Software-Defined GNSS is Ready for Launch

Hailey A. Nichols*†, Matthew J. Murrian‡, Todd E. Humphreys*
**Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin*
†*Locus Lock, Inc.*
‡*Coherent Technical Services, Inc.*

BIOGRAPHIES

Hailey A. Nichols (BS, Aerospace Engineering, Massachusetts Institute of Technology; MS, Aerospace Engineering, University of Texas at Austin) is a graduate student in the department of Aerospace Engineering and Engineering Mechanics at The University of Texas at Austin, and a member of the UT Radionavigation Laboratory. Her research interests currently include GNSS signal processing and commercial applications for software-defined radio. She is also the founder of Locus Lock, a start-up spinning out of the Radionavigation Lab, that provides end-users with precise PNT solutions through GNSS SDR.

Matthew J. Murrian (BS, Mechanical Engineering, University of Central Florida; MS, Aerospace Engineering, University of Texas at Austin) is a chief engineer at Coherent Technical Services, Inc. and a past member of the UT Radionavigation Laboratory. He specializes in digital signal processing, software-defined radio, and multi-sensor navigation and perception. He is the lead developer of GRID GNSS software-defined radio.

Todd Humphreys (BS, MS, Electrical Engineering, Utah State University; PhD, Aerospace Engineering, Cornell University) is a professor in the department of Aerospace Engineering and Engineering Mechanics at The University of Texas at Austin, where he directs the Radionavigation Laboratory. He specializes in the application of optimal detection and estimation techniques to problems in secure, collaborative, and high-integrity perception, with an emphasis on navigation, collision avoidance, and precise timing. His awards include The University of Texas Regents' Outstanding Teaching Award (2012), the National Science Foundation CAREER Award (2015), the Institute of Navigation Thurlow Award (2015), the Qualcomm Innovation Fellowship (2017), the Walter Fried Award (2012, 2018), and the Presidential Early Career Award for Scientists and Engineers (PECASE, 2019). He is a Fellow of the Institute of Navigation and of the Royal Institute of Navigation.

ABSTRACT

This paper explores how advancements in computer processing, both in single instruction, multiple data (SIMD) and multicore technology, have shaped the growth of software-defined Global Navigation Satellite Systems (GNSS) receivers. Historically, GNSS software-defined radio (SDR) has been limited to research and development purposes. But now, modern processor architectures and instruction sets are particularly efficient, paving the way for more capable SDR. GRID, the GNSS SDR developed in the Radionavigation Lab, has recently achieved a remarkable inflection point: under some processing configurations, the correlation operation, by which each channel's signal is mixed to baseband and de-spread via multiplication against a local code replica, is no longer the bottleneck process. This important milestone in pure software-defined GNSS makes SDR a formidable competitor against traditional mass-market application-specific integrated circuit (ASIC)-based GNSS receivers. Further, this paper offers an exploration of commercial use cases particularly well-suited for GNSS SDR: space-applications, wall-mounted electronic technologies, and automated vehicles. In detailing the status of GRID and its various applications, this paper presents the case that software-defined GNSS is ready for launch for mass market applications, rather than solely a tool for research and development.

INTRODUCTION

Software-defined GNSS, in which all GNSS receiver signal processing downstream of sampling and quantization is performed on a general purpose processor rather than an ASIC or FPGA, has seen its fortunes rise, fall, and rise again over the past quarter century. In 1996, GPS pioneer Philip Ward pronounced software-defined GNSS dead on arrival: "It is unlikely that the speed-power product of general purpose digital signal processors will make them the suitable choice to perform the code and carrier wipe-off function in the near future, perhaps never" [1]. The only GNSS SDR at the time was Dennis Akos's GPS/GLONASS SDR, which featured a novel front-end design that permitted sampled data collection and storage, but did not have the computational power to continuously track GNSS signals in real-time [1]. In 1999, the first GNSS SDR that operated in "real-time" operation emerged from Akos and co-authors in [2], which was able to process 60 seconds of IF data in 55 seconds. This effort was a notable achievement in the history of GNSS SDR. In 2004, Brent Ledvina and co-authors presented the first real-time dual-frequency (L1 C/A, L2C) software-defined GPS receiver, which supported 10 tracking channels [3]. 2007 was a banner year for GNSS SDR: Cambridge Silicon Radio spent $75M to buy NordNav, a Swedish company that had developed a software-defined GNSS receiver for use in cell phones and other embedded applications. CSR's purchase sparked great commercial interest in SDR technology. But the marginal power draw increase due to the NordNav SDR solution, although remarkably low for an SDR, was still too high for mass market cell phones. On this realization, CSR mothballed the NordNav software, opting instead to buy SiRF, which offered a high-performance, low-power, ASIC-based solution.

By 2016, embedded processing power had become more efficient. In that year, Trimble rolled out a software-defined GNSS receiver called Catalyst for widespread commercial use. The Trimble Catalyst SDR consumed digitized data from an radio frequency (RF) front-end which was embedded in a handheld antenna (DA1). The SDR harnessed the processing power of a smartphone or tablet to compute a user's position and time. DA2, Trimble's second-generation Catalyst product, now bundles a processor with the antenna to relieve the burden on the phone, but the solution remains software-defined and thus maintains the attractive flexibility of GNSS SDR [4].

Now in 2022, advancements in processor speeds and parallel instructions and architectures have unlocked expanded possibilities. The following sections detail the current status and performance of a state-of-the art software-defined GNSS receiver, GRID. The GRID receiver exploits parallelism at multiple levels of operation: (1) it performs correlation using the bit-wise parallel technique developed by Psiaki and Ledvina [5], (2) it supports hundreds of channels in real-time by distributing processing across multiple general purpose cores [6], and (3) it makes full use of single-instruction multiple-data (SIMD) instructions to accelerate correlation arithmetic and bit manipulation.

Harnessing the latest SIMD instruction sets, GRID's performance has recently achieved a remarkable inflection point: under some processing configurations, the correlation operation is no longer the bottleneck process. This represents a major milestone for software-defined GNSS: the very operation that Philip Ward warned in 1996 could make general purpose processors perpetually unsuitable for GNSS signal processing is now so efficiently implemented that it requires less than half the computational resources. GRID has become a more capable and power-lean GNSS SDR by leveraging these technological advancements.

GNSS SDR from inception has been a valuable platform and tool for research and development (R&D). Efficient, low-power, low-cost multi-core processors developed for smartphones, together with new algorithms tailored to exploit parallelism, have now made GNSS SDRs ready for widespread commercial use, not just for R&D. Some commercial uses for which GNSS SDR is particularly well-suited for are space- and aerial-applications, wall-mounted electronic technologies, automated vehicles, and a host of emerging technologies that require a high-performance GNSS solution.

This paper makes three primary contributions: (i) A comprehensive description of the state-of-the-art in pure GNSS SDR based on bit-wise parallel correlation. (ii) An explanation of how modern processor architectures and instruction sets have led to an inflection point in which correlation is no longer the processing bottleneck. (iii) An exploration of use cases particularly well suited for GNSS SDR.

BRIEF HISTORY OF GRID

GRID is a mature C++ software-defined GNSS receiver (technology ready level 7 by NASA standards) that has been developed over the past 16 years in the Radionavigation Lab (RNL) at the University of Texas at Austin. Led by Dr. Todd Humphreys and students in the RNL, GRID is a highly-optimized multi-core GNSS SDR that can be deployed on a variety of different processing platforms.

GRID has been the basis of many research/commercial applications over the past decade: it has been adapted to perform GNSS spoofing [7]. Also, GRID was the first GNSS SDR to operate in space [8] and the first GNSS receiver to show that centimeter-accurate positioning can be accomplished via a smartphone antenna [9]. GRID has also been utilized on the International Space Station to detect transmitted GNSS interference from terrestrial sources [10]. In more recent years, GRID has been the state-of-the-art receiver of any kind for urban real-time kinematic (RTK) positioning [11], [12].

CONTRIBUTIONS TO THE ADVANCEMENT OF SDR

This paper specifically focuses the GRID GNSS SDR, which is a software suite implemented on a general purpose processor. General purpose processors provide some advantages to traditional hardware-implemented GNSS receivers. In these traditional hardware solutions, all signal processing downstream of the RF front-end is on an application-specific integrated circuit (ASIC), as pictured in Fig. 1.

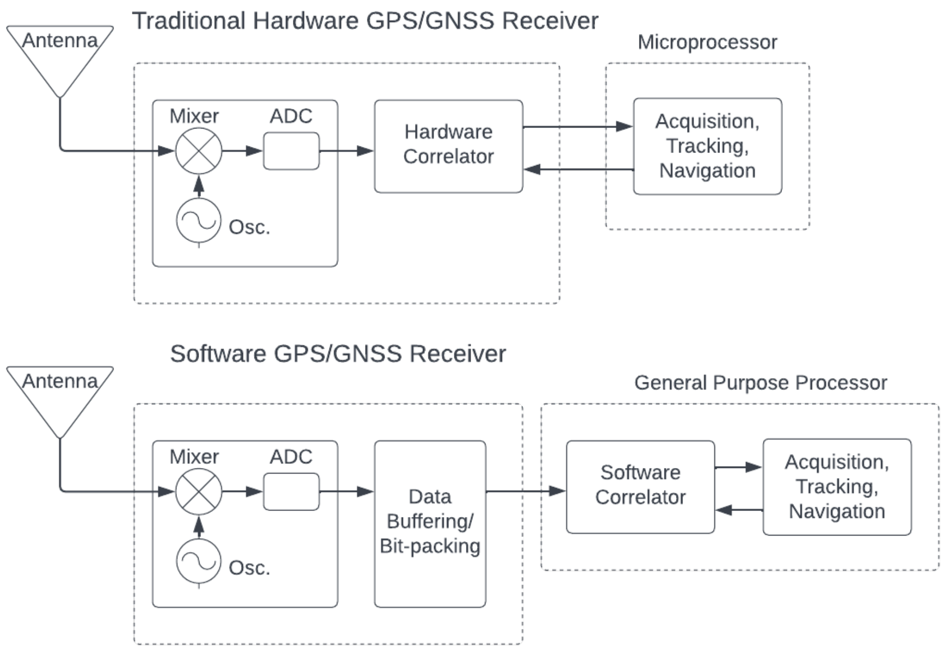

Traditional Hardware GPS/GNSS Receiver
Antenna
Microprocessor
Mixer
ADC
Osc.
Hardware Correlator
Acquisition, Tracking, Navigation
Software GPS/GNSS Receiver
Antenna
General Purpose Processor
Mixer
ADC
Osc.
Data Buffering/ Bit-packing
Software Correlator
Acquisition, Tracking, Navigation

Fig. 1: Block diagram of a hardware GNSS receiver vs. software-defined GNSS receiver, where the correlators are implemented on an ASIC for the hardware receiver and in software for SDR. Picture inspired by [5].

However, in a "pure" software GNSS receiver, all GNSS processing – including the correlation operation – is performed on a general purpose processor. In this software-defined approach, one piece of hardware is still needed, the RF front-end. The front-end downmixes, filters, and digitizes the GNSS signal from the satellites and outputs the raw digital stream to the processor. Within the processor, the software-defined GNSS receives and processes the raw stream of GNSS samples to produce a positioning solution.

Outside of "pure" SDR, there are other software-defined solutions such as Field Programmable Gate Array (FPGA)-based SDR. FPGA-based SDR utilizes hardware solutions, namely multipurpose microchips, to run the software-defined processing suite. The software design is programmed to the FPGA through specific hardware design languages. FPGAs contain an array of programmable logic blocks that are typically reconfigurable and programmable for different purposes.

Most GNSS SDRs can either process raw RF front-end samples in real-time or record samples for post-processing. SDRs implemented on a general purpose processor can replay data much faster than through an FPGA implementation as the latter is typically limited by the FPGA's maximum clock rate. In addition to its replay capabilities, SDRs enable researchers to collect large raw-sample datasets for distribution within the GNSS community, promoting collaboration and repeatable high-fidelity cross-verification. They are also an ideal tool for prototyping and offer continuous software maintenance and upgradeability.

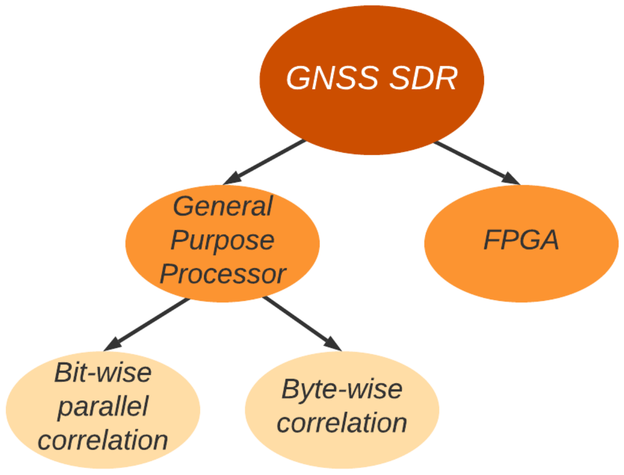


Fig. 2: Mapping of various types of software-defined radio. This paper focuses on "pure" SDR, which are those implemented on general purpose processors. GRID employs the bit-wise parallel correlation technique.

Among pure GNSS SDRs, there are two families: byte-wise SDRs and bit-wise SDRs, as pictured in Fig. 2. Byte-wise SDRs represent each front-end sample (real or complex) as a byte. For bit-wise SDRs, each sample (real or complex) is represented by one or more bits. Each sample is bit-packed with others of its kind (i.e. sign or magnitude) in separate buffers. Bit-wise SDRs can be unpacked with very little operations per bit. For low quantization (i.e. 1 or 2 bit), bit-wise SDR is quite amenable to parallelization and very memory efficient [13]. The GRID GNSS SDR implements the bit-wise parallel correlation technique. The following representations show what single-bit byte-wise vs bit-wise parallel SDR looks like in practice.

Byte-wise SDR
Two-bit Real (IF):

$$\overbrace{[S_1\ X\ X\ X\ X\ X\ X\ X][M_1\ X\ X\ X\ X\ X\ X\ X]\ldots[S_4\ X\ X\ X\ X\ X\ X\ X][M_4\ X\ X\ X\ X\ X\ X\ X]}^{64\ Bits\ Total}$$

Two-bit Complex (IQ):

$$\overbrace{[I_1^S\ X\ X\ X\ X\ X\ X\ X][I_1^M\ X\ X\ X\ X\ X\ X\ X]\ldots[Q_2^S\ X\ X\ X\ X\ X\ X\ X][Q_2^M\ X\ X\ X\ X\ X\ X\ X]}^{64\ Bits\ Total}$$

Bit-wise Parallel SDR
Two-bit Real (IF):

$$\overbrace{[S_1\ S_2\ S_3\ S_4\ M_1\ M_2\ M_3\ M_4]}^{8\ Bits\ Total}$$

Two-bit Complex (IQ):

$$\overbrace{[I_1^S\ I_1^M\ Q_1^S\ Q_1^M\ I_2^S\ I_2^M\ Q_2^S\ Q_2^M]}^{8\ Bits\ Total}$$

Modern Processor Capabilities

Computational power offered by GPPs has continued to increase and the cost for that additional computational power has continued to tumble, thanks to the smartphone revolution [14]. Cheap, generic, embeddable ARM processors are capable of fast bit-wise parallel correlation. Thus, cheap generic central processing units (CPUs), such as a Raspberry Pi 4, can host these cutting-edge signal processing techniques to provide end-users with centimeter-accurate positioning solutions.

Over the past decade, computers have been getting faster, more energy efficient, and more parallelized. This trend has favored the development and efficiency of software-defined radio. Fig. 3 shows the progression of microprocessors over the past 50 years. The advantages of software-defined radio for GNSS processing are compelling as these trends continue. GNSS SDR enjoys lower development times and costs, as well as easier maintenance and porting. In addition to the tremendous benefits of more computing power for SDR, as the costs of computational power continue to decrease, the cost of GNSS SDR will also suppress. This trend has aided and will continue to aid the development and capabilities of GNSS SDR in the future.

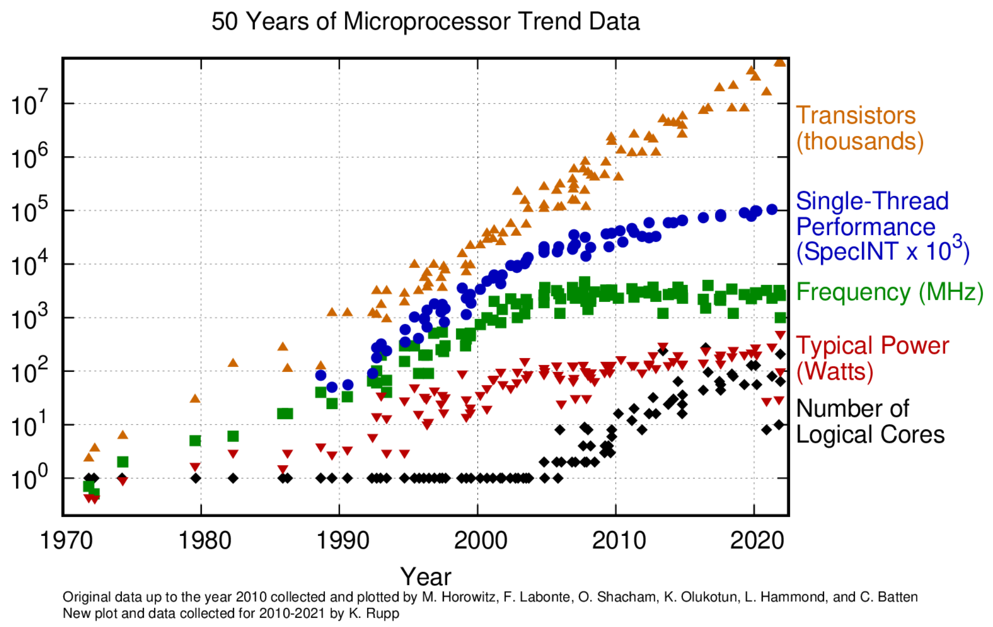


Fig. 3: 50 years of microprocessor trend data from [15].

Historical Evolution of Highly-Efficient SIMD Instruction Sets

The historical evolution of highly efficient vectorized processor instructions, known as SIMD, has aided the progression of GRID. In 1996, the first widely-deployed SIMD instruction set for the x86 architecture was Intel's MMX extensions. Intel released the Streaming SIMD Extensions (SSE) instructions in 1999, which was sparked by advancements in the AltiVec system in the Motorola PowerPC and IBM's POWER systems after the release of the MMX extensions. Since the late 90's, there have been numerous advancements to the SIMD extension sets for multiple architectures.

Intel later developed the Advanced Vector Extensions: AVX, AVX2 and AVX-512. These vector extensions provide new CPU instructions that impact computer functions such as storage and computation. In 2008, Intel released the Advanced Vector Extensions (AVX), which widened the SIMD register from 128 bits to 256 bits. AVX introduced a new coding scheme, new features, and new instructions. Further, AVX2 expanded most integer operations to 256 bits and alongside AVX2 came new operations, such as the fused multiply-add (FMA). The most recent Intel SIMD vector extension set is AVX-512, which has a register width of 512 bits. This extension was first proposed in 2013. AVX-512 compared to its predecessor, Intel AVX2, allows for twice the number of floating-point operations per second (FLOPS) per clock cycle. Specifically, AVX-512 SIMD instructions enable processing of twice the number of data elements that AVX or AVX2 can process, and four times that of the older SSE.

In addition to the traditional SIMD instruction sets like SSE, AVX, and AVX-512 on x86 architectures, there are also intrinsics for ARM architectures, called NEON. SIMD intrinsics are the C/C++ functions used in the compiler that actually implement the SIMD instruction sets. ARM NEON technology is an advanced SIMD architecture extension for the A-profile and R-profile processors. The NEON SIMD architecture is a collection of registers, with register sizes ranging from 8-bit to 128 bits.

Thus, in modern computers, SIMD register sizes typically range from 128 to 512 bits. Most x86 processors, outside of data centers, support up to 256 bit operation, whereas most modern 64-bit ARM processors support only 128-bit SIMD instructions [13]. Therefore, the widest register sizes are often not yet available on more widely available GPPs or embedded computers.

These advancements in SIMD have enabled higher efficiency in GRID, specifically the bit-level manipulation used for the correlation process. The SIMD instruction sets are ideal for processes and demanding calculations that have high data parallelism as they are able to operate on multiple elements of data in a single instruction. Two main processes that highly benefit from exploiting SIMD instruction sets are bit-unpacking and correlation.

Bit-unpacking is the repeated application of a particular combination of bit-wise operations: shifts, masks, assignments. GRID unpacks the bit-wise parallel word representations into circular buffers that are later used in the correlation process. It uses a series of scalar operations to unpack this data. Every bitmask, bitwise-or, and bit-shift is yet another processing instruction. However, by exploiting the efficiency of advanced SIMD instruction sets, the processor performs several of these scalar instructions in a single instruction.

In addition to the bit-unpacking process, SIMD instruction sets have been exploited in other parts of the processing chain, namely, the correlation process. SIMD innovations offer more highly parallelized CPU instructions, which has drastically increased the speed of bit-wise parallel correlation in GRID. The exploitation of SIMD instruction sets for both bit-unpacking and correlation is further detailed in the following sections. Despite this paper's focus on bit-wise SDR, it is important to note that both bit-wise and byte-wise GNSS SDR implementations greatly benefit from exploiting SIMD instruction sets.

OPTIMIZING THE PROCESSING CHAIN

This section presents the major operations within GRID's overall processing chain: data loading/processing, bit-unpacking, carrier replica generation, code replica generation, and correlation. These operations are a subset of the overall processing chain, but are specifically called out as they constitute a majority of the SDR's computational demand. For each operation, details are provided on how the operation has been optimized algorithmically and/or through SIMD instruction sets.

The particular variant of GRID whose optimizations are described below is called the Precise Positioning Receiver (PpRx). PpRx is optimized for implementation on x86- and ARM-based GPPs, as opposed to other GRID implementations targeting general purpose digital signal processors. For the purposes of this paper, one can think of GRID and PpRx as synonymous.

Data Loading and Processing

In GNSS SDR, there exists a performance trade-off concerning how the raw samples are read and processed. The tradeoff must balance thread contention and data/instruction cache locality. It is more computationally efficient to process raw samples for the longest time possible before interrupting the processing and moving on to another activity. If another process is introduced, the computer has to use other memory/cache resources to fetch the new data and instructions. Therefore, it is most efficient to process one "chunk" of data per GNSS log interval, which is the time between position fixes taken by the GNSS receiver. A chunk of data is defined as the optimal amount of data to be processed in one GNSS log interval.

If the user desires an output of one new GNSS position estimate every second, then the computer would read/process in one second intervals. However, for real-time application, by the time the processor starts processing, the first sample of that interval is already one second old. If it takes one additional second to process the one-second interval of data, then the GNSS receiver continuously operates with two seconds of latency.

To reduce latency, the computer needs to load and process smaller chunks of data. If it loads and processes one millisecond intervals, for example, then the lower bound on the processing latency will be one millisecond. Note that this lower bound does not include latencies already introduced before the SDR; i.e., delays incurred between the antenna element and the reading of bit-packed data into the SDR. To optimize the performance of PpRx, the processing chunk size trade-off needs to be well tuned.

PpRx first reads an interval of samples from the radio-frequency front-end hardware. The data loading/processing is completed as follows:

1) PpRx loads 1 large interval of data that is some percentage of the way to the next log interval. For example, the processor may load 65% of the data towards the next 1 interval: the next 650 milliseconds worth of data. This first interval is a blocking read: PpRx waits until all the data in this chunk is available.
2) PpRx then processes that data. During the time it takes to process, some amount of new data will become available.
3) Next, PpRx loads the new data that came available in the time it took to process but it does not go beyond the start of the next log interval. This loading interval is a non-blocking read: PpRx only loads the data that is available without waiting any particular amount of time.
4) Next, PpRx will process that second (typically smaller) interval of data.
5) Again, PpRx loads any new data that came available while processing the previous interval. This time, PpRx blocks to load some minimum amount of data (e.g., 1 milliseconds worth). Every channel update is done 1 code-interval at a time. For example, this blocking value for GPS L1 C/A is 1 millisecond.
6) Process these data.
7) Repeat 5 until the processor has loaded and processed up to the next log interval.

The percentage size of the initial interval loaded in Step 1 is adjusted to ensure at least one load is required in Step 5. It should not be the case that the non-blocking read in Step 3, takes the loading interval all the way up to the next log interval. If all the data is already available without blocking, the receiver is already latent. This requirement sets a maximum for the first percentage loaded for the data processing.

Requiring a minimum of one load in Step 5 allows the software to positively determine that the processor is capable of, on-average, real-time operation. If there is no size small enough for the initial load that achieves a minimum of one load in Step 5, then the processor is overall incapable of real-time processing. If a minimum of one load in Step 5 isn't achieved then the receiver can temporarily drop tracked signals, forego tracking new signals, and/or suspend FFT acquisition.

Beyond this, the processor chooses between latency and overhead. Latency is minimized by loading/processing 1 millisecond at a time. But, overhead is maximized. See the following examples to illustrate two extremes of this processing method.

Suppose the receiver is capable of 2x real-time with the current tracking load (so it can process 1 seconds of data in 500 milliseconds). The first example highlights where the initial read is 666 milliseconds:

1) A blocking read of 666 milliseconds of data.
2) It takes 333 milliseconds to process. Clock time is 0.999 seconds when finished processing.
3) A non-blocking read of 333 milliseconds of data.
4) It takes 166.5 milliseconds to process. Clock time is 1.1655 seconds when finished processing.
5) A blocking read of 1 millisecond of data.
6) It takes 0.5 milliseconds to process. Clock time is 1.166 seconds when finished processing. The receiver is 166 milliseconds latent but it only requires 3 reads.

Now, suppose the initial read is 500 milliseconds:

1) A blocking read of 500 milliseconds of data.
2) It takes 250 milliseconds to process. Clock time is 0.75 seconds when finished processing.
3) A non-blocking read of 250 milliseconds of data.
4) It takes 125 milliseconds to process. Clock time is 0.875 seconds when finished processing.
5) A blocking read of 1 millisecond of data.
6) The remaining 250 milliseconds of data is read and processed in 1 millisecond intervals. Clock time is 1.0005 seconds when finished processing. The receiver finishes with 0.5 milliseconds latency and requires 252 reads.

These two examples highlight two extremes:

1) Minimum number of reads (3) with maximum latency (166 milliseconds), or
2) Maximum number of reads (252) with minimum latency (0.5 milliseconds)

On the other hand, if this technique were not implemented for the example receiver capable of 2x real-time operation, the latency would be much worse off. If the processor chose to only use one read, e.g., use one large blocking read, the latency in real-time output would have been 500 milliseconds. However, if the processor used more than 252 reads, e.g., 1000 reads of 1 millisecond, the latency still would not be better than 0.5 milliseconds. Therefore, the first percentage loaded into PpRx is optimized for both latency and overhead.

Bit-Unpacking Schema

The RF data are streamed to a general purpose processor, where PpRx operates on the raw RF samples. The data are bit-packed with some front-end specific representation that does not have the same organization of the buffers that PpRx will eventually correlate against. PpRx first has to unpack these data into the correlation buffers. This can be processor intensive because PpRx is performing bit-level manipulation on every sample. This processor burden in the correlation operation has traditionally has been a major limitation of GNSS SDR.

The bit-packing pattern needs to be "unpacked" with reasonable efficiency. Ideally, the processor should organize the largest possible units to be extracted into the correlation buffers. If possible, this means the processor should copy an entire word (32 bits or 64 bits) from bit-unpacking to correlation buffers. If the processor has to isolate and copy individual bits, this process becomes quite intensive. Prior work in the RNL showed that exploiting SIMD extensions proved extremely useful for an optimized parallelized bit-unpacking algorithm [13].

The bit-wise parallel correlation technique first introduced in [5] operates on the digitized front-end data that is stored in circular buffers. Each bit in these circular buffers represents a separate sample of data and the bits are ordered sequentially with no gaps. The final organization of the correlation buffers in PpRx is as follows: one independent buffer per quantization bit per antenna. So, the sign bits of the primary antenna samples get extracted and isolated into a single buffer. Each bit in this buffer is a sign bit from a sample. A byte in the buffer stores 8 sign bits. This is also the case for the magnitude bits from the primary antenna. Also, for however many other antennas the front-end has (i.e. primary, secondary, etc.).

To illustrate an optimal bit-packing and unpacking schema, this paper highlights the schema for the RNL-developed low-cost multi-band RF front-end, RadioLion. See Fig. 4 for a diagram of the RF front-end paired with PpRx. The RadioLion, the dual-antenna, triple-frequency front-end, uses 64-bit packets. The first 32 bits are composed of 4 samples, 2 bits, and 4 channels (i.e. L1 primary, L1 alternate, L2 primary, L2 alternate). At a given sampling instant, all four channels generate one sample each. Each sample is two-bit quantized. Thus, the RadioLion L1/L2 streams bit-packing generates 8-bit (1 Byte) of data at each sampling instant.

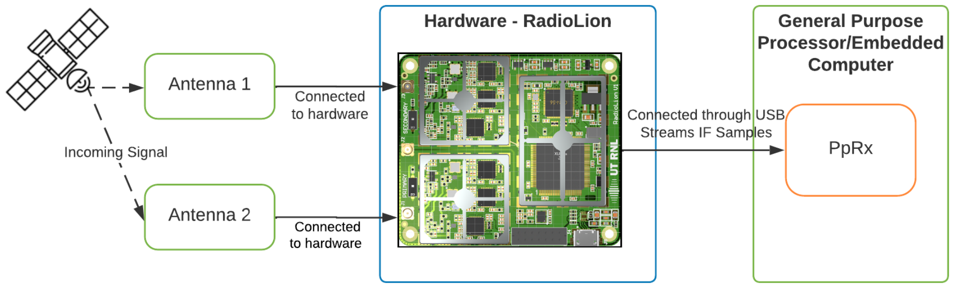


Fig. 4: This figure depicts an overview of the the hardware and software components that are needed for a fully functional receiver. First, two antennas are connected to the hardware (RadioLion). The RadioLion is pre-configured with the firmware and logic components needed to operate the board. When plugged in via USB to a computer, GRID/PpRx operates on the raw GNSS samples coming in from the RF front-end via USB. GRID/PpRx can operate in real-time or in post-processing mode, both of which provide a precise positioning solution.

The last 32 bits encode 8 samples, 2 bits, and 2 channels (i.e. L5 primary, L5 alternate). At a given sampling instant, the two channels generate one sample each. Each sample is two-bit quantized. The sampling rate of the second front-end is twice the sample rate of L1/L2. Thus, the RadioLion L5 stream generates 8-bit (1 Byte) of data every sampling instant.

Together, these individual RF front-ends within the hardware formulate a composite RF front-end. The output of this composite front-end is a single serial stream of data that is available to the SDR. As described above, the RadioLion is configured with two front-ends: one for L1/L2 and one for L5. Each front-end produces digitized data for a combination of the frequencies (L1, L2, L5) and antennas (primary or secondary). The digitized data is produced at a uniform sampling rate and with uniform quantization (e.g., 1-bit, 2-bit, etc.).

The L1/L2 half of RadioLion bit-packing has 4 samples per channel. When unpacking the L1/L2 part of RadioLion into the correlation buffers, the processor unpacks 4 bits at a time. The L5 half of the RadioLion bit-packing has 8 samples per channel. So, the processor can unpack the L5 part into the correlation buffers 8 bits at a time.

Packing of the RadioLion binary file is somewhat involved. The first byte in a RadioLion binary file corresponds to four samples from channel 0. Let $M_i^{t_k}$ and $S_i^{t_k}$ denote the magnitude and sign bit for the i^{th} channel at sample time t_k. The first byte is arranged as follows (left-most bit is most-significant)

$$\left[S_0^{t_k}\ S_0^{t_{k+1}}\ S_0^{t_{k+2}}\ S_0^{t_{k+3}}\ M_0^{t_k}\ M_0^{t_{k+1}}\ M_0^{t_{k+2}}\ M_0^{t_{k+3}}\right]$$

The first and fifth bits (from the left) form the earliest 2-bit sample, and the fourth and eighth bits (from the left) form the latest 2-bit sample.

Bytes 2, 3, and 4 in the binary file similarly correspond to channels 1, 2, and 3, respectively. These 4 Byte of data represent four samples from each of the four channels. The fifth and sixth byte in the binary file corresponds to channel 4. Let $M_i^{t_j}$ and $S_i^{t_j}$ denote the magnitude and sign bit for the i^{th} channel at sample time t_j, where the sample time is half that of t_k (likewise, twice the sampling frequency). The fifth byte is arranged as follows (left-most bit is most-significant)

$$\left[S_4^{t_j}\ S_4^{t_{j+1}}\ \ldots\ S_4^{t_{j+6}}\ S_4^{t_{j+7}}\ M_4^{t_j}\ M_4^{t_{j+1}}\ \ldots\ M_4^{t_{j+6}}\ M_4^{t_{j+7}}\right]$$

Bytes 7 and 8 in the binary file corresponds to samples from channel 5. This is because the second front-end is sampled at twice the sampling rate as the first front-end. Therefore, each L5 stream produces 2 Byte of data at every sampling instant of L1/L2. The pattern repeats for further samples.

In summary, PpRx unpacks the data from the six streams into 64-bit words. The 64-bit word is comprised of the L1/L2 streams and L5 streams; the L1/L2 streams consist of four sign bits and four magnitude bits and the L5 streams consist of eight sign bits and eight magnitude bits. The bits as they appear in the binary file are arranged as follows with the usual time-order (left-most bit is most-significant):

Bits 0–31:

$$\overbrace{[S_0^0\ldots S_0^3 M_0^0\ldots M_0^3]}^{\textit{Stream 0}}\overbrace{[S_1^0\ldots S_1^3 M_1^0\ldots M_1^3]}^{\textit{Stream 1}}\overbrace{[S_2^0\ldots S_2^3 M_2^0\ldots M_2^3]}^{\textit{Stream 2}}\overbrace{[S_3^0\ldots S_3^3 M_3^0\ldots M_3^3]}^{\textit{Stream 3}}$$

Bits 32–63:

$$\overbrace{[S_4^0\quad\ldots\quad S_4^7][M_4^0\quad\ldots\quad M_4^7]}^{\textit{Stream 4}}\overbrace{[S_5^0\quad\ldots\quad S_5^7][M_5^0\quad\ldots\quad M_5^7]}^{\textit{Stream 5}}$$

Or for further illustration, like this:

$$[S_0\ S_0\ S_0\ S_0\ M_0\ M_0\ M_0\ M_0][S_1\ S_1\ S_1\ S_1\ M_1\ M_1\ M_1\ M_1][S_2\ S_2\ S_2\ S_2\ M_2\ M_2\ M_2\ M_2][S_3\ S_3\ S_3\ S_3\ M_3\ M_3\ M_3\ M_3]\ldots$$
$$\ldots[S_4\ S_4\ S_4\ S_4\ S_4\ S_4\ S_4\ S_4\ M_4\ M_4\ M_4\ M_4\ M_4\ M_4\ M_4\ M_4][S_5\ S_5\ S_5\ S_5\ S_5\ S_5\ S_5\ S_5\ M_5\ M_5\ M_5\ M_5\ M_5\ M_5\ M_5\ M_5]$$

Despite the efficiency of bit-wise SDR, PpRx's overall processing capabilities can dramatically worsen if unpacking the data into the corresponding circular buffers becomes too processor intensive. For bit-wise SDRs, for efficient unpacking, each set of data contains extended runs of bits (sign or magnitude) stored in a particular output buffer. For example, the final serialized data contains a run of sign bits for a particular stream, then a run of magnitude bits from the same stream, and repeating for the rest of the streams from the RF front-end. This method proves to be quite efficient for unpacking into the correlation buffers for bit-wise parallel correlation as the bit-unpacking simply involves byte-wise "read" and "copy" operations [13].

Acquisition

GNSS signal acquisition can be thought of as a two-dimensional search in Doppler frequency and in pseudorandom code offset. The naïve approach to acquisition performs standard in-phase and quadrature signal correlation at each cell in the two-dimensional search space. This brute-force technique is simple to implement but is very computationally expensive, especially if one wishes to increase acquisition sensitivity by extending the coherent or non-coherent integration time. A approach first introduced in [16], exploits the tremendous computational efficiency of the Fast Fourier Transform (FFT) to search simultaneously all possible code offsets at a particular frequency. The result is a remarkably fast acquisition process.

PpRx uses the efficiency of this FFT function in the signal acquisition process. PpRx implements this FFT-based method using an extremely optimized C library, called FFTW [17], to perform signal acquisition. The FFTW library exploits SIMD intrinsics to optimize internal functions and processing speed. However, outside of the SIMD support in the library, PpRx does not implement additional SIMD instructions to optimize the FFT-based acquisition process.

Carrier Generation Optimization

In the processing chain, PpRx performs baseband mixing, which mixes the RF signal from a nominal IF down to baseband. To do so, any Doppler shift frequency on the signal is removed by either adding (or subtracting for high-side mixing) the Doppler shift frequency estimate to the nominal IF of the RF signal. The signals used for this baseband mixing are the generated carrier replicas. Carrier replicas are the time histories sine and cosine signals covering a range of Doppler shift frequencies and initial phase offsets. The sine signals represent the in-phase signals, whereas the cosine signals represent the quadrature signals [18].

For oversampled carrier generation, the naïve implementation of a "carrier generator" is as follows: given an initial phase and Doppler frequency, it generates sine/cosine samples by calling sine and cosine functions on demand, for each sample.

To reduce the computational burden of having to calculate the carrier replicas in real-time, the SDR precomputes the carrier replicas [18]. This is accomplished by using a set of look-up tables. The benefit of pre-calculated carrier replicas is that it is much faster than the naïve implementation. However, the cost of this pre-computation is that the generator does not generate an exact carrier output.

An overview of the generator is given in this paper, however, further details are provided in [18]. For bit-wise parallel SDR, the implementation below is applied to efficiently generate the oversampled carrier replicas.

The table look-up is used to convert the desired Doppler shift frequency and phase offset into the bit-wise parallel representations of the in-phase and quadrature carrier replicas needed for the carrier-mixing. Two tables are needed: one for the sign bits and one for the magnitude bits.

The quantized sampled carrier replica are represented in the bit-wise parallel format as a block of 32-bit words. In the simplest case, the carrier replicas are one-bit quantized with 0 and 1 respectively representing the values -1 and 1. Typically in PpRx, two-bit quantization for bit-wise parallel correlation is used, which means one bit represents the sign and the other represents the magnitude of the signal.

To create the look-up tables, the phase is quantized by a configurable amount. The quantization can be set low to increase processing speed via better cache performance (fewer cache misses); or set high to increase phase tracking accuracy. Most of PpRx's configurations use a quantization amount in the middle to achieve sub-millimeter phase accuracy.

To meet precision requirements needed for accurate carrier replica generation, the number of indices for the frequency and phase dimensions of the tables for 32-samples of sine is set to the number of Doppler search frequencies by the quantization amount of 2π radians of carrier phase [18]. With this table size, the pre-calculated table carrier replicas do not notably differ from carrier replicas generated through the naïve implementation by using the exact phase and frequency values.

However, each step may also be further optimized to use SIMD: same algorithm, different CPU instructions to implement it. This covers the general idea of naïve implementation, further optimized algorithmically, and further optimized by exploiting SIMD instruction sets on the optimized algorithm.

In the optimized algorithm, the SIMD instruction set accelerates the tabular look-up operations, as well as the generation of the table itself. The SIMD vectorized instruction sets parallelizes the computation and storage of the generated oversampled carrier replicas (the bit-wise parallel words) into the tables.

Code Generation Optimization

The oversampled code replicas are replicas of the spreading code for the particular signal, resampled to the sample rate of the RF samples in the circular buffers, and aligned to the frequency and code/phase offsets determined by the tracking loops (the delay-locked loop, frequency-locked loop, and phase-locked loops). The oversampled code generator nominally generates the spreading code of the signal.

Following the same concept introduced in the carrier generation section, the naïve implementation is further optimized algorithmically. The naïve implementation of “code generation” produces the spreading code time history for the local signal replica used to correlate against the received signal.

Similar to the carrier generation optimization as mentioned above, PpRx further optimizes the naïve method using pre-calculated look-up tables. An overview of the generator is given in this paper, however, further details are provided in [19].

The software correlator generates the C/A codes offline and stores them in memory, called the PRN code table. These stored codes are sampled at the RF front-end sampling frequency. The tabulated PRN codes translate a PRN code and its timing information into the bit-wise parallel format. The look-up table is based on the quantized offset of the initial prompt chip relative to the start of the code. The tables have two variable inputs: the sequence of code chip values and quantized offset of the initial prompt chip relative to the start of the code. PpRx works with both the prompt and early-minus-late (EML) replicas of a PRN code. The early chips are $0.5\ \Delta t_{eml}$ seconds before the prompt chips, and the late chips are $0.5\ \Delta t_{eml}$ seconds after the prompt chips.

There are three tables required for PRN code storage: (1) the prompt code, (2) the EML zero-mask word, and (3) the EML 2’s sign word.

The prompt code is stored as a single sign bit, where a 1 bit represents +1 and a 0 bit represents -1. The EML code, on the other hand, is stored in a 1.5-bit representation, with a sign bit and a zero-mask (magnitude) bit. The zero-mask bit is 0 if the EML code equals 0, and it is 1 if the code equals +2 or -2. The EML 2’s sign word is stored as a single bit, where the sign bit equals 1 if the EML code equals +2, and it equals 0 if the code equals -2. The 2’s sign bit is irrelevant if the corresponding zero-mask bit equals zero [19].

This method from the authors of [19] stores all possible PRN code chip sequences in a grid with possible code timing offsets in order to tabulate all possible variants of the sampled code.

Extensive profiling within PpRx up to this point has not yet demanded acceleration of the optimized algorithm through SIMD implementation, however, the look-up table method is subject to SIMD implementation in the future if see fit.

Correlation

The correlation process has three main stages. The first, it performs carrier-mixing, which mixes the authentic RF signal to baseband using the generated carrier replica. Next, it performs code-mixing, which mixes the baseband signal with a replica of the C/A code. Lastly, it performs accumulation, which sums the resulting signal over a C/A code period. See Fig. 5 for a block diagram view of the GNSS processing suite, including key elements such as correlation, code generation and carrier generation.

The naïve implementation requires a massive amount of bit-manipulation within the software correlators. In order to perform the carrier-mixing, the correlators multiply the input signal by the generated carrier replica, which is a complex exponential [5]. The generated carrier replica is stored in the pre-calculated look-up tables. The multiplication process results in a signal at baseband. A simple XOR multiplication of sign bits and storing the resultant data bits accomplishes baseband mixing. Correlation has greatly benefitted from the introduction of SIMD instruction sets as PpRx can use SIMD XOR to accelerate this first process.

The numerically controlled oscillator (NCO) carrier generator is governed by the phase-locked loop (PLL). The PLL’s goal in the correlation and tracking process is to continuously synchronize the phase and frequency of the generated carrier replica to the received signal’s carrier.

Next, to perform the code-mixing, the SDR needs to use the generated code replicas in memory and shift them in time until the replicas match up with the C/A code received from the satellite. See Fig. 6 for a simplified diagram of the time-shifting of the generated code replica. This is accomplished by using the autocorrelation function, which measures the similarity between the received waveform and time shifts of itself [20]. The time shifted signal used are the code generated replicas from the pre-calculated table look-up method. When the two codes match, there is a sharp and unique peak that is called

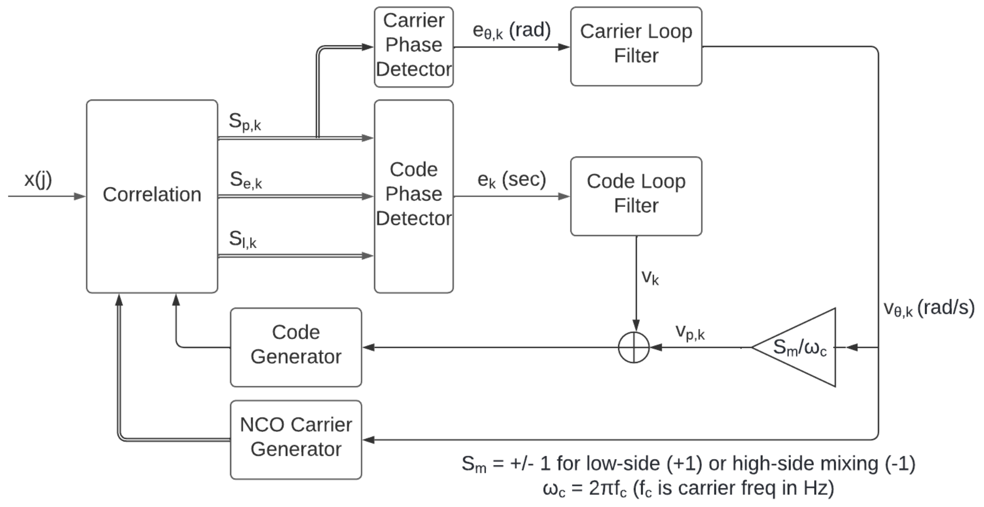


Fig. 5: GNSS software receiver architecture.

the correlation spike. From the autocorrelation process, the receiver can determine the clock offset from the number of chips it shifted the replica code by [20].

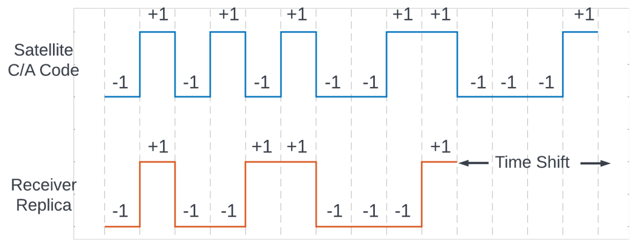


Fig. 6: Time-shift of generated code replica to match with the authentic signal's code.

Both prompt and EML correlations are needed to track the carrier frequency, carrier phase, and code phase in the SDR. The prompt and early-minus-late C/A code replicas are retrieved from memory in the generated PRN code table. The delay-locked loop (DLL) governs the code generator in the correlation process. The DLL's purpose is to control the code generator so that the PRN replica code is continuously time-aligned with the received code. To do this, it speeds up or slows down the code generator in the tracking loops to ensure alignment with the received signal. The code replicas are then mixed with the baseband signal by way of bit redefinitions and XOR operations. Similar to carrier-mixing, the exploitation of SIMD instruction sets used in the code-mixing has greatly improved the processing speed of this operation. The bit-wise parallel correlation uses a highly optimized sequence of SIMD instructions that operates on the separate circular buffers to correlate samples in parallel. The software correlators use SIMD XOR and table look-up operations. Every bit-mask, bitwise-or, and bit-shift is yet another instruction. Harnessing these new SIMD instruction sets, the processor performs several of these scalar instructions in a single instruction. See Fig. 8 for an 8-bit example of the bit-wise parallel method to compute the mixing and accumulation stages of the software correlators.

Lastly, the final step in the software correlation is accumulation. The accumulation operation serves to sum the results over the all the samples in a given PRN code period [5]. Thus requiring additional bit-wise parallel operations. Here, the SIMD popcount instruction is heavily exploited to count the number of bits in a long chunk of memory. PpRx relies on the SIMD-accelerated bit-wise parallel correlator implementation to achieve fast performance in the correlation operation. See Fig. 7 for a block diagram view of the software correlation process and Fig. 8 to view an 8-bit example of bit-wise parallel correlation using SIMD to accelerate the bit-manipulation.

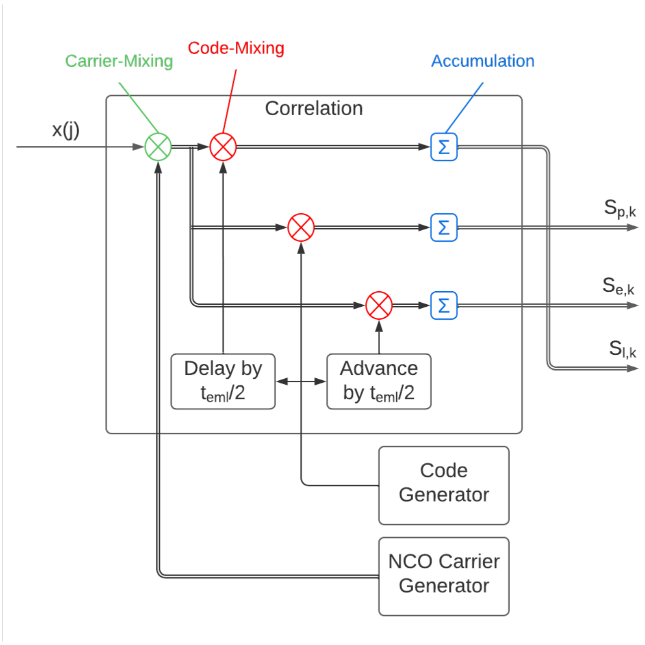
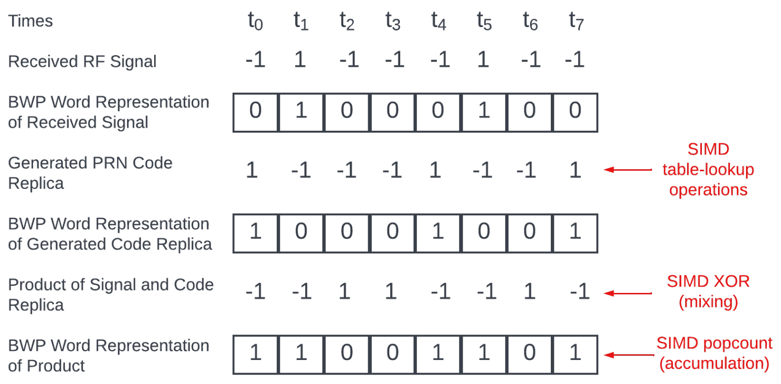


Fig. 7: Shown are the three steps in the correlation process: carrier-mixing, code-mixing, and accumulation. There is multiple correlators for early, prompt and late taps.



Times	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	
Received RF Signal	-1	1	-1	-1	-1	1	-1	-1	
BWP Word Representation of Received Signal	0	1	0	0	0	1	0	0	
Generated PRN Code Replica	1	-1	-1	-1	1	-1	-1	1	SIMD table-lookup operations
BWP Word Representation of Generated Code Replica	1	0	0	0	1	0	0	1	
Product of Signal and Code Replica	-1	-1	1	1	-1	-1	1	-1	SIMD XOR (mixing)
BWP Word Representation of Product	1	1	0	0	1	1	0	1	SIMD popcount (accumulation)

Fig. 8: This figure depicts the mixing stage of the authentic RF signal with the generated PRN code replica. The SIMD instructions used in the correlation process include using SIMD-accelerated table look-up operations, bit-wise XOR, and popcount in order to process bit-wise parallel (BWP in the figure) word representations.

In PpRx, correlation follows the same optimization pattern as introduced in the two previous sections: the naïve implementation is further optimized by exploiting SIMD instruction sets.

RESULTS

Harnessing the latest SIMD instruction sets, PpRx's performance has recently achieved a remarkable inflection point: under some processing configurations, the correlation operation, by which each channel's signal is mixed to baseband and de-spread via multiplication against local code and carrier replicas, is no longer the bottleneck process. SIMD instruction sets are ideal for calculations with high data parallelism, such as the correlation operation. This is because the SIMD instruction sets can operate on multiple samples in the same cycle. As noted by RNL-member Zachary Clements in [13] "Bit-wise parallel correlation amounts to a highly optimized sequence of SIMD instructions that operate on the separate circular buffers to correlate 2^p samples in parallel against local code and carrier replicas. For the latest variant of RNL's GRID receiver, $p = 7$, meaning that 128 samples are correlated in parallel."

The correlation operation not being the bottleneck process represents a major milestone for software-defined GNSS and paves the way for commercialization opportunities. This is the very operation that Philip Ward warned in 1996 could make general purpose processors perpetually unsuitable for GNSS signal processing is now so efficiently implemented that it requires less than half the computational resources. See Fig. 9 to show an overview of the different PpRx profiles used where the correlation operation is not the bottleneck process. Further detailed breakdowns are shown in the Appendix Figs. A.1 and A.2.

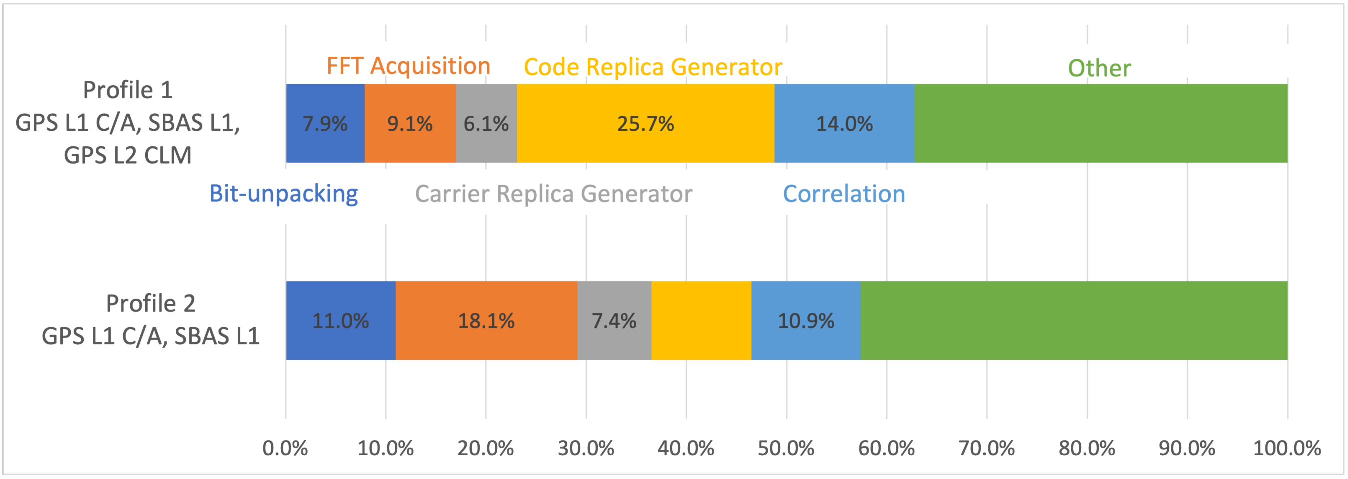


Fig. 9: This figure depicts the various computational power requirements, represented as percentages of the overall process, that each element demands in the overall processing chain. Profile 1 depicts a configuration that includes GPS L1 C/A, SBAS L1, and GPS L2 CLM. Under this configuration, correlation is so highly optimized at this point that PpRx is now spending more time outside of correlation than in it. In this case the Psiaki oversampled code generator is the bottleneck process when GPS L2 CLM is involved. Profile 2 depicts a configuration that includes GPS L1 C/A and SBAS L1 (no L2). Under this configuration, PpRx is nearly Pareto optimal: there is no single bottleneck process involved.

In addition, the speed of PpRx on various computational platforms is shown in Table I. The speed test shows results for live-recorded (not simulated) signals captured through a dual-antenna, dual-frequency RF front-end board. The recorded signals, both L1 and L2, have single-bit quantization and the the IF sampling rate for this configuration is 9.6 Msps. To generate the table, PpRx was configured to acquire and track only GPS L1 C/A signals for 8 satellites using up to 4 parallel execution threads. PpRx can post-process this captured data with remarkable speed to provide a positioning solution. See Fig. I for how much faster it takes each target platform to post-process the data compared to real-time operation. Further, the time indicated in the table represents the time PpRx took to post-process the full file, which is 200 seconds worth of data.

Target Platform	Odroid C2	Odroid C4	Odroid XU4	Intel Celeron N3161	Intel i5 1.3GHz	Intel Xeon 2.27GHz	Intel i7-6700HQ	Intel Xeon(R) Gold 6226R 2.9GHz
Time to Process (seconds)	4.06	3.38	3.90	3.75	2.09	1.47	0.89	0.32
Speed Multiplier of Post-Processing Compared to Real-Time Operation (rounded up to the nearest whole number)	50x	60x	52x	54x	96x	137x	225x	625x

TABLE I: Speed test of live-recorded GPS L1 C/A single-bit signals captured through a dual-frequency RF front-end board on various target processing platforms. The IF sampling rate for this configuration was 9.6 Msps. PpRx was configured to track only GPS L1 C/A for 8 satellites and up to 4 parallel execution threads.

USE CASES

Traditionally, one of the major limitations of GNSS SDR has been the higher power consumption and challenging computational requirements demanded by the signal acquisition and tracking process. It is still more power hungry compared to ASIC-based GNSS processing, but for many applications the flexibility of SDR is worth the watts.

For example, aerial and space-based applications require highly flexible and capable GNSS receivers due to their higher dynamic conditions and lower signal-to-noise ratios. In this era of emerging space-tech, the need for secure, robust, and precise GNSS is becoming increasingly more apparent as many of these technologies rely on GNSS as their primary sensing modality. Drones, satellites, autonomous aircraft, and spacecraft all rely on reliable positioning from external sensors, but especially GNSS, as it is key to ensuring safe, all-weather navigation for both consumer and commercial applications. PpRx has been licensed through the Radionavigation Lab to multiple commercial companies, but notably a major aerospace company that uses the technology across their suite of advanced spacecraft and satellites. The SDR is deployed across the company's mega-constellation of satellites used for broadband Internet. Ownership over a GNSS software-suite, rather than relying on external GNSS solutions providers was a key asset for this aerospace company to vertically integrate and bring their PNT suite in-house.

In addition to space-applications, one of the most notable market trends is the use of cheap RTK in mainstream devices. A GPS/GNSS receiver capable of RTK processes the GNSS signals along with a correction stream (correction data from static reference stations) to achieve centimeter-level positional accuracy. This technique that provides precise positioning is being increasingly used across the industry:

1) Bosch is building an integrated GNSS-IMU system for automated vehicles [21]
2) Bird is using precise positioning in their scooters [22]
3) Segway's answer is virtual geofencing, using a mixture of GPS and other sensors. It combines GPS and other sensor types to get down to 2 cm positioning accuracy [23]

RadioLion paired with PpRx and another implementation of GRID, called PpEngine, use RTK to obtain centimeter-accuracy in the navigation solution [11], [12]. As seen above, this technique is in higher demand as the abundance of automated and autonomous technologies increases. Automated vehicles that are equipped with large batteries can sufficiently supply the SDR and demand the cutting-edge signal processing technology that comes with it. GNSS SDR is a perfectly suitable solution for applications where the general purpose processor or embedded platform has enough physical power to supply the additional computational power needed for the GNSS processing.

In addition to the applications listed above, wall-mounted technology that needs access to a high performance GNSS solution would be a worthy candidate for GNSS SDR. This is because the utilization of electrical outlets makes the application practically insensitive to power consumption. Applications here could fall into the research and development space, or act as stand-alone static devices, such as a reference station.

Further, PpRx has also been in use across multiple industries over the past few years, including aerospace, defense, science applications, and emerging R&D. To highlight one example, PpRx has been operating on the International Space Station (ISS) in collaboration with the Naval Research Lab since 2017. Its original purpose was a part of a larger experiment for radio occultation, however it more recently has been able to detect transmitted interference sources from the ground [10].

Software-defined GNSS is ready to launch from research applications to mainstream commercial devices. Locus Lock, a start-up spinning out of the Radionavigation Lab, is commercializing SDR for mass market applications. Locus Lock packages the hardware platform (RadioLion) with GRID/PpRx to provide end-users with a fully functional receiver. This offering is a key asset for agile and assured PNT. Locus Lock's mission is to offer centimeter-accurate real-time positioning to ensure globally available, high integrity positioning at a fraction of the cost of current market GNSS solutions.

CONCLUSIONS

Advancements in processor speeds, parallel instructions, and architectures have unlocked expanded possibilities for GNSS SDR. Harnessing the latest SIMD instruction sets, GRID/PpRx's performance achieved a remarkable inflection point: under some processing configurations, the correlation operation was no longer the bottleneck process. Every step along the processing chain has been exploited for optimization, including data loading, bit-unpacking, acquisition, correlation, tracking loops, etc. This achievement in reducing the burden of the correlation operation on the overall processing chain has been an important milestone for software-defined GNSS.

This paper has described a state-of-the-art "pure" GNSS SDR based on bit-wise parallel correlation. In addition, this paper has offered an explanation of how modern processors and vectorized instruction sets have enabled GNSS SDR to be exquisitely efficient in parallel operations and bit-manipulation. Lastly, this paper has offered an exploration into use cases that are particularly well suited for software-defined radio. These achievements thus far in GNSS SDR, and the ripe market for low-cost and precise GNSS technology, has enabled GNSS SDR to be ready for launch into commercial applications.

ACKNOWLEDGMENTS

Research support was provided by the U.S. Department of Transportation (USDOT) under the University Transportation Center (UTC) Program Grant 69A3552047138 (CARMEN), and by affiliates of the 6G@UT center within the Wireless Networking and Communications Group at The University of Texas at Austin. Additional support was provided through the Texas Innovation Center and through the Launch Texas fellowship.

APPENDIX

REFERENCES

[1] Akos, D. M. and Braasch, M. S., "A Software Radio Approach to Global Navigation Satellite System Receiver Design," *Proceedings of the 52rd Annual Meeting of The Institute of Navigation*, Cambridge, MA, 1996, pp. 455–463.

[2] Akos, D. M., Normark, P., Enge, P., Hansson, A., and Rosenlind, A., "Real-Time GPS Software Radio Receiver," *Proceedings of the ION National Technical Meeting*, Institute of Navigation, Long Beach, CA, Jan. 2001, pp. 809–816.

[3] Ledvina, B. M., Psiaki, M. L., Sheinfeld, D., Cerruti, A. P., Powell, S. P., and Kintner, P. M., "A real-time GPS civilian L1/L2 software receiver," *Proc. of the Institute of Navigation GNSS*, 2004, pp. 21–24.

[4] Trimble, "Trimble Catalyst," June 2022, https://geospatial.trimble.com/products-and-solutions/trimble-catalyst.

[5] Ledvina, B. M., Psiaki, M. L., Powell, S. P., and Kintner, Jr., P. M., "Bit-Wise Parallel Algorithms for Efficient Software Correlation Applied to a GPS Software Receiver," *IEEE Transactions on Wireless Communications*, Vol. 3, No. 5, Sept. 2004.

[6] Humphreys, T. E., Bhatti, J., Pany, T., Ledvina, B., and O'Hanlon, B., "Exploiting multicore technology in software-defined GNSS receivers," *Proceedings of the ION GNSS Meeting*, Institute of Navigation, Savannah, GA, 2009, pp. 326–338.

[7] Humphreys, T. E., Ledvina, B. M., Psiaki, M. L., O'Hanlon, B. W., and Kintner, Jr., P. M., "Assessing the spoofing threat: Development of a portable GPS civilian spoofer," *Proceedings of the ION GNSS Meeting*, Institute of Navigation, Savannah, GA, 2008.

[8] Lightsey, E. G., Humphreys, T. E., Bhatti, J. A., Joplin, A. J., O'Hanlon, B. W., and Powell, S. P., "Demonstration of a Space Capable Miniature Dual Frequency GNSS Receiver," *Navigation*, Vol. 61, No. 1, Mar. 2014, pp. 53–64.

[9] Pesyna, Jr., K. M., Heath, Jr., R. W., and Humphreys, T. E., "Centimeter Positioning with a Smartphone-Quality GNSS Antenna," *Proceedings of the ION GNSS+ Meeting*, 2014.

[10] Murrian, M. J., Narula, L., Iannucci, P. A., Budzien, S., O'Hanlon, B. W., Powell, S. P., and Humphreys, T. E., "First Results from Three Years of GNSS Interference Monitoring from Low Earth Orbit," *Navigation, Journal of the Institute of Navigation*, Vol. 68, No. 4, 2021, pp. 673–685.

[11] Humphreys, T. E., Murrian, M. J., and Narula, L., "Deep-Urban Unaided Precise Global Navigation Satellite System Vehicle Positioning," *IEEE Intelligent Transportation Systems Magazine*, Vol. 12, No. 3, 2020, pp. 109–122.

[12] Yoder, J. E. and Humphreys, T. E., "Low-Cost Inertial Aiding for Deep-Urban Tightly-Coupled Multi-Antenna Precise GNSS," *Navigation, Journal of the Institute of Navigation*, 2022, To be published.

[13] Clements, Z., Iannucci, P. A., Humphreys, T. E., and Pany, T., "Optimized Bit-Packing for Bit-Wise Software-Defined GNSS Radio," *Proceedings of the ION GNSS+ Meeting*, St. Louis, MO, 2021.

[14] Dampf, J., Pany, T., Bär, W., Winkel, J., Stöber, C., Fürlinger, K., Closas, P., and Garcia-Molina, J., "More than we ever dreamed possible: Processor technology for GNSS software receivers in the year 2015," *Inside GNSS*, Vol. 10, No. 4, 2015, pp. 62–72.

[15] Rupp, K., "50 Years of Microprocessor Trend Data," February 2018, https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.

[16] Van Nee, D. and Coenen, A., "New fast GPS code-acquisition technique using FFT," *Electronics Letters*, Vol. 27, No. 2, 1991, pp. 158–160.

[17] Frigo, M. and Johnson, S. G., "FFTW," http://fftw.org.

[18] Ledvina, B., "Efficient Real-Time Generation of Bit-Wise Parallel Representations of Oversampled Carrier Replicas," *Aerospace and Electronic Systems, IEEE Transactions on*, Vol. 47, No. 4, Oct. 2011, pp. 2921–2933.

[19] Psiaki, M. L., "Real-Time Generation of Bit-Wise Parallel Representations of Over-Sampled PRN Codes," *IEEE Transactions on Wireless Communications*, Vol. 5, No. 3, March 2006, pp. 487–491.

[20] Misra, P. and Enge, P., *Global Positioning System: Signals, Measurements, and Performance*, Ganga-Jumana Press, Lincoln, Massachusetts, revised second ed., 2012.

[21] Bosch, "Vehicle motion and position sensor," 2021, https://www.bosch-mobility-solutions.com/en/solutions/sensors/vehicle-motion-and-position-sensor/.

[22] Bosch, "Bird vehicles can now accurately detect sidewalk riding and slow riders to a stop," October 2021, https://techcrunch.com/2021/10/13/bird-vehicles-can-now-accurately-detect-sidewalk-riding-and-slow-riders-to-a-stop/.

[23] Davies, C., "Segway built a smarter robot lawnmower," September 2021, https://www.slashgear.com/segway-built-a-smarter-robot-lawnmower-03689488/.

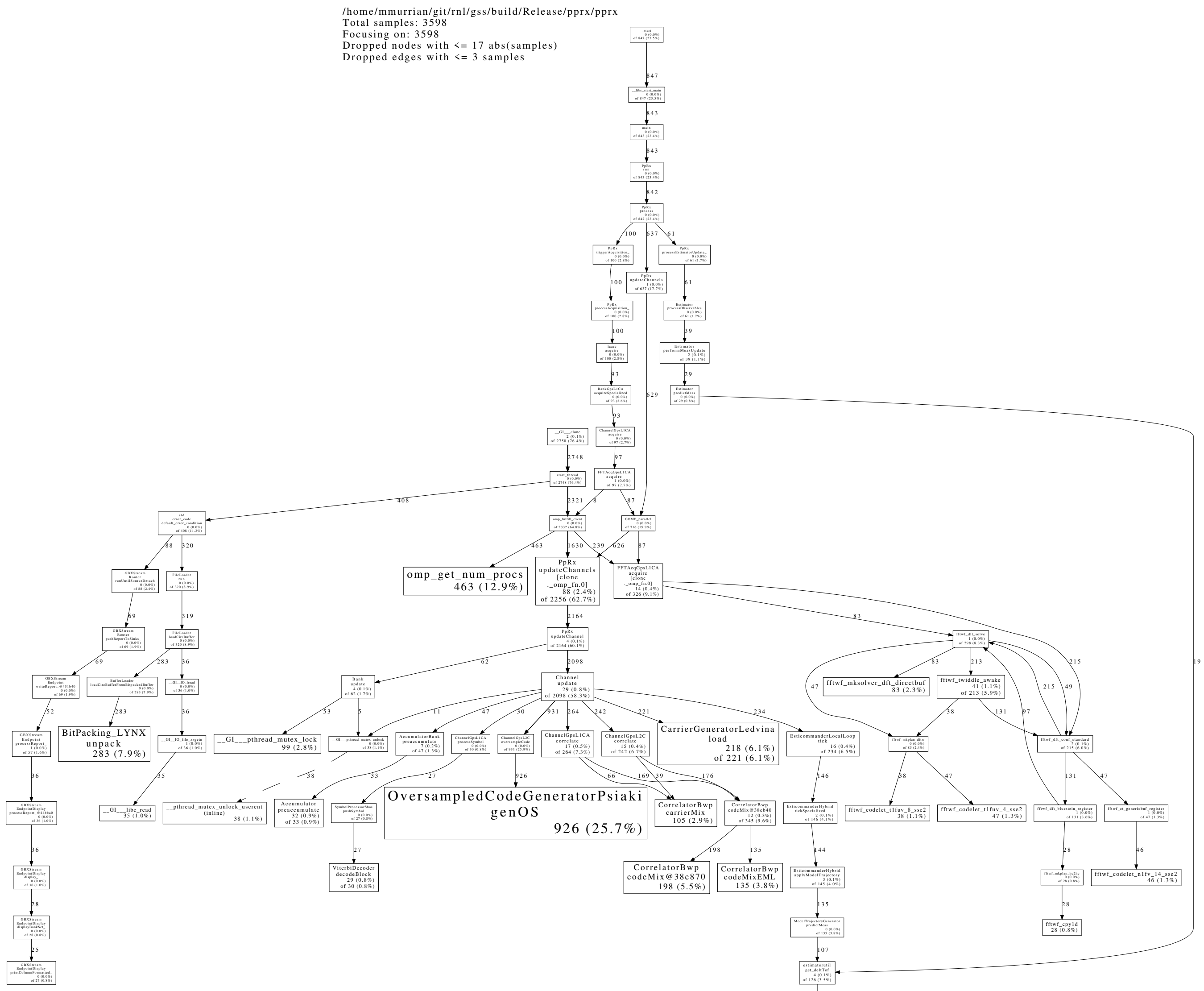
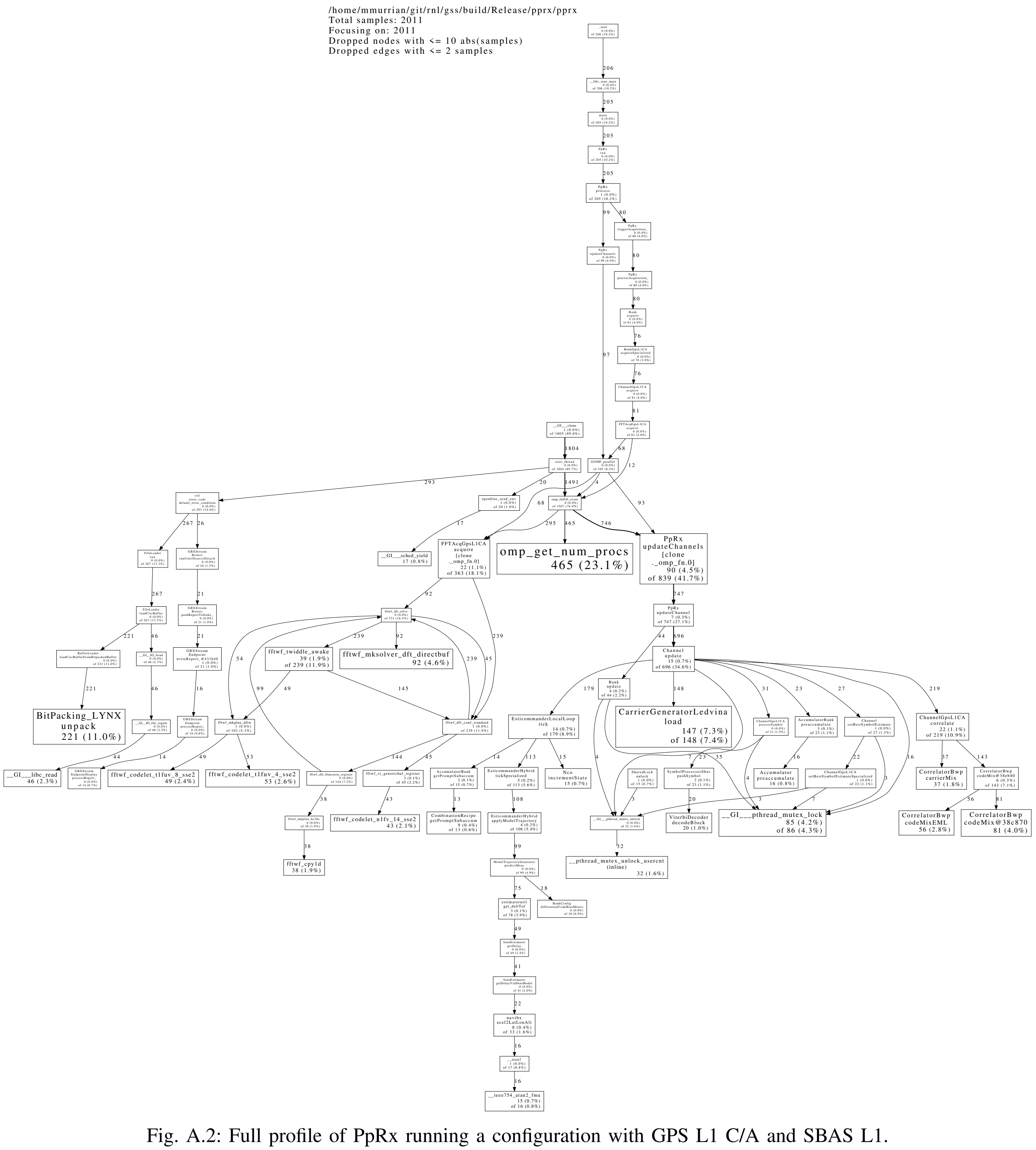

/home/mmurrian/git/rnl/gss/build/Release/pprx/pprx
Total samples: 3598
Focusing on: 3598
Dropped nodes with <= 17 abs(samples)
Dropped edges with <= 3 samples
omp_get_num_procs 463 (12.9%)
PpRx updateChannels [clone ._omp_fn.0] 88 (2.4%) of 2256 (62.7%)
BitPacking_LYNX unpack 283 (7.9%)
__GI___pthread_mutex_lock 99 (2.8%)
CarrierGeneratorLedvina load 218 (6.1%) of 221 (6.1%)
OversampledCodeGeneratorPsiaki genOS 926 (25.7%)
CorrelatorBwp carrierMix 105 (2.9%)
CorrelatorBwp codeMix@38c870 198 (5.5%)
CorrelatorBwp codeMixEML 135 (3.8%)
fftwf_mksolver_dft_directbuf 83 (2.3%)
fftwf_twiddle_awake 41 (1.1%) of 213 (5.9%)
fftwf_codelet_t1fuv_8_sse2 38 (1.1%)
fftwf_codelet_t1fuv_4_sse2 47 (1.3%)
fftwf_codelet_n1fv_14_sse2 46 (1.3%)
__pthread_mutex_unlock_usercnt (inline) 38 (1.1%)
Accumulator preaccumulate 32 (0.9%) of 33 (0.9%)
ViterbiDecoder decodeBlock 29 (0.8%) of 30 (0.8%)
__GI___libc_read 35 (1.0%)
fftwf_cpy1d 28 (0.8%)

Fig. A.1: Full profile of PpRx running a configuration with GPS L1 C/A, SBAS L1, and GPS L2 CLM.



/home/mmurrian/git/rnl/gss/build/Release/pprx/pprx
Total samples: 2011
Focusing on: 2011
Dropped nodes with <= 10 abs(samples)
Dropped edges with <= 2 samples
omp_get_num_procs 465 (23.1%)
PpRx updateChannels [clone ._omp_fn.0] 90 (4.5%) of 839 (41.7%)
FFTAcqGpsL1CA acquire [clone ._omp_fn.0] 22 (1.1%) of 363 (18.1%)
BitPacking_LYNX unpack 221 (11.0%)
CarrierGeneratorLedvina load 147 (7.3%) of 148 (7.4%)
fftwf_twiddle_awake 39 (1.9%) of 239 (11.9%)
fftwf_mksolver_dft_directbuf 92 (4.6%)
__GI___pthread_mutex_lock 85 (4.2%) of 86 (4.3%)
CorrelatorBwp codeMix@38c870 81 (4.0%)
CorrelatorBwp codeMixEML 56 (2.8%)
CorrelatorBwp carrierMix 37 (1.8%)
ChannelGpsL1CA correlate 22 (1.1%) of 219 (10.9%)
EsticommanderLocalLoop tick 14 (0.7%) of 179 (8.9%)
Channel update 15 (0.7%) of 696 (34.6%)
PpRx updateChannel 7 (0.3%) of 747 (37.1%)
fftwf_codelet_t1fuv_4_sse2 53 (2.6%)
fftwf_codelet_t1fuv_8_sse2 49 (2.4%)
__GI___libc_read 46 (2.3%)
fftwf_codelet_n1fv_14_sse2 43 (2.1%)
fftwf_cpy1d 38 (1.9%)
__pthread_mutex_unlock_usercnt (inline) 32 (1.6%)
__GI___sched_yield 17 (0.8%)
Accumulator preaccumulate 16 (0.8%)
Nco incrementState 15 (0.7%)
ViterbiDecoder decodeBlock 20 (1.0%)
__ieee754_atan2_fma 15 (0.7%) of 16 (0.8%)

Fig. A.2: Full profile of PpRx running a configuration with GPS L1 C/A and SBAS L1.