NAVSDR: A GPU-based Modular GPS Software Receiver

Liangchun Xu, *Wuhan University, China*
Nesreen I. Ziedan, *Zagazig University, Egypt*
Wenfei Guo, Xiaoji Niu, *Wuhan University, China*

BIOGRAPHIES

Liangchun Xu is a Ph.D. candidate at the GNSS Research Center, Wuhan University, China. He received his B.Eng. and M.Eng. degrees in Geomatics Engineering from Wuhan University, China, in 2012 and 2014, respectively. His research interests include GNSS receiver technology.

Nesreen I. Ziedan is the Acting Department Head and an Associate Professor at the Computer and Systems Engineering Department, Faculty of Engineering, Zagazig University, Egypt. She holds a Ph.D. degree in Electrical and Computer Engineering from Purdue University, U.S. She has several U.S. Patents in GPS receivers design and processing, and she is the author of a book entitled "GNSS Receivers for Weak Signals".

Wenfei Guo is a postdoctoral fellow at the GNSS Research Center, Wuhan University, China. He received his Ph.D. degree in Communication and Information System from Wuhan University in 2011. His research currently focuses on GNSS receivers.

Xiaoji Niu is a Professor at the GNSS Research Center, Wuhan University, China. He received the B.Eng. degree (with honors) in Mechanical and Electrical Engineering and the Ph.D. degree from Tsinghua University, Beijing, China, in 1997 and 2002, respectively. His research interests focus on integration navigation.

ABSTRACT

GPS software receivers are widely used in GPS-related algorithms' design and implementation. Two significant considerations involved in developing a GPS software receiver are the management of the newly introduced algorithms, and the execution performance of the baseband processing. A Graphics Processing Unit (GPU) can be used to accelerate the execution of a GPS software receiver. This paper introduces a novel GPS software receiver named "NAVSDR". NAVSDR adopts a modular design, which makes it scalable and reusable to accommodate new algorithms, such as GPU-based acquisition and tracking. The paper also proposes novel GPU-based acquisition and tracking architectures, which are independent of the GPU device, the signal type, and the integration length. The focus of the paper is on the following aspects: introducing the modular design of NAVSDR, demonstrating the detailed architectures of the proposed GPU-based acquisition and tracking, providing tests to evaluate the execution performance of the GPU-based acquisition and tracking, and describing how NAVSDR integrates the implementation of the proposed GPU-based acquisition and tracking properly and agilely with its modular design.

The performance evaluation tests indicate that the GPU-based acquisition and tracking can accelerate the execution of NAVSDR significantly, especially for the tracking process. With an NVIDIA GeForce GTX 750 Ti GPU and tested signals of 9.75 MHz sampling rate and 4-bit resolution, the GPU-based tracking in NAVSDR can achieve above 50 times performance gain at the peak compared with its CPU-based counterpart.

1. INTRODUCTION

GPS software receivers are widely applied in designing and testing new satellite navigation related algorithms. When building a GPS software receiver, a developer should not only consider the software function, but also the execution performance. The required high computational throughput in the signal acquisition and signal tracking processes prevents a standard GPS software receiver, which adopts Single Instruction Single Data (SISD) algorithms on a Central Processing Unit (CPU), from working in real time [1]. An efficient replacement for a CPU is a Graphics Processing Unit (GPU). A GPU is composed of massive parallel processors with high floating point performance and memory bandwidth. It can be used to accelerate both the acquisition and tracking processes in GPS software receivers. Accelerating the execution of GPS software receivers using General-Purpose computing on the GPU (GPGPU) has been carried out by many researchers [2-7]. A performance comparison between several GPU-based GPS SDRs can be found in [7]. This paper proposes novel GPU-based acquisition and tracking algorithms, which are independent of the GPU device, the signal type, and the integration length. Compared with existing GPU-based baseband algorithms, the proposed algorithms can achieve a competitive performance.

A significant consideration involved in building a GPS software receiver is how to maximize and maintain its flexibility and scalability in the face of the increase in the

number of the developed algorithms. A conventional GPS software receiver composed of one component will become bloated and unorganized after the addition of too many new functions. For example, the GPU-based acquisition and tracking algorithms perform the same functions as the already existing CPU-based versions in a GPS software receiver. Therefore, following the simplest practice of directly adding the code that implements the new GPU-based algorithms into the extant program will increase the size and complexity of the software. In fact, as more overlapping functions are introduced, the code of a conventional GPS software receiver could be rendered bloat, which could lead to a loose organization of the deployed software.

A modular design, which divides the software into several components according to different functions, can be adopted to address the aforementioned issue. The advantages of a modular design are as follows: 1) Independent components can integrate into different versions of GPS software receivers. A modular design can facilitate assembling the code, and it would only package the necessary components for the deployment of the software. This mechanism will encapsulate the final deployed software in a relatively small size. 2) A well-defined component provides the interface for the invocation of the external functions, while its implementation details are hidden. Therefore, multiple components of a GPS software receiver can be developed concurrently, without being interfered with each other. Moreover, when new components are integrated, a developer only needs to consider the coupling of the interfaces, instead of modifying the whole software. Therefore, a modular design will effectively increase the scalability of the software for future development.

This paper introduces a GPU-based modular GPS software receiver named "NAVSDR". NAVSDR is composed of seven pre-defined components according to the function division. The modular design enables NAVSDR to accommodate new algorithms, such as GPU-based acquisition and tracking properly and agilely. The proposed novel architectures for GPU-based acquisition and tracking are implemented in NAVSDR.

The remainder of this paper is organized as follows. First, the detailed modular design of NAVSDR is presented. Then, the GPGPU technology used to accelerate the execution of NAVSDR is introduced. Novel architectures for GPU-based acquisition and tracking are proposed and evaluated in the following two sections. Another section demonstrates how NAVSDR assembles the newly built GPU-based acquisition and tracking with its modular design. Finally, the conclusions are presented.

2. MODULAR DESIGN

The modular design includes both the component design and the interface design. The component design defines the expected function of each component, while the interface design defines the interface specifications between different components. A good modular design divides the program into loosely coupled, independent, and reusable components. The number of the defined interfaces is at the minimum level. While in each component, the essential functions are complete.

NAVSDR is composed of seven components: the preprocessing component, the correlation component, the acquisition component, the tracking component, the measurement extraction component, the Position, Velocity and Time (PVT) solution component, and the utility tools component. Each component is packaged into a dynamic-link library (DLL), and invoked by other components and the main application, which is the entrance of NAVSDR.

In NAVSDR, the aforementioned seven components are designed according to their different functions. Each component's function can be described as follows. The preprocessing component reads the pre-stored intermediate frequency (IF) sampling data into a data buffer residing in the CPU memory for further processing. The correlation component generates the local carrier and code replicas, and correlates them with the IF samples. The acquisition and the tracking components use the output of the correlation component to accomplish the signal acquisition and tracking functions. The measurement extraction component is responsible for extracting carrier phase and pseudorange measurements from the output of the tracking component. The PVT estimation component estimates the position, velocity, and time from the extracted measurements. Finally, the utility tools component provides the basic matrix operation functions, the time and coordinate transfer functions, and other common functions for NAVSDR.

The interface design of each component is based on the relationship between different components. In NAVSDR, the utility tools component is invoked by all the other components. The correlation component is invoked by the acquisition component and the tracking component. Moreover, the tracking component is the central component, which invokes the measurement extraction component. When more than four satellites' measurements are extracted, the measurement extraction component would invoke the PVT estimation component to generate the final position, velocity and time results. The preprocessing component, the acquisition component, the tracking component are independent from each other, and they hold the interfaces to integrate into a complete GPS software receiver. The modular design of NAVSDR is shown in Fig. 1. Each block in Fig. 1 represents a component of NAVSDR, in which the detailed functions of each component are listed. Moreover, the link lines between different blocks represent the invocations between different components in NAVSDR.

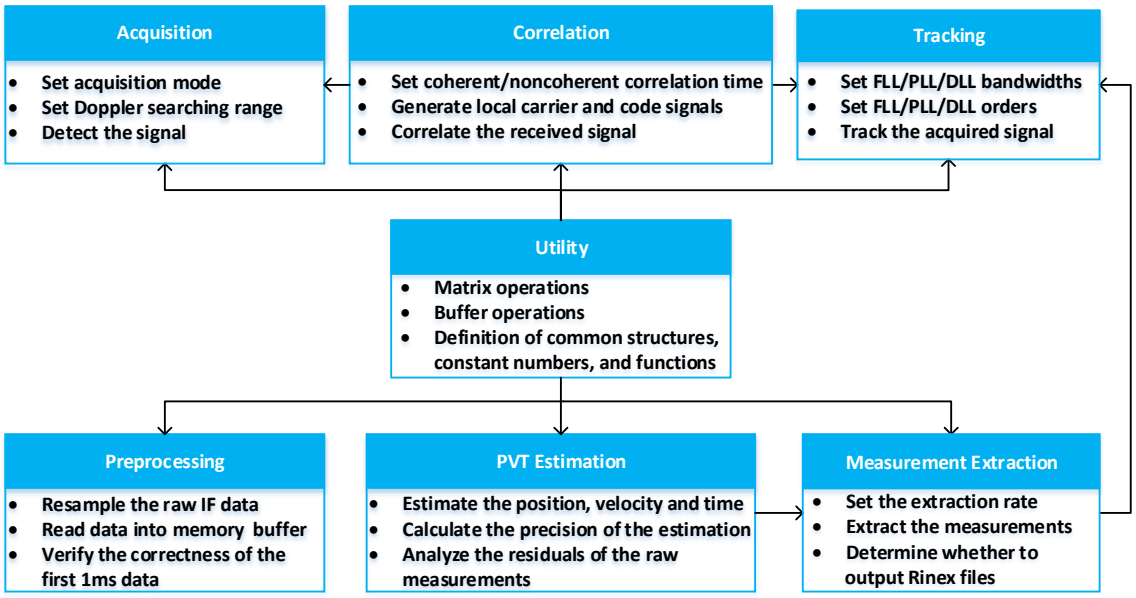

Fig. 1 Modular design of NAVSDR

3. GPGPU TECHNOLOGY

Compared with a CPU, a GPU is composed of massive cores with tremendous computational power and high memory bandwidth [8]. GPGPU provides an approach to accelerate scientific applications, including the correlation process in GPS software receivers.

CUDA is a parallel platform invented by NVIDIA for C/C++/Fortran programming on the GPU. CUDA C is employed in this paper to design and realize the GPU-based acquisition and tracking algorithms. In order to utilize the computational power of the multiprocessors in GPU, kernels (functions) that are executed by each individual CUDA thread should be defined as C functions using the keyword "__global__". Massive threads can execute on the GPU concurrently. Each 32 threads are coupled into a warp so that they can execute one common instruction with different data at a time. Branch divergence, which means executing different instructions in the threads of a warp, will significantly decrease efficiency, and thus it should be avoided. Warps are grouped into blocks, where the threads of a block are executed concurrently on one multi-threaded Streaming Multiprocessor (SM). Blocks are further grouped into a grid, where they are executed concurrently on multiple SMs in a GPU.

CUDA also provides different types of memories in the GPU to maximize the performance [9]. Global memory, shared memory, local memory, and registers are used in the proposed GPU-based architecture design. They are allocated for grid, block, and thread. The bandwidth of different kinds of GPU memories varies. The registers provide the fastest memory transfer with nearly 8TB/s bandwidth. The shared memory and global memory follow the registers in terms of the memory transfer speed. The local memory is an abstract memory type to hold spilled registers. Register spilling occurs when a block requires more registers on an SM than the available. GPUs with CUDA capability 2.0 or higher spill registers to L1 cache, which is physically integrated with the shared memory. The older GPUs spill registers directly to the global memory. Therefore, the local memory's bandwidth is dependent on the device. The slowest memory transfer is between the host memory and the device memory with about 8GB/s bandwidth. This bandwidth value is limited by the Peripheral Component Interconnect Express (PCIe) connector between the CPU memory and the GPU memory.

4. GPU-BASED ACQUISITION

Signal acquisition is a search process that determines the visible satellites and provides rough estimates of their code delays and Doppler shifts. A common and efficient signal acquisition technique is circular correlation using FFT. Circular correlation works as follows. The received signal is first stripped off the carrier, which is compensated by each possible Doppler shift. Following that, the received signals and local code replica signals are transformed into the frequency domain using FFT. The complex conjugate of the FFT of each local code signal is multiplied by each FFT of the Doppler compensated signal. The results of the multiplication process are transformed back to the time domain by applying IFFT, which generates the required correlation at all the possible code delays and Doppler shifts. The circular correlation process can be described by:

$$p(f_d, k, n) = IFFT\left(FFT(S_{IF}(n) \cdot R_{carr}(f_d, n)) \cdot FFT^*(C_p(k, n)) \right) \quad (1)$$

where $S_{IF}(n)$ is the received IF signal, $R_{carr}(f_d, n)$ is the local carrier replica signal, which is generated for a possible Doppler shift, f_d, and $C_p(k, n)$ is the local code replica signal, which is generated for a possible satellite, k.

The acquisition process is repeated for each possible satellite, which is time consuming. Executing the acquisition process on a GPU can lead to a huge reduction in the acquisition time. The acquisition process can be accelerated by the GPU using the cuFFT library in CUDA Toolkit. In the GPU-based circular correlation algorithm, local carrier replica signals with different Doppler shifts are generated and stored consecutively in an array, which resides in the GPU memory. The Doppler removal is processed in parallel using a kernel function for calculating two vectors' product. The advanced data layout feature of the cuFFT library is used in the algorithm design of the GPU-based acquisition. This feature can allow transforming a subset of the product array. The subset array used here contains the product of each local carrier replica signal, compensated with Doppler shift, and the received IF signal. The advanced data layout feature can be set by calling the function "cufftPlanMany". Similarly, the FFT process of the local code replica signals and the final IFFT calculation can also benefit from this batch process feature. Another kernel function is written to generate the local code replica signals and complete the code correlation after the FFT process. The parallel GPU-based acquisition can search the visible satellites, the Doppler shifts, and the code delays concurrently. As a comparison, the CPU-based circular correlation can only search the code delays or the Doppler shifts in a parallel way. A detailed description of the GPU-based circular correlation algorithm is shown in Fig. 2, where, T is the coherent integration length, N_f is the number of Doppler shifts, N_s is the number of samples in T ms, and N_c is the number of satellites signals being processed.

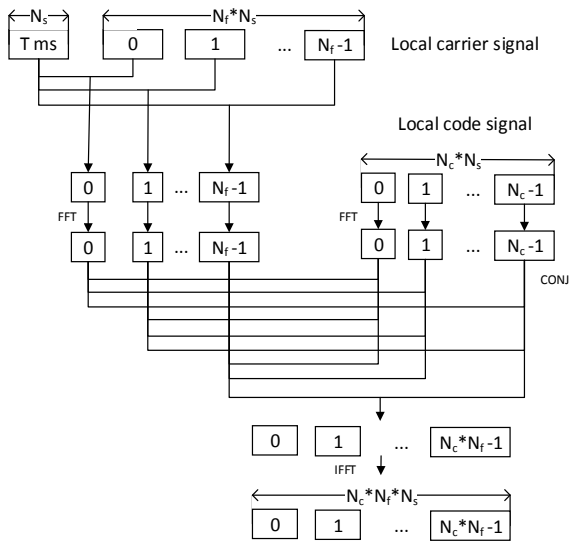

Fig. 2 The GPU-based acquisition using FFT

The final output array of the GPU-based acquisition contains the correlated results at every possible Doppler shift and code delay. The size of the array is equal to the multiplication of the number of possible visible satellites, the number of possible Doppler shifts, and the number of possible local code replica signals. Before continuing with the signal detection process, the output array, which has a huge size, is transferred from the GPU memory to the CPU memory in a coalescent way. The array should be divided into N_c equal parts in order to extract the acquisition results in a way similar to the one that is adopted in the CPU-based acquisition.

The aforementioned GPU-based acquisition design is an ideal solution for parallel acquisition process. However, the required GPU memory of the algorithm is very huge. Considering the maximum transform size of the cuFFT library is limited by the available GPU resources [10], the cuFFT library would sooner or later fail to allocate GPU resources for such a huge size of array, when the integration time is increased. This could prevent the application of IFFT in the last step of the proposed GPU-based acquisition. A compromise can resolve this issue by serially applying the final IFFT for N_c times. This practice is at the cost of degrading the parallel processing, but it successfully reduces the maximum required transform size from $N_c * N_f * N_s$ to $N_f * N_s$, which guarantees the proposed GPU based acquisition to process more than 10 ms long coherent integration on a GPU with medium calculation power. The implementation of the GPU-based acquisition in this paper adopts the aforementioned compromised algorithm. NAVSDR provides an acquisition view as shown in Fig. 3 to visualize the correlation results of each satellite in the acquisition process.

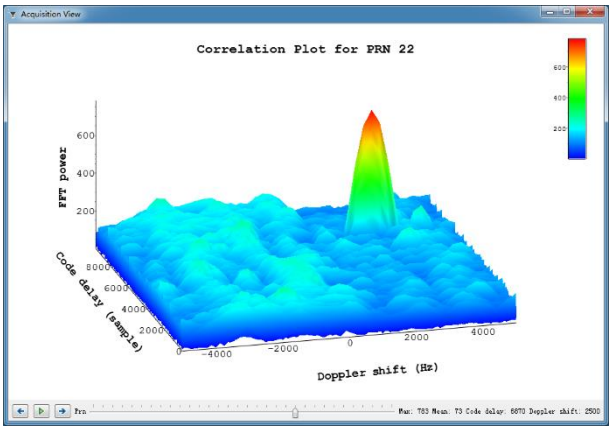

Fig. 3 The acquisition view of NAVSDR

In order to evaluate the execution performance of the proposed GPU-based acquisition algorithm, the cold start experiment with different coherent integration lengths, $T\ ms$, is conducted. The tested GPS signal is complex numbers with 19.5 MHz sampling rate (9.75 MHz for both in-phase and quadrature-phase samples), and 4-bit resolution. The Doppler shift range is [-5000 Hz, 5000 Hz], and the number of searched satellites is, $N_c = 32$. The coherent integration time length range is [1 ms, 10 ms] with an interval of 1 ms. In this test, with a Doppler range of $\pm 5000\ Hz$, the number of Doppler bins, N_f, with a separation of $1/T$, is calculated as follows.

$$N_f = \frac{5000 - (-5000)}{1/(0.001 * T)} + 1 = 10T + 1 \qquad (2)$$

Besides the complexity of the algorithm itself, the implementation's performance of the GPU-based acquisition is also influenced by the GPU device used. The chosen device here is an NVIDIA GeForce GTX 750 Ti GPU. Its technical specifications can be found in [11]. The execution time of the GPU-based acquisition with different coherent integration lengths is collected and plotted in Fig. 4. A serial implementation of the CPU-based acquisition using FFTW library provides a benchmark for evaluating the execution performance of the GPU-based acquisition. The CPU used here is a quad-core 3.6 GHz Intel Core i7-4790.

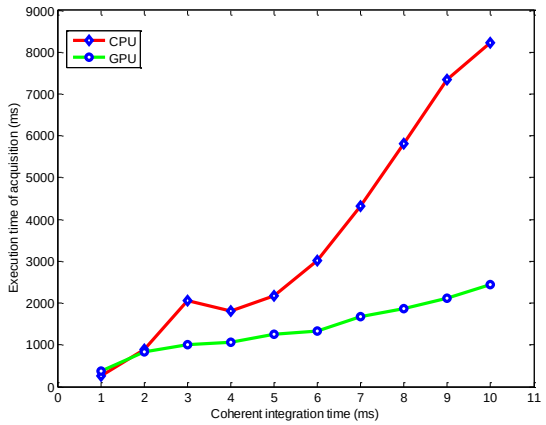

Fig. 4 The execution time of the CPU-based acquisition and the GPU-based acquisition

With the coherent integration time increased, the execution time of both the GPU-based and the CPU-based acquisition processes tends to increase in a long run. However, the execution time of the CPU-based acquisition decreases when the coherent length increases from 3 ms to 4 ms. This can be explained by the fact that the speed of the FFTW library in "MFLOPS" fluctuates with the input array size [12]. The execution time of the GPU-based acquisition approaches that of the CPU-based acquisition when the integration length is 1 ms or 2 ms. This is because a low computational load cannot exert the full power of the GPU. On the contrary, the overhead of the kernel functions and the memory transfer between the GPU memory and the CPU memory compose a large portion of the execution time, which would cancel out the speedup brought by the GPU-based computation. This phenomenon will also occur when evaluating the execution performance of the GPU-based tracking. With the integration time increased, the computational gain on the GPU becomes significant. When the integration time is 10 ms, the speedup brought by the GPU-based acquisition is about 3.3 times.

5. GPU-BASED TRACKING

Following signal acquisition, a tracking process is performed. The tracking process aims at estimating the code and carrier parameters with high accuracy, and providing a continuous tracking of the change in the received signal. The tracking process also requires a correlation process. However, the correlation is calculated at the current estimates of the code and carrier parameters, not at all the possible code delays and Doppler shifts, like in the acquisition process. The prompt correlated signal, $S_{IQ,p}$, can be found from

$$S_{IQ,p} = \sum_{n=0}^{N-1} S_{IF}(n) R^*_{carr}(n) C_p(n) \quad (3)$$

where $S_{IF}(n)$ is the received IF signal, $R_{carr}(n)$ is the local carrier replica signal, and $C_p(n)$ is the local prompt code replica signal. The early correlated signal, $S_{IQ,e}$ and the late correlated signal, $S_{IQ,l}$ used for code tracking loop can be also calculated in a similar way.

The tracking process can be parallelized on both the channel level and the sampling data level, where each channel is responsible for the correlation of one satellite. Considering the aforementioned CUDA programming model, each channel's correlation process can be mapped into one block, and each signal sample can be mapped into one thread. The integration after carrier and code wipeoff can be transformed to a block reduction, which is the process of calculating the sum of the variables in threads of a block. The algorithm of the GPU-based tracking is shown in Fig. 5, where N_c is the number of satellites signals being processed, and N_s is the number of samples in one correlation process. Since the assigned blocks per grid, $gridDim.x$, and threads per block, $blcokDim.x$, are limited numbers, they may be less than $N_c, and\ N_s$, respectively. There are two for loops assisting the blocks and threads to complete the whole computation. The proposed algorithm is actually applicable to all GNSS signals, since N_c and N_s are all variables in this algorithm. Therefore, the algorithm is independent of the correlation time length, the code type, and the GPU device, which makes the proposed GPU-based tracking architecture reusable and scalable.

The GPU-based tracking process needs six reductions in each block to generate the early, prompt, and late results for both in-phase and quadrature-phase received signals. There are two subset algorithms for the block reduction in the GPU-based tracking. 1) The block sum can be calculated using a similar tree-like reduction described in [13]. Shared memory of the GPU is used in this process. The aforementioned design and implementation approach is applicable on all CUDA-enabled GPUs. 2) The block sum can be also calculated using warp shuffle functions. For the GPUs with CUDA capability greater than 3.0, some new CUDA features including warp shuffle functions are introduced. "__shfl_down" and "__shfl_xor" functions can be utilized to perform a reduction across 32 threads in a warp without consuming any shared memory [9]. The sums of different warps in a block can be added up using atomic function in CUDA to generate a correct block sum. Similar to the GPU-based acquisition, the last step of the GPU-based tracking is to transfer the correlation results from the GPU memory to the CPU memory.

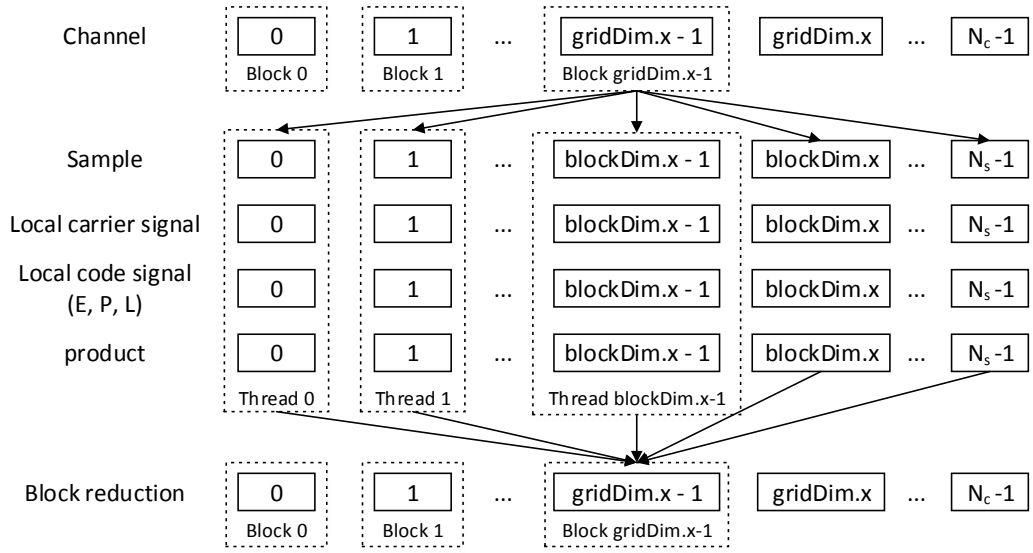

Fig. 5 The GPU-based tracking

NAVSDR provides a tracking view as shown in Fig. 6 to visualize the correlation results of each satellite in the tracking process.

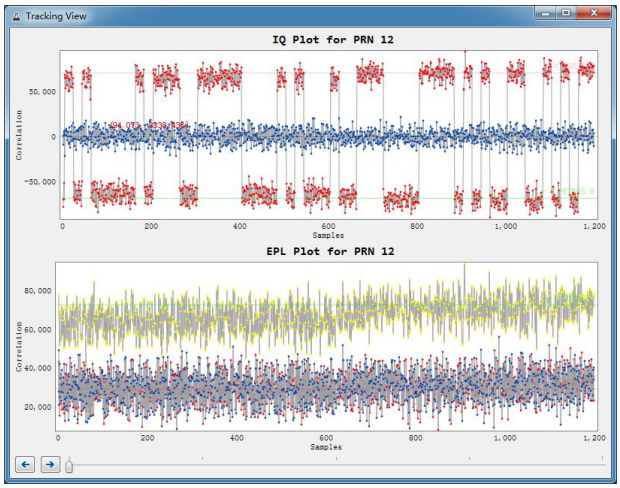

Fig. 6 The tracking view of NAVSDR

In order to evaluate the execution performance of the proposed GPU-based tracking algorithm, the same CPU-based acquisition result is used to initialize both the GPU-based and CPU-based tracking processes. The parameters of the tested GPS signal and the devices used here are the same as that in the performance test of the GPU-based acquisition. In the performance test of the GPU-based tracking, the number of the satellites signals being processed, N_c, is variable, which is different from the situation in the cold start mode of the GPU-based acquisition. A common practice to evaluate the performance of the GPU-based tracking is to obtain the number of tracking channels the GNSS software receiver can maintain in real time [7]. The coherent integration time for each satellite is usually set to be one code length. Since the tested signal here is the GPS L1 C/A, the integration length is 1 ms. The number of the satellites signals being tracked is the number of the visible satellites that have been acquired, which could range between 8-12 satellites. This number is small, and hence in order to test the tracking algorithm under high computational load, massive noise channels are needed so that the extreme power of the GPU can be exerted. In this test, noise channels are simulated by duplicating the normal tracking channels, which contain the real satellites signals. The number of the tracking channels containing both real and simulated satellite signals, N_c, is varying in the range of [0, 1024], with an interval of 1. The execution time of the GPU-based tracking under different numbers of processing channels is collected and plotted in Fig. 7. In order to quantify the execution performance of the GPU-based tracking, a horizontal line indicating 1 ms execution time is drawn to find the corresponding number of processing channels under such situation, which is 330 here. This is the maximum number of channels that the GPU-based tracking can maintain in real time. It should be noted that the execution time of the GPU-based tracking is not 0 when the number of the processing channels is 0. The memory transfer of the input raw IF data and other overhead processing due to the execution of kernel function are responsible for this phenomenon.

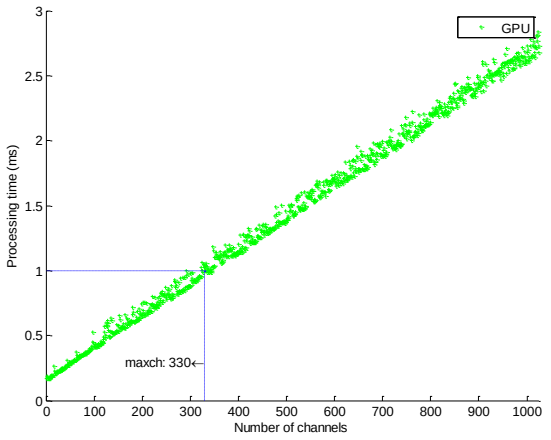

Fig. 7 The performance of the GPU-based tracking

Besides the performance evaluation of the GPU-based tracking, a comparison of the GPU-based tracking and the serial version of the CPU-based tracking is also conducted. The speedup brought by the GPU-based tracking under different numbers of the processing channels is plotted in Fig. 8. This plots shows that the speedup grows with the number of the processing channels increased until it reaches a peak value at nearly 50. After that, the speedup is prone to be a constant number, and its growth rate decreases gradually to 0. There is a transient state for the speedup to grow to the peak because the GPU is not fully occupied when the number of the processing channels is still small. The increase in the number of the processing channels in small values will not lead the execution time growth of the GPU-based tracking to be as significant as that of the CPU-based tracking. While the computational load grows to be great enough with the number of the processing channels increased, both the GPU and the CPU are fully utilized. The speedup will then meet a bottleneck, which represents the extreme performance gap between the GPU and the CPU devices when implementing the tracking algorithm.

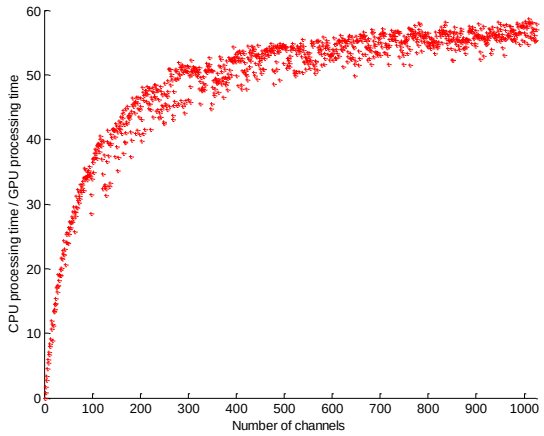

Fig. 8 The speedup of the GPU-based tracking

The speedup peak of the GPU-based tracking is larger than the speedup peak of the GPU based acquisition. The main reason is that the FFTW library used in the CPU based acquisition supports Single Instruction Multiple Data (SIMD) instructions [14], which speed up the computation on the CPU up to above 10 times faster than without using them in an ideal situation [15]. As for the CPU-based tracking, there is no such accelerating methods used. Therefore, the benchmark of the GPU-based acquisition is higher than that of the GPU-based tracking. This leads to the conclusion that the speedup brought by the GPU-based tracking is more significant.

6. MODULAR ASSEMBLY

NAVSDR adopts a modular design so that it can accommodate newly developed algorithms agilely. The GPU-based acquisition and tracking algorithms are implemented, and compiled into dlls, which are composed of the same interfaces as the CPU-based versions. The GPU-based acquisition and tracking components can replace the CPU-based acquisition and tracking components to perform the same functions in NAVSDR. The modular design not only facilitates the assembly of different algorithms, but also enables the developer of each component of NAVSDR to work independently and concurrently. A GUI interface is developed to facilitate the usage of NAVSDR. It can display the constellation plot, the carrier to noise ratios (C/N0) of the signals, the positioning results, and the Dilution of Precisions (DOP) as shown in Fig. 9.

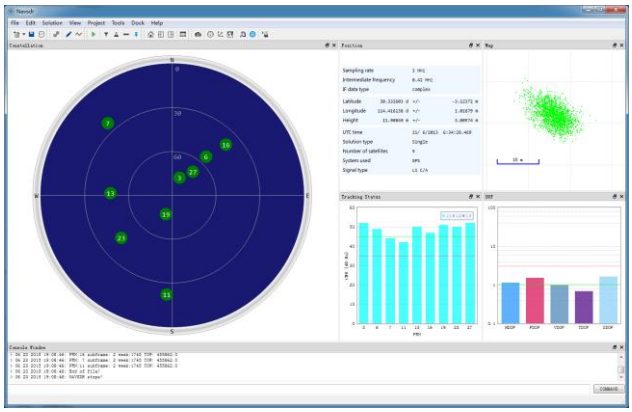
Fig. 9 GUI interface of NAVSDR

7. CONCLUSIONS

In this paper, a GPU-based modular GPS software receiver named “NAVSDR” is introduced. NAVSDR adopts a modular design and is composed of seven well-defined components. The modular division makes NAVSDR more scalable and reusable for the newly added algorithms, such as the GPU-based acquisition and tracking. The paper proposes and describes novel GPU-based designs for signal acquisition and tracking adopted in NAVSDR. The proposed GPU-based acquisition and tracking are implemented and optimized using CUDA. Their execution performance is evaluated and compared with their CPU-based counterparts. The results asserted that the GPU-based acquisition and tracking can efficiently accelerate the execution of the baseband processing in NAVSDR. The GPU-based tracking can even maintain above 300 channels’ processing in real time. The paper also presents the mechanism of the modular assembly when integrating the GPU-based acquisition and tracking into NAVSDR.

ACKNOWLEDGMENTS

The work in this paper was supported by the National Natural Science Foundation of China (41174028, 61273053, 41404029), China Postdoctoral Science Foundation funded project (2013M542061, 2014T70738), the Fundamental Research Funds for the Central Universities (2014618020201), and National Natural Science Foundation of Hubei province (2014CFB727).

REFERENCES

[1] Heckler GW, Garrison JL (2006) SIMD correlator library for GNSS software receivers. GPS Solutions, 10: 269-276

[2] Hobiger T, Tadahiro G, Jun A, Yasuhiro K, Tetsuro K (2010) A GPU-based real-time GPS software receiver. GPS Solutions, 14: 207-216

[3] Huang B, Yao Z, Guo F, Deng S, Cui X, Lu M (2013) STARx–A GPU-based Multi-System Full-Band Real-Time GNSS Software Receiver. ION GNSS+ 2013, Institute of Navigation, Nashville, Tennessee, September, 1549-1559

[4] Pany T, Gohler E, Irsigler M, Winkel J (2010) On the state-of-the-art of real-time GNSS signal acquisition–a comparison of time and frequency domain methods. Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference, 1-8

[5] Pany T, Riedl B, Winkel J (2010) Efficient GNSS signal acquisition with massive parallel algorithms using GPUs. International Technical Meeting of the Satellite Division of the Institute of Navigation, 1889-1895

[6] Park KW, Yang JS, Park C, Lee MJ (2014) Implementation of GPGPU-based Real-time Signal Acquisition and Tacking Module for Multi-constellation GNSS Software Receiver. ION GNSS+ 2014, Institute of Navigation, Tampa, Florida, September, 1410-1416

[7] Seo J, Chen YH, De Lorenzo, DS, Lo S, Enge P, Akos D, and Lee J (2011) A real-time capable software-defined receiver using GPU for adaptive anti-jam GPS Sensors. Sensors, 9: 8966-8991

[8] Farber R (2011) CUDA application design and development. Elsevier

[9] Wilt N (2013) The cuda handbook: A comprehensive guide to gpu programming. Pearson Education
Xie G (2009) Principles of GPS and Receiver Design (in Chinese). Publishing House of Electronics Industry

[10] Nvidia (2014) CUFFT Library User's Guide. Available on the Internet

[11] Nvidia (2014) Whitepaper-NVIDIA GeForce GTX 750 Ti. Available on the Internet

[12] Frigo, Matteo, and Steven G. Johnson. "FFTW: An adaptive software architecture for the FFT." Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on. Vol. 3. IEEE, 1998.

[13] Harris M (2007) Optimizing parallel reduction in CUDA. Available on the Internet

[14] Frigo M, Johnson SG (2013) FFTW user's manual. Massachusetts Institute of Technology.

[15] Suzuki T, Kubo N (2014) GNSS-SDRLIB: An Open-Source and Real-Time GNSS Software Defined Radio Library. ION GNSS+ 2014, Institute of Navigation, Tampa, Florida, September, 1364-1375

[16] Nvidia (2014) CUDA C Programming Guide. Available on the Internet

[17] Nvidia (2014) CUDA C Best Practices Guide. Available on the Internet

[18] Nvidia (2014) Maxwell Compatibility Guide. Available on the Internet