M* in non-coherent Integration considering GPU operation. Compare the operation time of non-correct integration considering GPU operation and find the optimal *N*x*M*.

3.2 Validation of non-coherent integration implementation

Check the result of doppler frequency that is output when non-coherent integration is added to the Signal Process section of the signal tracking loop. Fig. 5 shows the Doppler frequency output as *T* changes. In the experiment, T was switched to 1, 2, 5 and 20, and as T increased, the variance of Doppler frequencies was reduced. Fig. 5 shows a continuous decrease in the Doppler frequency. This is because the satellite is moving. As a result, we added non-coherent integration to confirm that signal tracking performance is improved.

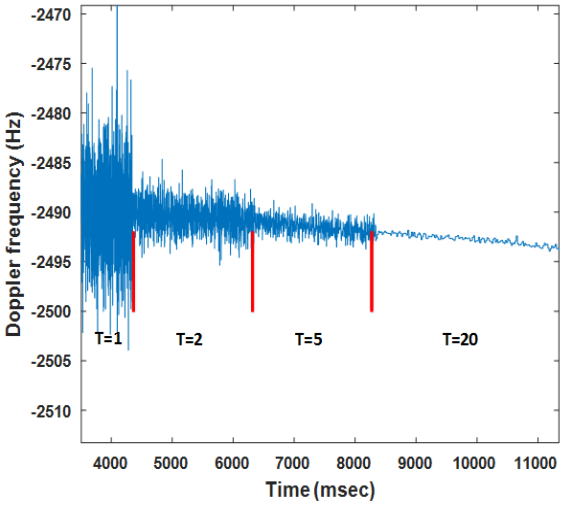

Fig. 5. GPS doppler frequency shift (GPS No. 12)

3.3 Comparison of computational time of non-coherent integration using GPU

The computational efficiency of GPU varies depending on number of data transfers and amount of data processing at a time. The operation time is checked by dividing three parts, the transfer time between CPU to GPU(A), kernel call and correlation operation time(B), and the transfer time GPU to GPU(C) in Table 2 and changing the *N*x*M*

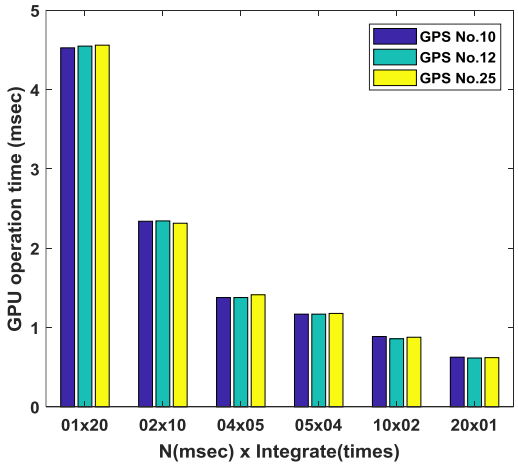

Fig. 6. CUDA operation time according to NxM

Fig. 6 shows the operating time of the CUDA for each satellite, changing the *N*x*M*, and Table 2 shows the operating time ratio for each part. Table 2 shows the ratio of operation time when *N*x*M* is 1x20. In all three parts, the operation time decreases as the size of N increasing. Compared to 1x20 and 20x1, the most time-reduced part is kernel call and operation time(B). It decreased by 90%. More than 80% of the operation time was reduced in all three parts. Fig 6. can be confirmed that the total operation time of GPU increases by N, which results in shorter processing time per 20 msec. The increasing the N in all three satellites, the shorter the time the GPU operates. When *N*x*M* is 20x1, its operation time is the shortest and its computational efficiency is the best.

Table 2. Operating time ratio in three parts

	1x20	**2x10**	**4x5**	**5x4**	**10x2**	**20x1**
A	1	0.52	0.32	0.27	0.21	0.17
B	1	0.51	0.30	0.25	0.18	0.11
C	1	0.50	0.31	0.32	0.21	0.14

4 Conclusion

Non-coherent integration of SDR using GPU was implemented to improve the computational efficiency of GPS signal tracking loops. Non-coherent integration reduced errors and improved tracking performance. Integration period of 20msec (*T*=20) gives the best tracking performance of the tracking loop. To evaluate the performance of GPU-based signal tracking module, the 1msec x 20times basic non-coherent module is changed to Nmsec x Mtimes module. By changing N, we compare the operation time of the three parts: 1) memory copy from CPU to GPU, 2) kernel call and operation time of GPU and 3) memory copy from GPU to CPU. In all three parts, the computation efficiency was improved as the size of N increased. From the experiments using real GPS signals, the computational performance of N=20 is improved by 80% compared to N=1. Therefore, the implemented SDR using notebook computer can tracks more channels simultaneously in the real time.

References

1. C.-H. Kang, S.-Y. Kim, J.-H. Yang, & C.-G. Park, Analysis of GPS signal acquisition algorithm for SDR, Institute of Control, Robotics and Systems(2011)
2. J. McGinthy, Captain, USAF, Global Navigation Satellite System Software Defined Radio, Air force Institute of Technology, Ohio(March 2010)
3. Nesreen I. Z., GNSS Receivers for Weak Signals, pp.28-37, (2006)
4. E. D. Kaplan, C. J. Hegarty, Understanding GPS/GNSS principles and applications, Third edition, Artech House, (2017)
5. NVIDIA, CUDA C Best Practices Guide, Design Guide, (October 2018)
6. K. H. Yoo, S. K. Sung, T. S. Kang, Y. J. Lee, E.-S. Lee, et al., Analysis of GPS+QZSS Availability at Uraban Canyon of Seoul , KSAS, (2008)