Parallel GNSS Satellite Simulator for Real-Time Multi-GNSS Signal Generation

Yongtaek Hwang†, Jaeo Song†, Dohun Kim†, Hoyoung Yoo†
†*Dept. of Electronics Engineering*
Chungnam National University
Daejeon, Republic of Korea
ythwang.cas@gmail.com, josong.cas@gmail.com, dhkim21.cas@gmail.com, hyyoo@cnu.ac.kr

***Abstract*—In Multi-GNSS environments, the simultaneous generation of signals from a large number of satellites increases the volume of data that must be processed by the satellite simulator, resulting in a significant increase in signal generation time. This paper proposes a parallel GNSS satellite simulator that utilizes the GPU to achieve real-time Multi-GNSS signal generation. The proposed architecture offloads the Signal Calculator module of the conventional CPU-based signal generator to the GPU, supporting parallel computation of signal elements and IF signal accumulation across all visible satellites and all samples within a 1 ms interval simultaneously. To validate the generated signals, the IF output file was up-converted to RF via a USRP and fed into a commercial receiver, resulting in successful navigation. Furthermore, a comparison of signal generation speed against the conventional CPU serial architecture and a CPU parallel architecture revealed that the proposed GPU-based architecture achieved approximately 91.3% and 88.3% faster generation times, respectively. In addition, the maximum number of satellites that can be generated in real time increased from 6 in the CPU parallel architecture to 80 in the proposed GPU-based architecture. These experimental results show that the proposed GPU-based architecture can substantially decrease the computational bottleneck of GNSS signal generation and provide sufficient throughput for large-scale real-time satellite signal simulation in Multi-GNSS environments.**

Keywords—GNSS, Satellite Simulator, Signal Generator, GPU, Parallel Processing

I. INTRODUCTION

The Global Navigation Satellite System (GNSS)provides position, velocity, and time (PVT) information to a receiver by processing signals transmitted from satellites orbiting the Earth[1]. Currently operational GNSS constellations include the United States' GPS, Russia's GLONASS, China's BDS, Japan's QZSS, and India's NavIC. In addition, South Korea is currently developing the Korean Positioning System (KPS)[2].

Receivers have evolved from using a single GNSS constellation to simultaneously employing two or more constellations, an approach known as Multi-GNSS. The use of Multi-GNSS increases the number of visible satellites, thereby improving the geometric diversity of satellite configurations and reducing the dilution of precision (DOP), which in turn advances navigation performance[3,4]. Moreover, when a particular GNSS becomes unavailable due to system failure, jamming, or spoofing, the receiver can maintain navigation capability by using signals from alternative constellations, thereby enhancing system robustness [5].

In developing such Multi-GNSS receivers, it is more efficient to use a signal generator that emulates satellite signals rather than relying on live satellite signals. A satellite simulator can conveniently produce various GNSS signals and simulate diverse reception environments, making them widely adopted in receiver development. However, in Multi-GNSS environments, the large number of satellites increases the volume of data to be processed, resulting in prolonged signal generation times. To address this issue, recent studies have investigated GPU-based GNSS signal generation; for example, [6] proposed a GPU-accelerated BDS signal generation method that improves digital IF signal generation by optimizing memory and enabling parallel GPU computation. Therefore, it is necessary to further improve the operational speed and the scalability of satellite simulators for large-scale signal generation.

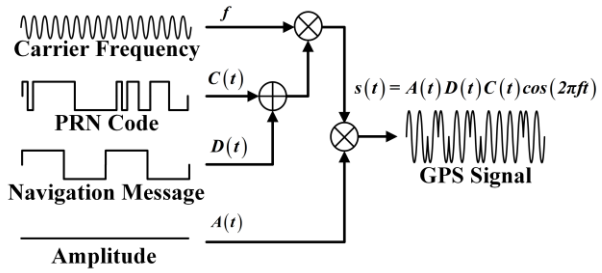

Fig. 1. GNSS signal structure

Serial processing using a single CPU core has innate limitations in handling massive volumes of data at high speed. A prominent approach to overcome this limitation is to employ GPU-based parallel processing. While [6] mainly focused on kernel-level acceleration and memory-access optimization, this paper focuses on the architectural redesign of a conventional GNSS satellite simulator. This paper proposes a parallel GNSS satellite simulator that utilizes the GPU to achieve real-time Multi-GNSS signal generation. The signals generated by the proposed simulator are validated using a commercial receiver, and the generation speed is compared with that of conventional CPU serial and CPU parallel architectures to evaluate the data processing efficiency and real-time satellite generation capacity in Multi-GNSS environments.

II. BACKGROUND

A. GNSS Signal Structure

A GNSS receiver determines its position based on the differences in time of arrival (TOA) of signals received from multiple satellites[7]. The range to each satellite is computed by multiplying the signal propagation time by the speed of light, yielding a measurement known as the pseudorange. Since pseudorange-based positioning involves four unknowns,

979-8-3315-5204-6/26/$31.00 ©2026 IEEE

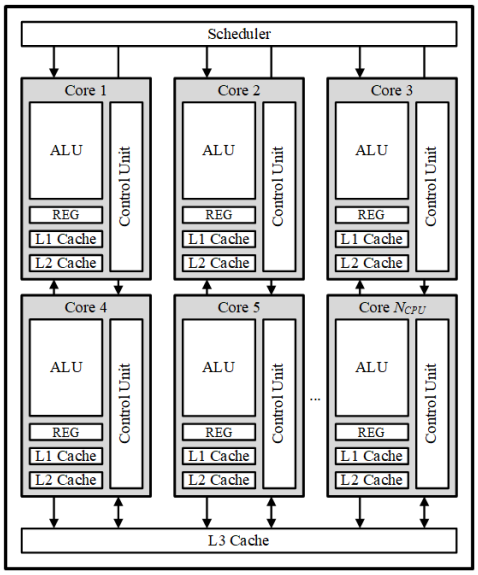

Fig. 2. CPU architecture

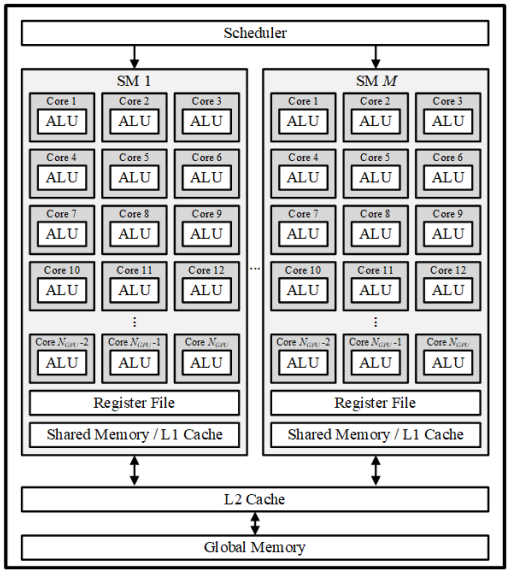

Fig. 3. GPU architecture

namely the three-dimensional receiver coordinates (x, y, z) and the receiver clock bias, signals from a minimum of four satellites must be received simultaneously to compute a position solution.

As illustrated in Fig. 1, the fundamental structure of a GNSS signal can be expressed as in Eq. (1).

$$s(t) = A(t)D(t)C(t)\cos(2\pi f t)\,. \qquad (1)$$

where $s(t)$ denotes the GNSS signal at time t, $A(t)$ is the signal amplitude, $D(t)$ is the navigation message, $C(t)$ is the pseudorandom noise (PRN) code, and f is the carrier frequency. The signal amplitude is determined by the power received at the receiver antenna, which varies with the satellite-to-receiver distance and atmospheric conditions. The navigation message contains satellite orbital information, known as ephemeris data, satellite clock correction parameters, and ionospheric correction coefficients, enabling the receiver to compute satellite positions and pseudoranges for positioning. Accordingly, the signal generator must accurately incorporate the navigation message to produce realistic signals. GNSS employs Code Division Multiple Access (CDMA) to distinguish signals from multiple satellites within the same frequency band. Each satellite is assigned a unique PRN code, and the receiver separates individual satellite signals by correlating the received signal with locally generated PRN codes[8]. The carrier frequency is uniquely assigned to each GNSS constellation and frequency band; for example, the GPS L1 C/A signal uses 1575.42 MHz.

B. CPU & GPU Structure

Fig. 2 and Fig. 3 illustrate the architectures of the CPU and GPU, respectively. A CPU consists of multiple cores, each comprising an arithmetic logic unit (ALU), a control unit, registers, and dedicated L1 and L2 caches[9]. A shared L3 cache is accessible by all cores, facilitating inter-core data exchange. The CPU is equipped with relatively complex control logic, making it well-suited for handling sophisticated control flows. Consequently, the CPU can execute sequential and complex operations at high clock frequencies. However, since the number of cores is limited to the tens, the CPU has built-in limitations for parallel processing tasks that require simultaneous handling of big datasets.

The GPU, originally designed for graphics processing, has been widely adopted for general-purpose computing (GPGPU) owing to its massive parallel processing capability. A GPU comprises multiple streaming multiprocessors (SM), each containing tens to hundreds of cores. Individual GPU cores have a simpler architecture than CPU cores, consisting primarily of ALUs without complex control logic[10]. Instead, the cores within a single SM share a register file and shared memory/L1 cache, operating in a single instruction, multiple threads (SIMT) fashion, where the same instruction is applied to multiple data elements concurrently. All SMs share the L2 cache and global memory, enabling access to large-scale datasets. Due to these architectural characteristics, the GPU has lower single-thread performance than the CPU. However, it can execute thousands of threads simultaneously, achieving high throughput for tasks that involve repetitive operations on large volumes of data. NVIDIA provides the Compute Unified Device Architecture (CUDA) programming model, which enables developers to write parallel programs that execute on the GPU[9,10]. In the CUDA model, computations are organized as grids of thread blocks, where each individual thread block is mapped to an SM and individual threads execute the same kernel function on different data elements. This programming model is especially well-suited for data-parallel workloads in which the same operation must be applied independently to a large number of elements. In particular, GNSS signal generation involves repetitive per-sample and per-satellite computations with minimal data dependencies between operations, making it an ideal candidate for GPU-based acceleration.

C. Conventional GNSS Satellite Simulator

Fig. 4 shows the architecture of a conventional CPU-based GNSS satellite simulator, which serves as the baseline for the proposed approach[12]. The simulator takes as input user parameters such as the IF and sampling frequency, receiver dynamics including attitude and trajectory information, a RINEX file containing satellite ephemeris data, and PRN code tables for each satellite. The output is an IF signal stored in a binary file. The simulator consists of two main modules: the Pseudorange Calculator and the Signal Calculator. The Pseudorange Calculator computes satellite coordinates using the Keplerian orbital equations based on ephemeris data from

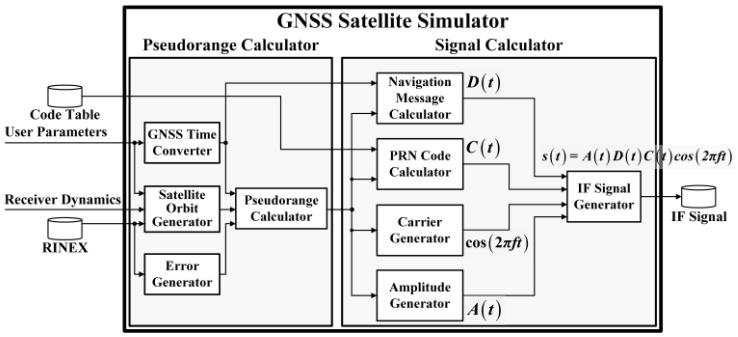

Fig. 4. Conventional GNSS satellite simulator

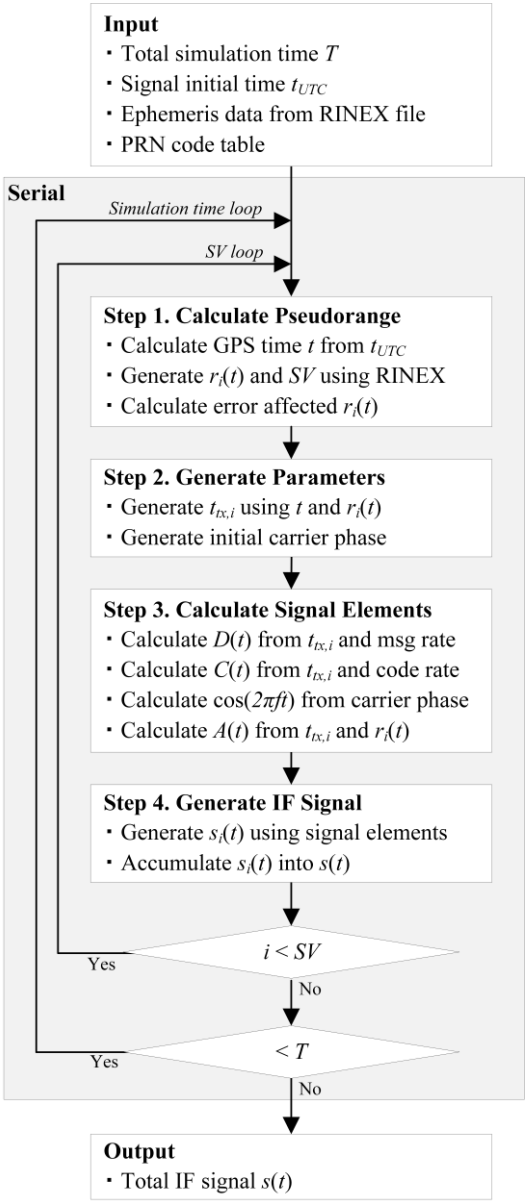

Fig. 5. Conventional GNSS satellite simulator flowchart

the RINEX file, and it calculates the pseudoranges between each satellite and the receiver, accounting for error sources such as ionospheric and tropospheric delays and satellite clock bias. The Signal Calculator then computes the signal components described in Eq. (1), namely the navigation message $D(t)$, PRN code $C(t)$, carrier $\cos(2\pi ft)$, and signal amplitude $A(t)$, and combines them to generate the final IF signal for each visible satellite.

Fig. 5 presents the flowchart of the conventional CPU-based GNSS satellite simulator. In Step 1, the GPS time t is computed from the input UTC time t_{UTC}, and the pseudorange $r_i(t)$ and the number of visible satellites SV are determined using the RINEX file, followed by the calculation of error-affected pseudoranges. In Step 2, the transmission time $t_{tx,i}$ for each satellite is derived from the pseudoranges, and the initial carrier phase required for carrier generation is computed[13]. In Step 3, the navigation message, PRN code, carrier, and signal amplitude for each satellite are calculated on a per-sample basis using the previously computed parameters, and this process is repeated for all visible satellites. In Step 4, the individual signals $s_i(t)$ are accumulated to produce the final IF signal $s(t)$. This entire process is repeated until the simulation time T is reached. In this architecture, Steps 1 through 4 are executed sequentially within a loop over the visible satellites. That is, for each sample, the signal elements must be computed and accumulated one satellite at a time. Consequently, the signal generation time increases linearly with both the number of visible satellites and the number of samples. For example, when generating a GPS L1 C/A signal with a sampling frequency of 25 MHz, 25,000 samples must be processed per millisecond, and this computation must be repeated for every visible satellite. In a Multi-GNSS environment where 30 or more satellites may be visible simultaneously, the computational burden becomes substantial, leading to signal generation times that exceed real-time requirements. This constitutes a fundamental processing bottleneck, motivating the development of a parallel-processing-based approach.

III. Proposed GNSS Satellite Simulator

To overcome the processing bottleneck of the conventional CPU-based architecture described in the previous section, this paper proposes a GPU-based GNSS satellite simulator that leverages massive parallelism to accelerate signal generation. The proposed GPU-based GNSS satellite simulator retains the same two-module structure as the conventional architecture shown in Fig. 3. However, the key distinction is that the Signal Calculator module is offloaded from the CPU to the GPU for parallel execution. The rationale for this partitioning is based on the computational characteristics of each module. The Pseudorange Calculator involves sequential per-satellite computations, including orbit determination using Keplerian equations and pseudorange calculation with error modeling. These operations require conditional branching and iterative convergence, which are better suited for CPU execution. In contrast, the Signal Calculator is dominated by repetitive per-sample and per-satellite computations, where the same set of operations, namely navigation message lookup, PRN code

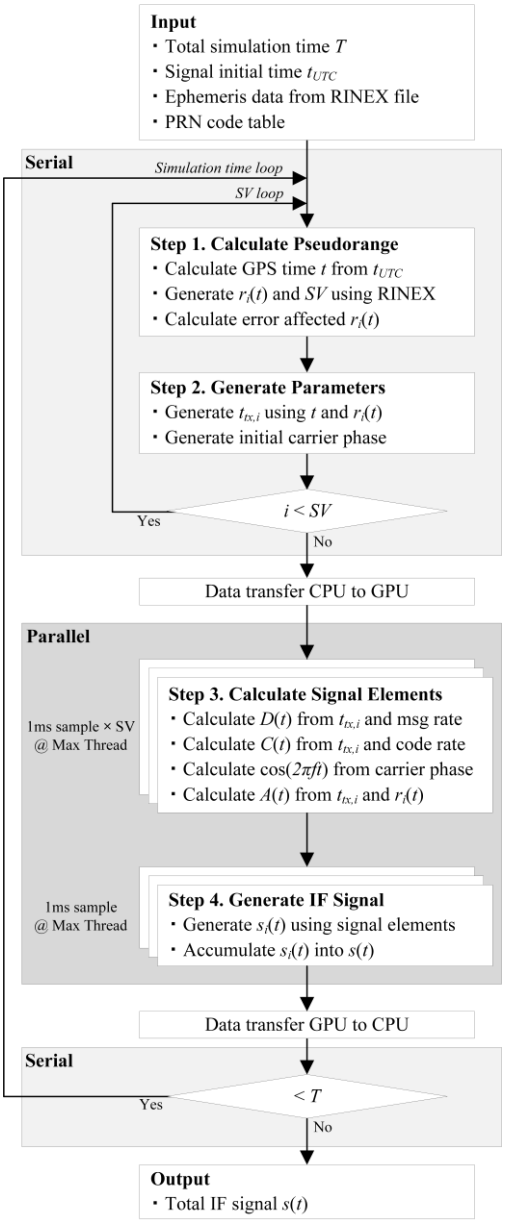

Fig. 6. Proposed GNSS satellite simulator flowchart

generation, carrier synthesis, and amplitude calculation, must be applied independently to each sample for each visible satellite. This computational method, characterized by high data parallelism and minimal inter-thread dependencies, is ideally suited for GPU execution using the CUDA programming model[11]. In the proposed architecture, each GPU thread is responsible for computing the signal elements and generating an IF sample for a specific satellite and sample index combination, enabling thousands of computations to run concurrently.

To make the CUDA kernel configuration more explicit, the proposed Signal Calculator is formulated as a two-dimensional sample–satellite parallel structure. Let N_{samp} denote the number of samples processed within one interval. For a sampling frequency F_s and a processing interval T_{int}, the number of samples is given by

$$N_{samp} = F_s T_{int}. \tag{2}$$

In this work, T_{int} is set to 1ms. The CUDA block dimension is defined as

$$\mathbf{B} = (B_x, B_y), \tag{3}$$

where B_x and B_y denote the number of samples and satellites assigned to a thread block, respectively. The corresponding grid dimension is determined as

$$\mathbf{G} = \left(\left\lceil \frac{N_{samp}}{B_x} \right\rceil, \left\lceil \frac{SV}{B_y} \right\rceil\right). \tag{4}$$

For each GPU thread, the sample index n and satellite index i are calculated as

$$\mathrm{n} = b_x B_x + t_x, \qquad \mathrm{i} = b_y B_y + t_y \tag{5}$$

where b_x and b_y are the block indices, and t_x and t_y are the thread indices. Threads satisfying $n < N_{samp}$ and $i < SV$ compute the signal component of the *i*-th satellite at the *n*-th sample. The block dimensions are selected to satisfy

$$B_x B_y \leq 1024, \tag{6}$$

which is the maximum number of CUDA threads per block.

Fig. 6 presents the operational flow of the proposed GPU-based GNSS satellite simulator. Steps 1 and 2 are executed on the CPU, identical to the conventional architecture. In Step 1, the pseudoranges and visible satellite information are computed, and in Step 2, the transmission times and initial carrier phase are derived. Upon completion of these steps, the computed parameters are transferred from CPU memory to GPU global memory via the PCIe bus. Steps 3 and 4 are then executed as GPU kernel functions. In Step 3, the CUDA kernel is launched using the sample–satellite mapping defined above. Each thread computes the signal elements required for the corresponding satellite and sample, including $D(t)$, $C(t)$, $\cos(2\pi ft)$, and $A(t)$. Thus, the signal elements for all samples and all visible satellites within a 1 ms interval are computed in parallel. In Step 4, the signal elements computed in Step 3 are combined to generate the per-satellite IF signal data $s_i(t)$. The generated per-satellite IF signal data are temporarily stored in shared memory within each thread block, and the signal data corresponding to the same sample index are accumulated across multiple satellites to form the final received IF signal sample $s(t)$. This shared-memory-based accumulation structure reduces repeated global memory accesses during the accumulation of satellite signals and mitigates memory access latency. After the final IF signal data are generated, they are transferred from GPU global memory to CPU memory and stored in a binary file. This process repeats for each 1 ms interval until the simulation time T is reached.

Fig. 7 shows the timing diagram of the conventional CPU-based GNSS satellite simulator. In the CPU-based architecture, each iteration corresponds to the generation of a single sample. Within each iteration, the pseudorange calculation, parameter generation, signal element computation, and IF signal accumulation are performed sequentially from the first satellite to the last satellite. The computation for the next satellite begins only after all operations for the current satellite are completed, and one sample is finalized only when all visible satellites have been processed. Consequently, the time required per iteration increases proportionally with the number of visible satellites. Furthermore, the total processing time is the product of the per-iteration time and the total number of samples to be generated, both of which scale with the simulation parameters. This serial dependency fundamentally limits the throughput of the conventional

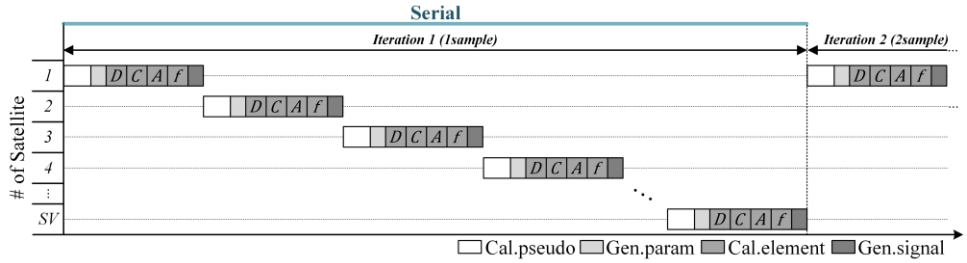

Fig. 7. Conventional GNSS satellite simulator timing diagram

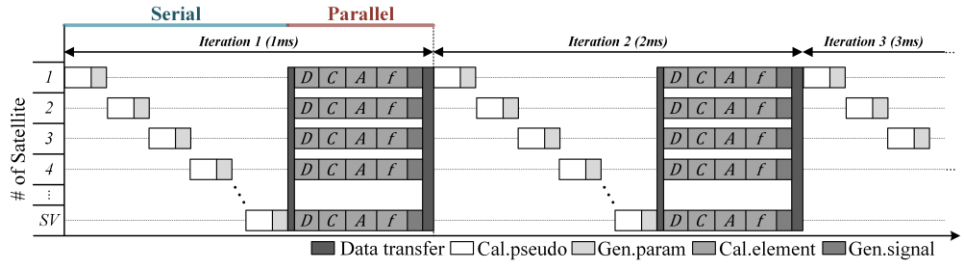

Fig. 8. Proposed GNSS satellite simulator timing diagram

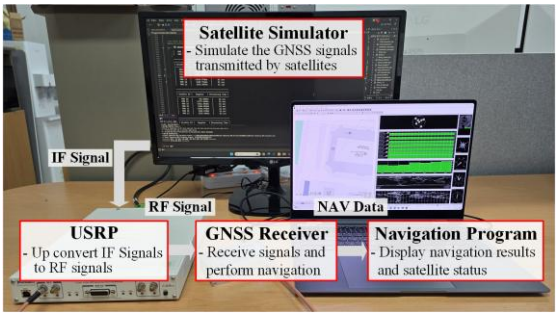

Fig. 9. Experimental environment

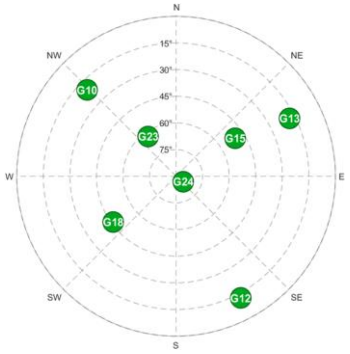

Fig. 10. Skyplot of the acquired satellites

architecture. Fig. 8 shows the timing diagram of the proposed GPU-based GNSS satellite simulator. In the GPU-based architecture, each iteration processes all samples within a 1 ms interval. Each iteration is divided into three phases: a CPU computation phase, a data transfer phase, and a GPU computation phase. During the CPU phase, pseudorange calculation and parameter generation are performed for all visible satellites. The computed parameters are then transferred to the GPU during the data transfer phase. In the GPU computation phase, signal element computation and IF signal accumulation for all visible satellites across all samples within the 1 ms interval are executed in parallel. Upon completion, the generated IF signal is transferred back to the CPU, and the next iteration proceeds.

Compared to the conventional CPU-based architecture, the proposed GPU-based architecture exhibits two key differences. First, whereas the CPU-based architecture processes per-satellite computations sequentially, the GPU-based architecture assigns independent GPU threads to satellite–sample pairs, enabling parallel computation across the satellite and sample dimensions. Second, whereas the CPU-based architecture operates on a per-sample iteration basis, the GPU-based architecture operates on a 1 ms iteration basis, processing the samples within the interval in parallel. These characteristics make the proposed architecture advantageous for GNSS signal generation scenarios that require a large number of repetitive per-satellite and per-sample computations. However, each iteration requires data transfer between the CPU and GPU via the PCIe bus, introducing communication overhead. Therefore, the performance benefit of the proposed architecture is evaluated based on the measured execution time results presented in the experimental section.

IV. Experimental Results

To evaluate the proposed GPU-based GNSS satellite simulator, the testbed shown in Fig. 9 was configured using a PC-based simulator, a USRP for IF-to-RF up-conversion, and a commercial u-blox receiver. The simulator generated GPS L1 C/A signals with a sampling frequency of 25 MHz and an intermediate frequency of 5.42 MHz. Signals from seven visible satellites were generated for 100 s, starting at 06:00 UTC on August 16, 2025, and the resulting RF signal was used for receiver navigation. As shown in Figs. 10 and 11, the acquired satellite geometry was consistent with the simulator configuration, and successful navigation was achieved with horizontal and vertical position errors of 0.725 m and 0.954 m, respectively. These results confirm that the proposed simulator generates valid GNSS signals that can be processed by a commercial receiver.

The signal generation speed was then compared with conventional CPU serial and CPU multi-threaded parallel architectures using the same signal configuration. The experimental platform consisted of an Intel Core Ultra 9 285K CPU, an NVIDIA GeForce RTX 3060 GPU, and 64 GB of RAM. GPU kernel profiling with NVIDIA Nsight Compute showed that the proposed Signal Calculator kernel achieved over 80% occupancy, indicating effective GPU resource

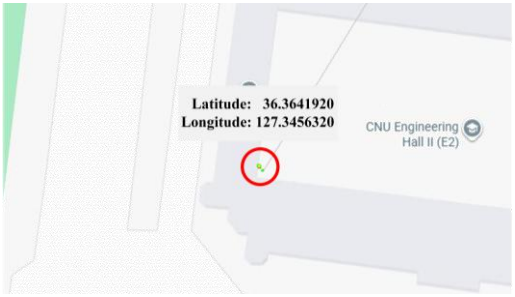

Fig. 11. Navigation results using u-blox receiver

utilization. As shown in Fig. 12 and Table I, the CPU serial architecture required 108.333 s to generate the 100-s signal, while the CPU parallel architecture reduced the generation time to 80.623 s. In contrast, the proposed GPU-based architecture completed the generation in 9.382 s, including 1.306 s for CPU-to-GPU transfer, 5.153 s for GPU kernel execution, and 2.223 s for GPU-to-CPU transfer. Although data transfer introduced additional overhead, the reduction in Signal Calculator execution time dominated the total performance improvement. Consequently, the proposed architecture achieved approximately 91.3% and 88.3% faster generation times than the CPU serial and CPU parallel architectures, respectively. Moreover, the maximum number of satellites that could be generated in real time increased from 8 to 80, demonstrating the suitability of the proposed architecture for large-scale Multi-GNSS signal generation.

V. Conclusion

This paper presented a GPU-based GNSS satellite simulator for efficient signal generation in Multi-GNSS environments. The proposed architecture offloads the Signal Calculator module to the GPU, enabling parallel computation of signal elements and the accumulation of IF signals across all visible satellites and samples. The generated signals were validated using a USRP and a commercial u-blox receiver, confirming successful navigation with horizontal and vertical position errors of 0.725 m and 0.954 m, respectively. Compared with conventional CPU serial and CPU parallel architectures, the proposed GPU-based simulator achieved up to 91.3% faster signal generation while increasing the number of satellites generated in real time from 6 to 80. These results demonstrate that the proposed architecture effectively reduces the computational bottleneck of GNSS signal generation and is suitable for real-time large-scale satellite simulation. Future work will extend the simulator to additional GNSS signal types, such as GPS L5, Galileo E1, and BDS B1I, and incorporate more realistic reception environment models, including multipath, signal blockage, and jamming scenarios.

Acknowledgment

This research was supported by the National Research Foundation of Korea (NRF) (No. 2022R1A5A8026986) and the Ministry of Science and ICT (MSIT) under the ITRC program (IITP-2025-RS-2024-00436406 (50%)) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

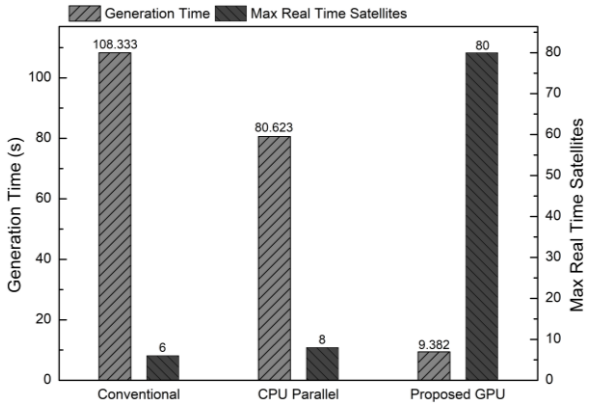

Fig. 12. Results of generation time and number of real-time satellites

TABLE I. Comparison of Performance by Architecture

Component	Conventional	CPU Parallel	Proposed GPU
Pseudorange Calculator	0.468s	0.546s	0.700s
Data Transfer (CPU-GPU)	-	-	1.306s
Signal Calculator	107.865s	80.077s	5.153s
Data Transfer (GPU-CPU)	-	-	2.223s
Total	108.333s	80.623s	9.382s
# Real-Time Satellites	6	8	80

References

[1] C. J. Hegarty, "GNSS Signals - An Overview," *2012 IEEE Int. Freq. Control Symp. Proc.*, vol. 1, pp. 1–7, 2012.

[2] J.-M. Joo and M.-B. Heo, "Korean Positioning System Development Plan," in *2020 IPNT Conference*, Nov. 2020, pp. 29–30.

[3] W. Enderle et al., "Space User Visibility Benefits of the MultiGNSS Space Service Volume: An Internationally-Coordinated, Global and Mission-Specific Analysis," *Proc. 31st Int. Tech. Meet. Satell. Div. Inst. Navig. (ION GNSS 2018)*, pp. 1191–1207, 2018.

[4] C. Meneghini and C. Parente, "Advantages of Multi GNSS Constellation: GDOP Analysis for GPS, GLONASS and Galileo Combinations," *International Journal of Engineering and Technology Innovation*, vol. 7, no. 1, pp. 01–10, 2017.

[5] A. Elmezayen, M. Karaim, H. Elghamrawy, and A. Noureldin, "Enhanced GNSS Reliability on High-Dynamic Platforms: A Comparative Study of Multi-Frequency, Multi-Constellation Signals in Jamming Environments," Sensors, vol. 23, no. 23, p. 9552, 2023.

[6] P. Shi, S. Wu, T. Zeng, and R. Xue, "Satellite Navigation Simulation Signal Generation Method Based on GPU Acceleration," *J. Math. Inform.*, vol. 20, pp. 39–49, 2020.

[7] E. D. Kaplan and C. J. Hegarty, *Understanding GPS/GNSS: Principles and Applications*, 3rd ed. Norwood, MA, USA: Artech House, 2017.

[8] U.S. Space Force, "Navstar GPS Space Segment/Navigation User Interfaces," Interface Specification IS-GPS-200, Rev. N, Aug. 2022.

[9] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 6th ed. Cambridge, MA, USA: Morgan Kaufmann, 2019.

[10] NVIDIA Corporation, "CUDA C++ Programming Guide," ver. 12.6, 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[11] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *ACM Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008.

[12] H. Wang, J. Chang, J. Lv, et al., "An implementation scheme of GNSS signal simulator based on DSP and FPGA," in *Proc. Int. Conf. Computer Science and Service System (CSSS)*, IEEE, 2011, pp. 27–29.

[13] J. B.-Y. Tsui, *Fundamentals of Global Positioning System Receivers: A Software Approach*, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2005.